This is a draft version of:
C. Bertrand and C. Fuhrman, “Towards defining software development processes in DO-178B with OpenUP,” in *Proceedings of the Canadian Conference on Electrical and Computer Engineering*, 2008, pp. 851-854.

TOWARDS DEFINING SOFTWARE DEVELOPMENT PROCESSES IN DO-178B WITH OPENUP

Christophe Bertrand, Christopher P. Fuhrman

Department of Software and IT Engineering, ÉTS (École de technologie supérieure), Montreal, Canada
{christophe.bertrand.1|christopher.fuhrman}@etsmtl.ca

ABSTRACT

Civil avionics software must be certified according to standards mandated by governmental agencies, such as the Federal Aviations Administration in the United States. Typically the certification is done in the context of the DO-178B standard. For companies seeking a first-time certification, preparation for DO-178B can be a daunting challenge. The documentation and planning of high-integrity software is therefore a software engineering problem. As a solution, we consider an open-source derivative of the Unified Process, called OpenUP, as a base process model from which to begin. Because of their importance in the DO-178B standard, software requirement activities are the focus of our study. We show that most of DO-178B’s objectives in this dimension could be supported with activities in OpenUP.

Index Terms— Civil avionics software, software certification, software process models, DO-178B, Eclipse Process Framework, OpenUP

1. INTRODUCTION

The industry in the field of aerospace, or more precisely avionics, has been confronted with rapid technological change of real-time embedded software. To ensure a degree of rigor in the software practices and to ultimately assure the safety of airline passengers, the Federal Aviation Administration (FAA) has imposed a software certification suited to the development of critical systems. The FAA chose a standard, published by the Radio Technical Commission for Aeronautics (RTCA), responding to the needs of safety and reliability vital in this field: DO-178B Software Considerations in Airborne Systems and Equipment Certification [1]. This certification standard, which is also used in Europe, Canada and Japan, enables customers to be assured that the software aspects of a critical system are developed rigorously, following a disciplined and documented process.

Many commercial avionics companies have difficulty defining and adhering to processes to support the objectives imposed by DO-178B [2]. The difficulty lies in the level of rigor normally required to certify an avionics product but also in the flexibility of the DO-178B standard. DO-178B defines objectives that a software developer must meet to attain certification, but it remains unimposing about the specific activities that must be done to support the objectives. This flexibility is intended to support various life-cycle models that may exist for different types of avionics projects [3]. Basically, it’s up to the company to document its software processes and show that they support the objectives of DO-178B.

A company can possibly “under-specify” or “over-specify” the development activities for a certification of DO-178B. As an example of under-specified activities, the proposed activities may not be sufficiently detailed or adequate to convince the FAA that they satisfy the DO-178B objectives. At the other extreme, it is possible for a company to propose activities that sufficiently support the objectives, but which impose too strict a method of development. Over-specifying activities can be costly for a company, because it must show that it is following the proposed activities, if it is to attain a certification.

One possible solution would be to start with a recognized minimal process for developing software, which is hopefully consistent with the standards outlined in DO-178B. We have considered the OpenUP 1.0 model [4], which is an open-source version of the Unified Process based on the Eclipse Process Framework [5]. Although there are other frameworks to support software process modeling, because of the limited scope of our study, we did not consider them in this paper. The study presented here focuses on supporting with OpenUP the traceability between the high and low-level requirements, which is one of the most important parts of the DO-178B standard.

This paper is presented as follows. In section 2 and 3, we present respectively an overview of DO-178B and OpenUP. In sections 4 and 5 we discuss respectively the methodology and results of our study. In section 6, we present our conclusions.

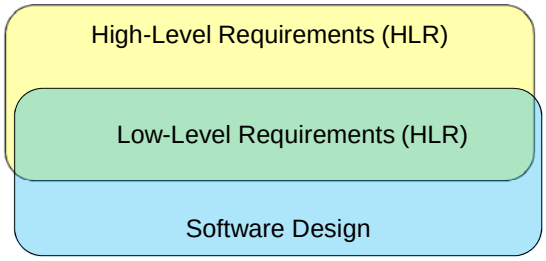

Figure 1 – HLR, LLR and Design in DO-178B [2]

2. OVERVIEW OF DO-178B

DO-178B, *Software Considerations in Airborne Systems and Equipment Certification*, is a document developed by the commercial avionics industry and RTCA, Inc. to allow government agencies and avionics software developers to obtain certified avionics software [2]. DO-178B describes objectives to reach the security conditions required by the aviation community.

In a nutshell, DO-178B describes three key processes [2]: 1) Planning, 2) Correctness and 3) Development. In the planning process, the software developer has to document the specific processes that will be followed in the correctness and development processes. Because DO-178B does not specify precise details about the latter processes, each software project involving DO-178B in the real world can potentially have different definitions of these processes. To assure quality yet remain flexible, DO-178B defines objectives for the correctness and development processes, and thus the software developers must document in the planning process how the other processes meet the objectives.

For example, the development process includes objectives for software requirements, design, coding and integration. However, if a company prefers a waterfall lifecycle to carry out the development, DO-178B will accommodate it. If another company prefers an iterative development process, DO-178B is flexible and can accommodate that, too. As long as the objectives are being met by documented processes, many possible variations of development processes are permitted.

DO-178B has flexibility according to the criticality levels of the software in the avionics system being developed. There are five criticality levels. Level A is the most critical, and applies to software that, if it were to misbehave, could result in catastrophic failure such as total loss of life. Levels B, C and D represent decreasing levels of criticality, where misbehaving software would result in some loss of life (Level B) to minor injuries (Level D). Level E is for software that, were it to misbehave, would have no effect on the aircraft's operational ability. The proportion of avionics systems/software at these levels is presented in Table 1.

2.1. DO-178B project planning

DO-178B requires five key plans: Plan for Software Aspects of Certification (PSAC), Quality Assurance Plan (QA), Configuration Management Plan (CM), Software Development Plan (SWDP), and Software Verification Plan (SWVP). The PSAC must be submitted to and approved by the government agency [2], e.g., the FAA in the US. The PSAC has links to the other plans as well as other project artifacts.

The QA plan defines how audits of the software and software processes will be carried out, and needs to be prepared by someone in the company who's independent from the development organization. The CM plan specifies how configuration control will take place, including naming conventions, revision control strategies and how problem reports are created and managed.

The SWDP contains schedules for milestones, deliverables, staffing, and addresses the software development lifecycle (e.g., iterative) and environment (e.g., software compilers, requirements management tools, etc.). The SWVP is similar to the SWDP, but for the verification process. That is, peer review process, software verification activities, types of tools for verification (e.g., formal tools).

Table 1 – Proportions of avionics in each criticality level [2]

DO-178B Criticality level	Avionics systems	Software code
Level A	20-30%	40%
Level B	20%	30%
Level C	25%	20%
Level D	20%	10%
Level E	10%	5%

2.2. DO-178B Requirements and Software Design

As part of its flexibility, DO-178B allows a company to define different methodologies for how software requirements and design are handled. To achieve this goal, it defines different levels of software requirements, namely high-level requirements (HLR) and low-level requirements (LLR). The intent is such that an overlap is between the HLR and the software design, as shown in Figure 1. In this regard, DO-178B states:

> *"Software development processes produce one or more levels of software requirements. High-level requirements are produced directly through analysis of system requirements and system architecture. Usually, these high-level requirements are further developed during the software design process, thus producing one or more successive, lower levels of requirements. However, if Source Code is generated directly from high-level requirements, then the high-level requirements are also considered low-level requirements, and the guidelines for low-level requirements also apply."*

3. OVERVIEW OF OPENUP

OpenUP is an open-source derivative of the Unified Process (UP) [6] being developed under the Eclipse Process Framework (EPF) project. The Unified Process is a popular iterative software development process for developing object-oriented systems [7]. The Rational Unified Process (RUP) [8] is also a derivative of the UP, but it is marketed as a commercial product by IBM Rational.

OpenUP, besides being open-source, is minimalist in nature. That is, it seeks to provide a process model containing the fundamental elements of software development. OpenUP is intentionally very limited so as to be a common denominator of all software projects. It does not provide specific details "on many topics that projects may deal with, such as large team sizes, compliance, contractual situations, safety or mission critical applications, technology-specific guidance, etc." [4].

Because it is based on EPF, OpenUP is designed to allow tailoring of these fundamental processes via extensions known as plug-ins. The plug-in design philosophy has been a key success factor in the Eclipse.org projects in general. The idea is that OpenUP is the common base of software processes that any project will need, and the project will customize its processes on top of OpenUP using plug-ins designed with the EPF Composer software.

OpenUP and EPF are relatively new and little academic documentation exists about them. However, they are part of an active open-source project, and usage documentation is available on the web site and news server forums. Currently, extensions to OpenUP exist to adapt it for various agile software development methodologies, e.g., XP (eXtreme Programming) [9] and SCRUM [10].

As a process model, OpenUP is organized with the following areas, many of which are identical to UP: content areas, roles, disciplines, tasks, artifacts and processes. Content areas are broken down by stakeholder, team and individual perspectives. Disciplines apply to the various disciplines in software engineering, such as Requirements, Architecture, Development, Test, Project management, etc.

The people involved in a typical software project are modeled with roles such as Stakeholder, Analyst, Architect, Developer, Tester, Project manager, etc. These roles interact with software artifacts, either to create, use or maintain them. Such artifacts include requirements and architecture documentation.

The relationship between roles and artifacts is modeled by tasks. A task is a unit of work that a role may perform. Figure 2 shows in (a) the general case of a Role, a Task and an Artifact; and in (b) a specific case of an Architect and a specific task she would perform and the artifact she would produce.

4. METHODOLOGY

To assess whether OpenUP can be successfully used in a software project involving DO-178B certification, we simulate the same approach used in certification. That is, we consider to what degree the existing documented activities in OpenUP would support the objectives of DO-178B. The methodology we use is theoretical, since a real company seeking certification would

Figure 2 – Elements of OpenUP

Table 2 – Objectives relating to traceability of HLR and LLR [1]

DO-178B Process	DO-178B Objectives
Software Requirements Process	Y HLR developed Y HLR derived defined
Software Design Process	Y Software Architecture developed Y LLR developed Y LLR derived defined
Software Verification Process	Y From SRATS : HLR developed correspond to system requirements Y LLR developed and Software Architecture correspond to HLR

have to involve a government agency. Ultimately it is that agency which must decide whether a documented process is adequate to support the objectives of DO-178B.

Because of the breadth and complexity of OpenUP and DO-178B, we have limited the scope of our study the elements involving bidirectional traceability of requirements to design. This area is of particular importance in DO-178B. Table 2 presents the DO-178B objectives that concern the bidirectional traceability between the HLR and LLR.

Our approach is then to identify which tasks of OpenUP relate to these objectives, and ideally can support them. Given the exploratory nature of our study, the authors simply discussed together the possible mappings to obtain a quick set of OpenUP tasks. We do not consider the other possible relations between OpenUP and DO-178B, such as at the level of artifacts, roles or disciplines, although these areas can be useful.

Table 3 presents a list of OpenUP activities along with their inclusive tasks that fall under the scope of the DO-178B objectives identified in Table 2.

5. RESULTS

During our analysis, we discovered that terminology is a challenge when trying to match OpenUP tasks to the objectives of DO-178B. In particular, the terms used for "architecture" are not consistent in DO-178B and OpenUP. Furthermore, DO-178B is more specific about HLR and LLR than is OpenUP, which doesn't explicitly make this distinction. Indeed, HLR and LLR are somewhat arbitrary levels of abstraction and will likely vary from project to project [2].

Although DO-178B never clearly defines architecture, one can infer that it has a simpler, perhaps lower-level meaning than in OpenUP. For example, in section 5 of DO-178B, it is stated:

"The development of a software architecture involves decisions made about the structure of the software. During the software design process, the software architecture is defined and low-level requirements are developed. Low-level requirements are software requirements from which Source Code can be directly implemented without further information."

In its clarification on LLR in DO-178B, DO-248B states:

"The aim of the architectural design is to break down the functions of the software requirements so that they can be regrouped into a logical software structure."

In contrast, the OpenUP glossary defines architecture as follows:

"[Architecture] describes the blueprint for software development, frequently represented using a number of architectural views. It also contains the rationale, assumptions, explanations and implications of the decisions that where made in forming the architecture as well as the global mapping between views."

Because of these different perspectives and levels of abstraction, it is not always obvious how tasks from OpenUP "literally" fit into DO-178B. We had to take care to consider as concretely as possible the objectives of DO-178B when considering the corresponding tasks in OpenUP.

Another noteworthy point in our results is that we made the assumption that the definition of HLR and LLR as defined by DO-178B were mapped respectively to OpenUP tasks "Detail Requirements" and "Design the Solution". Although this correspondence is logical, we point out that OpenUP has no explicit definition of high- or low-level requirements. In the

Table 3 – OpenUP activities and tasks relating to our DO-178B objectives

OpenUP Activity	OpenUP Tasks
Identify and Refine Requirements	Find and outline requirements Detail requirements
Develop the Architecture	Outline the architecture Refine the architecture
Develop Solution Increment	Design the solution

Table 4 – DO-178B process objectives supported by OpenUP tasks

DO-178B objectives		Ref. [1]	Detail Require-ments	Outline Archi-tecture	Refine Archi-tecture	Design the Solution	Create Test Cases
Forward Traceability	HLR are developed	5.1.1 a	Y				
	Derived HLR are defined	5.1.1 b	Y				
	HLR are accurate and consistent	6.3.1 b	Y				
	HLR are verifiable	6.3.1 d					Y
	Software Architecture is developed	5.2.1 a			Y	Y	
	Software architecture is consistent	6.3.3 b		Y			
	Software architecture is verifiable	6.3.3 d					
	LLR are developed	5.2.1 a				Y	
	Derived LLR are developed	5.2.1 b				Y	
	LLR are accurate and consistent	6.3.2 c				Y	
	LLR are verifiable	6.3.2 d					Y
Backward Traceability	LLR comply with HLR	6.3.2 a				Y	
	LLR conforms to standards	6.3.2 e					
	LLR are traceable to HLR	6.3.2 f					
	Software architecture is compatible with HLR	6.3.3 a					
	Software architecture conforms to standards	6.3.3 e					

interest of supporting DO-178B with OpenUP, this is not a problem. However, for a real avionics project, one must use caution about defining HLR and LLR abstraction levels shown in Figure 1. Hilderman et al [2] point out that although DO-178B is flexible about HLR and LLR definition, it's important to not define Design Descriptions as LLR, since it may require "unnecessary testing". If one were to use OpenUP, this problem would still exist.

Table 4 shows the detailed results of the mapping we did between DO-178B objectives and the OpenUP tasks, within the limited scope of our study. Most of the targeted DO-178B objectives are supported at least by one identified OpenUP task. However, five objectives, shown in boldface type in the table, were not found to be supported by OpenUP.

Of these unsupported objectives, four are relating to backwards traceability of elements in the software design. These objectives are unique to safety-critical software, so it is not surprising that OpenUP does not support them. After all, OpenUP is intended to be a lean set of process elements.

As for the remaining unsupported objective "Software architecture is verifiable", DO-178B states this objective "is to ensure that the software architecture can be verified, for example, there are no unbounded recursive algorithms." We found no related tasks in our analysis of OpenUP that could support this safety-related objective.

6. CONCLUSIONS

This paper has presented how OpenUP could possibly be used in the context of an avionics project requiring certification under DO-178B. Because of space limitations, we have presented results limited to the areas of software design including high- and low-level requirements.

Despite differences in the terminology relating to software architecture, requirements abstractions, and the fact that Open-UP doesn't have specific support for safety-critical software projects, our results show that OpenUP could still likely be useful in the context of a first-time certification project.

As a future project, we intend to investigate process customization to adapt OpenUP to support DO-178B. This work would involve developing OpenUP plug-ins, and experimenting on a real software project.

REFERENCES

[1] RTCA Inc., *RTCA/DO-178B: Software considerations in airborne systems and equipment certification*, 1992.

[2] V. Hilderman and T. Baghi, *Avionics certification: a complete guide to DO-178 (software), DO-254 (hardware)*, Avionics Communications Inc., 2007.

[3] RTCA Inc., *RTCA/DO-248B: Final report for clarification of DO-178B "Software considerations in airborne systems and equipment certification"*, 2001.

[4] R. Balduino, "Introduction to OpenUP (Open Unified Process)," 2006. [Online]. Available: www.eclipse.org/epf/general/OpenUP.pdf. [Accessed: 2 Dec. 2007].

[5] P. Kroll, “Who will benefit from the Eclipse Process Framework,” 2007. [Online]. Available: eclipse.org/proposals/beacon/Who will benefit from Eclipse Process Framework.pdf. [Accessed: 6 Dec. 2007].
[6] I. Jacobson, et al., *The unified software development process*, Addison-Wesley, 1999.
[7] C. Larman, *Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development*, Prentice Hall PTR, 2005.
[8] P. Kruchten, *The rational unified process: an introduction*, Addison-Wesley, 2004.
[9] K. Beck and C. Andres, *Extreme programming explained: embrace change*, Addison-Wesley, 2005.
[10] K. Schwaber and M. Beedle, *Agile software development with Scrum*, Prentice Hall, 2002.