JOURNAL OF AEROSPACE COMPUTING, INFORMATION, AND COMMUNICATION
Vol. 3, June 2006

Reverse Engineering of Software Life Cycle Data in Certification Projects

Leanna Rierson* and Barbara Lingberg†
Federal Aviation Administration (FAA)

Some applicants, developers, and commercial-off-the-shelf (COTS) software vendors have proposed reverse engineering as an approach for satisfying RTCA/DO-178B objectives for airborne software. RTCA/DO-178B, *Software Considerations in Airborne Systems and Equipment Certification*, serves as the means of compliance for most airborne software in civil aircraft. DO-178B defines reverse engineering as *"the method of extracting software design information from the source code"* and provides guidance particular to reverse engineering, when it is used to upgrade a development baseline.[1]

For purposes of this paper, reverse engineering is an approach for creating software life cycle data that did not originally exist, cannot be found, is not adequate, or is not available to a developer in order to meet applicable DO-178B objectives. Reverse engineering is not just the generation of data–rather it is a process to assure that the data is correct, the software functionality is understood and well documented, and the software functions as intended and required by the system. Reverse engineering is not, as some software developers propose, just an effort to generate the software life cycle data without intent to *build in quality* and the resulting design assurance.

This paper explores reverse engineering in airborne software projects, by explaining a definition for the certification domain, describing the motivation for its use, and documenting the certification concerns. Two actual cases of reverse engineering are also described to illustrate the certification concerns in real projects.

Introduction

SOME applicants, developers, and commercial-off-the-shelf (COTS) software vendors have proposed reverse engineering as an approach for satisfying DO-178B objectives for airborne software. DO-178B defines reverse engineering as: *"the method of extracting software design information from the source code"*.[1] Section 12.1.4 of DO-178B addresses "Upgrading a Development Baseline," which may be implemented using reverse engineering. DO-178B paragraphs 12.1.4.d and 12.1.4.f provide guidance particular to reverse engineering.

Paragraph 12.1.4.d states: *"Reverse engineering may be used to regenerate software life cycle data that is inadequate or missing in satisfying the objectives of this document. In addition to producing the software product, additional activities may need to be performed to satisfy the software verification process objectives"*.[1]

Paragraph 12.1.4.f states that the developer *"should specify the strategy for accomplishing compliance with this document in the Plan for Software Aspects of Certification"*.[1]

Received 20 April 2004; revision received 29 July 2004; accepted for publication 1 October 2004. Based on "Reverse Engineering of Software Life Cycle Data In Certification Projects" by Leanna Rierson and Barbara Lingberg, Presented at the 21st Digital Avionics Systems Conference © [2002] IEEE.
* Leanna Rierson, Chief Scientific and Technical Advisor, Federal Aviation Administration; 800 Independence Ave, SW; Washington, D.C. 20591. Barbara Lingberg, Software Program Manager, Federal Aviation Administration; 800 Independence Ave, SW; Washington, D.C. 20591. This paper does not represent the official Federal Aviation Administration (FAA) position. The authors are FAA employees, and while the paper is intended to be consistent with FAA policy, it has not been coordinated through the FAA's approving officials and represents solely the opinions of its authors.
† Federal Aviation Administration, Washington, D.C. 20591

This paper answers some of the common questions about reverse engineering by exploring:
- What is reverse engineering?
- What motivates the use of reverse engineering?
- What examples of reverse engineering have been encountered in civil aviation?
- What are the certification concerns regarding reverse engineering?

What is Reverse Engineering?

Roger Pressman's definition of reverse engineering supplements DO-178B's definition. Pressman defines reverse engineering for software as *"the process of analyzing a program in an effort to create a representation of the program at a higher level of abstraction than source code. Reverse engineering is a process of design recovery"*.[2]

For a civil aircraft certification project, reverse engineering is a type of re-engineering where software life cycle data that did not originally exist, cannot be found, is inadequate, or is not available to a developer in order to satisfy the applicable DO-178B objectives is created. It is not just the generation of data—rather it is a process of assuring that the data is correct with respect to the original data and intended functions, the software functionality is understood and well documented, and the software functions as intended and required by the system. It involves recovery of requirements and design, as well as conducting the relevant verification, quality assurance, and configuration management activities to the appropriate level to ensure integrity of the software, to ensure all software life cycle data is available and correct, and that an appropriate level of design assurance is achieved. It is not, as some developers propose, just an effort to create the software life cycle data without intent to *build in quality* and the resulting design assurance.

What Motivates Use of Reverse Engineering?

A number of developers desire to use reverse engineering in order to use existing software in their airborne application. This is typically non-aviation software and, in many cases, only the source code exists. However, in some cases, other software life cycle data may exist but is incomplete or inadequate. If properly applied, a reverse engineering approach can allow a developer to gain the necessary assurance for airborne software, create any missing software life cycle data, obtain an appropriate level of design assurance for the software, and provide a baseline for future use.

Software selected for reverse engineering typically has:
- a mature and stable version that has been used in a number of applications and demonstrated to be of high quality,
- functionality that, in whole or part, is useful to the user,
- been developed either to no standards at all or to other standards (e.g., military standards), but doesn't satisfy the DO-178B objectives, and
- been developed as a COTS product.

Some of the potential or perceived advantages that motivate developers to pursue reverse engineering include:
- Use of existing software can be more cost and schedule effective than developing new software.
- An operating software program represents a working design even if it has not been documented.
- The data package from a reverse engineering project may be reused and may serve as the foundation for future projects and products.

What Observations have been Made in Actual Reverse Engineering Projects?

During the past two years, the authors have been involved in several reverse engineering projects. Two cases involving real-time operating systems (RTOS) are discussed below. The company names have been removed and proprietary details sanitized. Both positive and negative observations when using reverse engineering are identified.

CASE 1: Company 1, Using RTOS A

Company 1 is an avionics manufacturer who made the decision to use COTS RTOS A in their avionics project. RTOS A has been used in many non-aviation projects but did not have the required software life cycle data to satisfy applicable DO-178B objectives. Company 1 contracted with Company X to reverse engineer the RTOS A to satisfy DO-178B Level C objectives.

Unfortunately, Company 1 did not have good control of Company X. Additionally, they failed to establish communication with the company. As a result, the package delivered to Company 1 by Company X was incomplete and inaccurate. Therefore, Company 1 ended up using Company X's work as a starting point and completing the reverse engineering effort themselves.

A number of observations were made in this particular case:

- When subcontracting with a company to perform reverse engineering, it is important to have clearly stated expectations that are mutually accepted. Additionally, continual oversight is necessary to ensure that the expectations are being fulfilled. Company 1's failure to follow these principles created a number of avoidable problems (e.g., inadequate testing and poor traceability).
- The specifications and interfaces between the RTOS and the avionics system was unclear. For example, traceability to system level requirements was incomplete. This led to inaccurate or incomplete functionality and code that did not trace to requirements.
- Company 1 had little insight into certain aspects of the RTOS functionality. This lack of understanding led to implementation problems such as stack overflow and timing deadlines that were not met.
- Many parts of the RTOS did not meet requirements of a safety critical aviation system (e.g., deterministic and fail-safe); therefore, a subset of the RTOS was proposed late in the program. This resulted in minimal use of the original RTOS functionality and project schedule overruns.
- Company 1 attempted to merge the high-level and low-level requirements for the RTOS into a single level. This caused problems (e.g., requirements traceability and verifiability) because the *what* and the *how* were mixed.
- The requirements were written as pseudocode and were difficult to understand. This caused considerable problems for the testing activities of the project. For example, many of the requirements were not verifiable. Therefore, Company 1 had to rewrite requirements before they could be successfully verified. Additionally, Company 1 had to rewrite the test cases and procedures.

Overall, this reverse engineering effort was a failure that resulted in significant rework by Company 1, severe limitations on the RTOS A functionality, and cost and schedule overruns.

CASE 2: Companies 2, 3, 4, Using RTOS B

COTS Company Y reverse engineered RTOS B that was slated for use by avionics Companies 2, 3, and 4. A safe-subset of RTOS B was identified by Company Y and reverse engineered to meet applicable DO-178B objectives. Some observations from this effort are:

- Companies 2, 3, and 4 used different processors and required slightly different features of RTOS B. Some parts of RTOS B data were reused and some parts were not. This meant that three versions of RTOS B existed. The impact of changes for the multiple versions was difficult to evaluate.
- Company Y had a well-planned reverse engineering process, but it was not being followed.
- A DO-178B compliance review resulted in many compliance findings. After the review, Company Y made efforts to address the findings, began to follow their process, and eventually satisfied the applicable DO-178B objectives.
- The interface between Companies 2, 3, and 4 system requirements and the reverse engineered RTOS B requirements were not well defined and led to traceability issues.
- The avionics companies who worked closely with Company Y were more successful than those that did not.
- The RTOS B reverse engineering and the systems development by Companies 2, 3, and 4 occurred simultaneously. This led to a number of configuration management challenges, because multiple baselines had to be maintained.
- Since RTOS B was changing, the RTOS user's manual was not up-to-date. Therefore, Companies 2, 3, and 4 did not have full insight into all RTOS functions, changes, and error messages. This caused considerable concern for the aviation companies and the certification authorities since it meant, for example, that code could be left untested or not tested for robustness.
- COTS Company Y originally attempted to go from code to low-level requirements at the same time that the high-level requirements were being developed from the user's manual. However, the low-level and high-level requirements developed in this manner didn't match well.

- COTS Company Y ended up reverse engineering the life cycle data (code–> design–> high-level requirements) and performing a forward verification (reviewing high-level requirements–> design–> code). The backwards development of the life cycle data and forwards verification worked well.

The reverse engineering project in this second case started out on relatively shaky ground. However, COTS Company Y's commitment to compliance with applicable DO-178B objectives led to a successful project. The overall key to this project's success was well-defined processes and adherence to those processes.

What are Certification Concerns in Reverse Engineering?

The two cases described above provide insight into some real-life reverse engineering projects that have been reviewed by certification authorities. These and other projects have raised several certification concerns regarding reverse engineering of airborne software. These concerns are summarized below. Developers should be proactive in addressing these concerns in their projects.

Concern 1: Lack of a Well-Defined Process

A key to successful reverse engineering is a well-defined reverse engineering process. Generating the necessary software life cycle data does not just happen–it must be planned, like any other software development effort. A common downfall of reverse engineering projects seen to date has been poor or non-existent planning data.

Reverse engineering can be considered a life cycle model, going from code to design to requirements. The processes and activities in that life cycle and the transition criteria between those processes and activities should be well defined in the software plans (i.e., the Plan for Software Aspects of Certification, Software Development Plan, Software Configuration Management Plan, Software Quality Assurance Plan, and Software Verification Plan). The plans and standards should clearly define how the DO-178B objectives will be satisfied through the reverse engineering effort.

A well-defined reverse engineering process helps ensure that the quality is built into the software. Of course, the plans and standards should also be understood and followed by the development team. For new development teams, this may require special training and oversight.

Concern 2: Insufficient Support for Satisfying DO-178B Objectives

Some companies have proposed reverse engineering projects without adequate justification for how safety objectives will be met. Certification authorities have observed some of the following problems:

- Failure to satisfy many DO-178B objectives in the reverse engineering effort.
- Use of reverse engineering just to satisfy a list of data items, rather than to ensure the design assurance and quality of the product and its adequacy for an airborne application.

Reverse engineering should be used cautiously and only in well-justified cases (i.e., for a project that has been used in a number of applications and has shown itself to be of high integrity). The use of reverse engineering in new software development (e.g., reverse engineering a prototype) is strongly discouraged by the certification authorities.

Concern 3: Lack of Access to Experts and Original Developers

Developing the design, requirements, and test cases for a complex software component, such as an operating system, can be nearly impossible without some access to the original developers. Without expertise in the domain being reverse engineered, the ability to accurately determine the software functionality is questionable and can be difficult to determine. This can result in an inconsistent and incorrectly derived product that will not meet the DO-178B objectives.

In the most successful reverse engineering efforts, the original developers have been contacted in order to gain a thorough understanding of the software functionally, particularly in difficult or complex areas. Every effort should be made to gain access to the original developers or to hire people with expertise in the specific domain area (e.g., operating system experts for an RTOS project).

In addition, particularly when a party other than the developer is undertaking the reverse engineering effort, communication between all parties helps to ensure success.

Concern 4: Complex or Poorly Documented Source Code

Many reverse engineering efforts start with source code that is complex or poorly documented. The code may contain numerous pointers and complex data structures. The code also may not contain comments, which can make it difficult to understand.

Developers should consider the condition of the code before starting a reverse engineering effort. Complex or poorly documented code may make satisfying the DO-178B objectives and aviation software policy difficult or impossible. Poorly documented or complex code will make it difficult for the reverse engineering team to assess what the code was intended to do and to develop complete and accurate requirements and design data. The verification effort will also be difficult to complete with poorly documented or complex code.

A thorough understanding of the code is essential to successful reverse engineering. Poorly documented or complex code is not a good candidate for reverse engineering.

Concern 5: Abstraction and Traceability Difficulties

Pressman writes: *"Reverse engineering can extract design information from source code, but the abstraction level, the completeness of the documentation, the degree to which tools and human analyst work together, and the directionality of the process are highly variable... Ideally, the abstraction level should be as high as possible. That is, the reverse engineering process should be capable of deriving procedure design representations (a low level of abstraction); program and data structure information (a somewhat higher level of abstraction); data and control flow models (a relatively high level of abstraction); and entity-relationship models (a high level of abstraction)"*.[2]

Pressman goes on to say that *"the completeness of a reverse engineering process refers to the level of detail that is provided at an abstraction level. In most cases, the completeness decreases as the abstraction level increases"*.[2]

The problem is a balance between completeness and abstraction level. In DO-178B terminology, the transformation of low-level requirements to high-level requirements is difficult, and often should be considered as a two-phased approach. The first phase is to understand the software system that was built; this results in a finer granularity of high-level requirements. The second phase is to determine what the software should be doing, thus fixing the code and updating the low-level and high-level requirements and the associated traceability.

Another issue for reverse engineering in the area of requirements granularity is lack of system requirements which, itself, should add a higher-level of abstraction. These requirements establish the manner in which the system will use the application software. This can lead to a disconnect between what is expected of the software and what the software actually delivers, resulting in traceability problems between the system requirements and the software high-level requirements.

In reviewing reverse engineering projects, certification authorities frequently find the following abstraction and traceability problems:

- Airborne system requirements cannot be correlated to the reverse-engineered product's high-level software requirements.
- In some cases, high-level requirements are frequently written like low-level requirements (i.e., the abstraction level is too low); this makes testing of both high-level and low-level requirements difficult. While in some cases for software components (such as library routines), this may be justified, in many cases it is not. This generally means that true high-level requirements have not been generated, resulting in a significant gap in the levels of abstraction between the system requirements and the reverse-engineered high-level requirements, thereby jeopardizing the ability to establish traceability between the two.
- A lack of traceability from high-level requirements to low-level requirements to code, and test cases and procedures often exists, i.e., the forward traceability (and often the backwards traceability) is not established.
- Attempting to do a combined bottom-up and top-down approach rarely works, i.e., the requirements may not meet in the middle. This is often because the developers generate the high-level requirements they thought they had, whereas the low-level requirements reflect the software that they truly have.
- Developers often attempt to delay the traceability effort until the end of the project. This can result in code that does not trace to requirements or requirements that were not fully implemented.
- Developers sometimes attempt to merge requirements into a single level for very complex software. They try to omit either the high-level requirements or the low-level requirements, which makes satisfying the applicable DO-178B objectives difficult.

- Traceability is often not done carefully. Inaccurate traceability can make it difficult to determine whether code that cannot be traced to is dead, deactivated, or missing requirements.
- Establishing traceability and compliance to system-level and safety requirements is difficult in a reverse engineering effort and can often be vague or incomplete.
- Some developers may attempt to use the source code with no change. This can imply that the developers are attempting just to produce data instead of analyzing the software from a functional and safety perspective.
- Some developers use derived requirements as a placeholder for code that cannot be traced. Derived requirements should be very sparse in a reverse engineering effort, when the effort starts with the source code. When derived requirements do exist, they must be handled very carefully and addressed by the system safety experts (per DO-178B).
- When traceability is not performed thoroughly, unwanted functionality may exist in the code without the users' knowledge.

Concern 6: Certification Liaison Process Problems

Many reverse engineering efforts do not perform the certification liaison process well. The following problems often exist:

- Designees or certification authorities are not informed or involved in the planning phases of a reverse engineering process resulting in either rejection due to non-compliance of the planned process or rework for both the applicant and the certification authorities.
- Developers often hire sub-contractors or suppliers to reverse engineer a component. However, the sub-contractors do not have a communication avenue with the certification authorities or designees, nor an understanding of DO-178B and other aviation software guidance.
- Aviation companies underestimate their involvement and risk in using a re-engineered COTS product.
- Applicants and/or aviation system developers are responsible for compliance with regulations and policy and often underestimate their role in completing and demonstrating compliance with these regulations and policy.
- Use of COTS reverse-engineered products introduces a third or fourth party into projects which further complicates communication, and the certification authority typically has no relationship or authority over COTS software vendors.

Summary

Some applicants, developers, and commercial-off-the-shelf (COTS) software vendors have proposed reverse engineering as an approach for satisfying RTCA/DO-178B objectives for airborne software. If properly applied, a reverse engineering approach can allow a developer to gain the necessary assurance for airborne software, complete any missing software life cycle data, ensure that quality is built in, and provide a baseline for future projects and projects.

The certification concerns regarding use of reverse engineering can be summarized as follows:

- Reverse engineering should be used cautiously and only in well-justified cases (i.e., for a project that has been used in a number of applications and has shown itself to be of high integrity). The use of reverse engineering in new software development is strongly discouraged by the certification authorities (e.g., reverse engineering prototyped software). Additionally, a justification of cost savings without technical quality is not acceptable as a means of compliance to the regulations.
- A developer should make a case for why reverse engineering is feasible and how it will satisfy the objectives of DO-178B, other applicable regulatory guidance, and the overall safety objectives of the regulations, for the particular project. The following DO-178B objectives may be particularly difficult to satisfy and require special attention: Objectives 1 through 7 of Table A-1; Objectives 5 and 6 of Table A-3; Objectives 5, 6, and 12 of Table A-4; Objectives 4 and 5 of Table A-5; Objectives 1 and 2 of Table A-9; and Objective 2 of Table A-10.
- Reverse engineering processes should be well-defined and well-planned. The approach should be planned into life cycle processes and activities, with transition criteria, and should be documented in the plans and standards. The development team should follow these plans and standards.
- The reverse engineering project should be coordinated with the appropriate certification authorities. Since there are a number of concerns regarding reverse engineering, any projects using it should be coordinated

with the certification authorities as early in the life cycle as possible. Certification authorities may perform software reviews to ensure that the software life cycle data produced is complete and correct, and that all applicable DO-178B objectives are satisfied.
- Developers should address the concerns documented in the "Certification Concerns in Reverse Engineering" section of this paper and any other project-specific concerns.

Acknowledgment

The authors wish to thank Cheryl Dorsey (Digital Flight), Mike DeWalt (Certification Services Inc.), and Will Struck (Federal Aviation Administration) for their technical review of this paper. They also thank the international Certification Authorities Software Team (CAST) for their input on reverse engineering.

References

[1]RTCA/DO-178B, *Software Considerations in Airborne Systems and Equipment Certification*, December 1, 1992.
[2]Pressman, Roger S., *Software Engineering: A Practitioner's Approach*, McGraw Hill, 1997.

Ellis Hitt
Associate Editor

About the Authors

Leanna Rierson is the FAA's Chief Scientific and Technical Advisor for Aircraft Computer Software. She has software engineering experience at the FAA's Avionics Branch, the Wichita Aircraft Certification Office, Cessna Aircraft, and NCR. Ms. Rierson holds a bachelor's degree in electrical engineering and a master's degree in software engineering.

Barbara Lingberg is a Software Program Manager in the FAA's Aircraft Engineering Division and program sponsor for FAA's Software/Digital Systems Safety Program. Prior to work in aircraft certification, she was Software Lead for the Wide Area Augmentation System. Ms. Lingberg holds a master's degree in Software Systems Engineering and a bachelor's degree in Mathematics.