Deriving DO-178C Requirements Within the Appropriate Level of Hierarchy

Jamie P. White
Department of Computer Science
University of North Dakota
Grand Forks, USA
Jamie.white@gmail.com

Hassan Reza
Department of Computer Science
University of North Dakota
Grand Forks, USA
reza@aero.und.edu

***Abstract*— In this paper, a set of criterion is proposed that assists an engineer in placing a derived requirement as defined by DO-178C in the proper document within the requirement document hierarchy. The proper documentation of derived requirements has historically posed some issues when it comes to requirements-based testing. For one thing, if the derived requirements are inappropriately documented, then it will be very difficult to establish traceability between individual requirements to the elements of design, implementation, and verification. Consequently, the lack of correlation between elements of requirements, design, code, and verification can jeopardize the safety of systems because it will be impossible to establish forward and backward traceability. To this end, the proposed criteria discussed in this work attempts to improve the visibility of derived requirements to prevent the unwanted consequences of masking required information from developers.**

Keywords-requirement traceability; requirements analysis; safety critical systems; RTCA/DO-178C; software testing; software design

I. INTRODUCTION

An aircraft system is a complex system that is composed of a hierarchy of subsystems, which are further decomposed into software and hardware elements. A set of requirement documents describe each level of this requirement hierarchy. The highest layer, the system requirements, specifies the observable requirements at the system level (e.g., The System shall display total fuel quantity on the FMS Departure page). These very high level requirements are allocated to specific subsystems such as a Flight Management System (FMS) (e.g., FMS shall display total fuel quantity in either pounds or kilograms). Subsystem requirements are then further allocated to software and hardware high-level requirements (HLRs) (e.g., FMS shall display total fuel quantity in kilograms when metric units are selected). These HLRs can then be decomposed into low-level requirements (LLRs) at which point they should be specific enough to implement in hardware or software. Figure 1 illustrates a hierarchy of requirements starting at the system level that is decomposed into subsystem requirements and then further decomposed into hardware and software requirements.

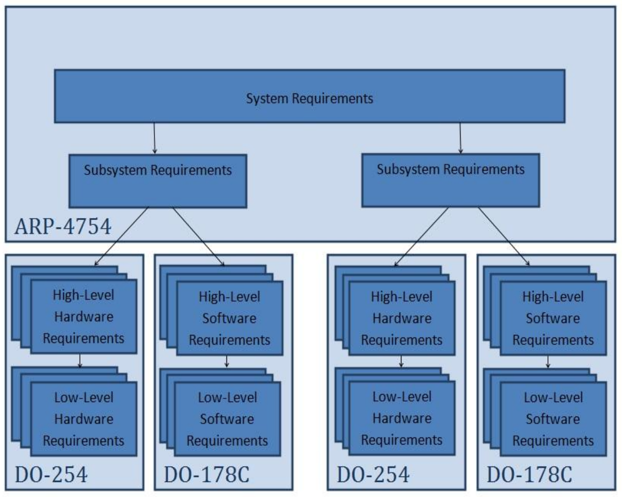

System Requirements
Subsystem Requirements
Subsystem Requirements
ARP-4754
High-Level Hardware Requirements
Low-Level Hardware Requirements
DO-254
High-Level Software Requirements
Low-Level Software Requirements
DO-178C
High-Level Hardware Requirements
Low-Level Hardware Requirements
DO-254
High-Level Software Requirements
Low-Level Software Requirements
DO-178C

Figure 1. Example of a system requirement document hierarchy

High-level requirements do not always contain sufficient details to describe the underlying requirement documents. As such it is necessary to create derived requirements (DR). A DR as defined by DO-178C [10] is a requirement that is not directly traceable to a higher-level source; it is inferred or deduced from a specific source/user. As an example of a derived requirement for ABC system can be read as “The ABC DataFusion subsystem shall write position, velocity, and maneuverability data received from Radar data signal processing site 1 to external storage”. Such low-level details may be outside the scope of the SW-HLR document.

Low-level requirements are implementation of high-level requirements; they can be generated differently by different engineers but having the same functionalities. Low level requirements may then be implemented by different programmers in totally different ways, but yet representing the same functionalities [18].

An example of corresponding low-level requirements for ABC system can be read “The ABC DataFusion subsystem shall read the position, velocity, and maneuverability data received from the Radar Subsystem every 60 seconds”.

As DO-178C requires the existence of source code is directly traceable to a requirement, it will then become necessary to derive such requirements in a low-level software requirements (SW-LLR) document. Figure 2 shows an example of a software derived requirement (SW-DR) that is derived within a SW-LLR document.

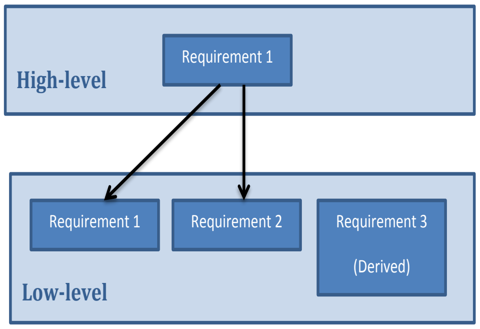


Figure 2. Example of a derived requirement

Deciding at what level a requirement should be derived at is generally done on a case by case basis (e.g., is it more appropriate to derive the requirement at the SW-HLR requirement document versus the SW-LLR document?). Such a decision could have unintended consequences and cost by hiding information applicable to other stakeholders.

As an example, let us assume that a SW-HLR document exists for a multi-threaded FMS application. From the SW_HLR document, multiple SW_LLR documents are created and categorized by independent functions worked on by multiple software engineering teams. In one of the SW_LLR documents a DR is intoduced that impacts a shared resource among the other software functions. Once implemented the other software functions could exhibit unpredicatable behaviors or defects, which may lead to additional time spent debugging the issue. If the defect is identified late in the software development process, then the cost for fixing the defect will become increasing expensive [13].

The paper is organized as follows. Section 2 provides an overview of the certification agencies and DO-178C. Section 3 contains related work. Section 4 defines criteria to serve as a guideline for the appropriate placement of a derived requirement regardless if it impacts safety aspect of a system. Such criteria are based on nonfunctional requirements such as operability, safety, reliability, observability, etc. Criteria for functional requirements are also defined, which include interfaces and configurationable elements. Section 5 shows a simple example of requiremnents for the display of fuel quantity in an aircaft cockpit. Finally Section 6 discusses a conclusion and future work.

II. BACKGROUND

Certification agencies such as the Federal Aviation Administration (FAA), Transport Canada, and the European Aviation Safety Agency (EASA) rely on industry standards to serve as guidelines on how to create aircraft systems that are certifiable, or trusted for use in airborne applications. ARP-4754 [7] serves as the guideline for system and subsystem processes, DO-254 [8] for hardware processes, and DO-178C [10] for software processes.

Avionics systems have contained software since the 1970s. As the certification of avionic systems increased in complexity, additional methods were necessary to achieve the same level of assurance as hardware based systems [11].

Radio Technical Commission for Aeronautics (RTCA) and European Organization for Civil Aviation Equipment (EUROCAE) formed committees to create common certification criteria for software development [11]. The works from these committees were merged, which led to RTCA publishing DO-178 [17] and EUROCAE publishing ED-12 with both documents containing identical content [11]. DO-178 categorized systems as critical, essential and non-essential and defined the rigor needed to develop software to each level [16].

DO-178C, published in December 2011, is the recent standard, which describes the processes in the creation of flight critical software. These processes outline the stages which include the creation of multiple levels of requirements, design, implementation and verification. From the previous version of DO-178B, published in 1992, little has changed from this core document. The changes mostly consist of fixing errors and inconsistencies, word improvements, and adding several clarifications and objectives [11]. The bulk of the work was the creation of supplement documents, referred to by DO-178C, that provide guidance on model-based development, tool qualification, object-oriented technology, and formal methods.

DO-178C describes that system level requirements are decomposed into SW-HLRs. SW-HLRs are defined by DO-178C as being developed from the analysis of system requirements, safety-related requirements and system architecture [10]. SW-LLRs are then created, which further decompose the SW-HLRs. SW-LLRs are software requirements that were developed from SW-HLRs or are derived, which describe in sufficient detail to allow source code to be implemented without additional information [10]. The role of SW-HLRs is to describe the ‘what’ and for the SW-LLRs to describe the ‘how’ [5][6]. SW-DRs can be SW-HLRs or SW-LLRs and are not directly traceable to higher-level requirements. SW-DRs are used to specify additional behavior beyond what is defined in the higher level requirements [10].

Software is implemented from the SW-LLRs and is verified from requirements based testing that verify the correctness of SW-HLRs and SW-LLRs. Finally, traceability is required to be maintained at each stage (i.e., SW-LLRs are traceable to SW-HLRs, code is traceable to SW-LLRs and verification is traceable to both SW-LLRs and SW-HLRs).

The certification standards specify that traceability, both forward and backward, is needed for the allocated requirements at each requirement document level. Requirements traceability describes the ability to describe and follow a requirement in both a forward and backward direction [1]. A requirement management tool such as IBM Rational DOORs supports linking between levels of requirements through the use of requirement attributes [2].

In DOORs, a requirement document is called a module. One way to organize a project would be to create a separate DOORs module for each level of the requirement hierarchy (e.g., system requirements, subsystem requirements, SW-HLRs, SW-LLRs). A DOORs link can then be used to connect multiple requirements. A requirement can be linked with another requirement that exists in the same module or a different module. DOORs links are directional and are categorized as ‘in-links’ or ‘out-links’. In the context of DO-178C, the practice of linking is done to show decomposition of requirements. As an example, a SW-HLR could contain in-links from one or more SW-LLRs and an out-link to a subsystem requirement. Using links in DOORs, forward and backward traceability is maintained (e.g., for a specific SW-HLR, traceability information exists for the source of the SW-HLR as well as the SW-LLRs that decompose it).

Finally, DO-178C requires additional processes for SW-DRs. Rationale must be documented to support the existance of a SW_DR When using DOORs, this documentation can be captured as an attribute attached to the SW-DR. Additional processes must be created to provide DRs to the system process. Typically, this is done by having a safety engineer review all SW-DRs and its rationale to determine if there is an impact to safety (i.e., could the SW-DR cause loss of function resulting in additional pilot workload or provide misleading information to the pilots). SW-DRs that are determined to impact safety are captured by the system safety assessment process. Most DRs typically do not impact safety so these requirements are larged ignored at the system level. Even if a SW-DR requirement that impacts safety is tracked at the system level, these requirements are not easily traced to other functional requirements of the system. Hence, it is still important that SW-DRs that impact safety are derived at the appropiate level.

III. RELATED WORK

Since the 1970s, product organizations have used requirements traceability in order to have complete and consistent information about the product being built [2]. Since this time, much work has been done studying requirements traceability and traceability tool support [3]. Others have proposed frameworks for the organization of traceability information [4]. Studies have been conducted to understand the benefits and costs of traceability [1][2].

In 2011, RTCA (Radio Technical Commission for Aeronautics), Inc. published DO-178C “Software Considerations in Airborne Systems and Equipment Certification”, which serves as a guide for the creation of airborne software using traditional methods [5], and which typically utilize the C and ADA programming languages. DO-331[ref] was also published in 2012 which describes how to implement software using model-based development (MBD) [6]. Many companies selling aviation products follow DO-178C or the previous release of DO-178B to prove airworthiness of their software elements. Regulatory agencies such as the Federal Aviation Administration (FAA), Transport Canada, and the European Aviation Safety Agency (EASA) audit these software elements seeking compliance to DO-178C.

Nonfunctional requirements (NFRs) play an important role in the creation of software architecture and are blamed for system re-engineering when not considered when designing the architecture [14]. In an avionics environment, certain NFRs play important role, which include security, maintainability, safety, availability, integrity, and schedulability [15]. These NFRs should be considered when creating the software architecture.

IV. METHOD

In this section, 6 points of criterion have been suggested for the proper placement of derived requirements within a requirement document hierarchy. The criteria are intended to cover certain special cases such as requirements that specify external interfaces, externally observable behavior, configurable elements, etc. Engineers should consider each criterion to determine the most appropriate placement for a derived requirement to ensure information is not hidden from the stakeholders.

As an example, system or subsystem engineers will have little visibility of requirements defined in a SW-HLR or SW-LLR document. If a requirement is derived in a SW-HLR or SW-LLR document, there would be no link for the system or subsystem engineer to follow from their requirements documents. This could result in important information being hidden. On the other hand, putting large amounts of irrelevant details in a high level requirements document could result in the document becoming unmanageable.

The end goal is to place requirements in the requirements document that provides the most visibility to the stakeholders while preserving the scope of the document. Hence, there is a fine line between putting too much information in a high-level requirements document and providing an appropriate amount of visibility to stakeholders. In addition, the CAST-15 position paper provides guidance that for software requirements a high-level requirements document should describe the “what” and a low-level requirements document should describe the “how” [5]. Placing the derived-requirements in the correct requirements document will improve traceability by making the requirements more visible to the stakeholders. Some of the consequences of poor traceability include lower changeability and higher maintenance costs [3].

A. *Requirements that specify an external interface*

Software and hardware elements contain external interfaces. Software elements provide *APIs* allowing communication with other software elements. Hardware can contain external interfaces for data buses (e.g., ARINC 429 connectors) power adapters or other form factors.

The introduction of the derived requirement for such features would depend on the scope or visibility of these interfaces (i.e., the software functions or software applications that have access to these interfaces). For instance, if a software element provides a service to software elements in other subsystems, it would be appropriate to create a high-level parent requirement in the systems requirement document (e.g., “The system shall provide an interface to collect fault information”) Such a requirement would be further decomposed in the subsystem and software requirements document until it is specific enough to be implemented. If the software element provides an API that is only visible to software elements in the same subsystem, the parent requirement should be derived in the subsystem requirement document (e.g., FMS shall provide an interface to store FMS faults to the FMS maintenance log).

B. Requirements that describe externally observable behavior

Externally observable behavior includes any set of inputs that yields an output that can be noticeable to the user. The set of requirements in the systems or subsystems documents should provide high-level detail for the capabilities and functional requirements that are observable to the user. The user (i.e., pilot) must have a complete understanding for the controls of the aircraft and in turn the aircraft must respond in a deterministic fashion. Requirements must exist to capture all behavior. Requirements should also be added to cover any corner or fault conditions. For instance, if a FMS software element is placed in a fault state, which produces an error message visible to the user, there should be a basis for that requirement at the system level (e.g., FMS shall display "FMS Unavailable" within the target window when unavailable). Such a requirement at the system level would be more appropriate than at the subsystem level since it is observable to the user. Information should not be 'hidden' from the user. In addition, test procedures should be written based on system or subsystem requirements for anything observable at the system level.

C. Requirements that describe configurable elements

Many components of an aircraft are configurable to support reuse on different aircraft types or for selectable options for a specific aircraft type.

This criterion depends on which level the element needs to be configurable. For example, an aircraft manufacture may sell a common avionics package to its customers, which are allowed to purchase additional features. Features may be enabled through the use of licenses. The basis of license management should be defined at the system level with requirements on how to configure the system. Alternatively, software elements may be reused on many aircraft systems, which contain a single configuration per aircraft type. Describing the configurable elements at a higher level would add no value. As an example, the owner of an aircraft may subscribe to services such as Graphical Weather Radar that would require a key to enable. Maintenance personnel could enable the feature by entering the license key in one of the avionics application maintenance pages. System requirements should capture this capability. Another example is a Radio Interface Unit (RIU), which may be configurable to support multiple aircraft types to support reuse. At the system level, there would be no need to capture such requirements since it was a design decision to make the RIU configurable to support reuse. The implementation details should in turn be hidden from the user.

D. Requirements that describe performance, schedulability, and design margin

For software, especially within an Integrated Modular Avionics (IMA) environment, system requirements describing performance, schedulability, and design margin (e.g., maximum allowed latency, CPU utilization, memory usage, etc) should be defined. This is required so that when the aircraft is integrated each application has sufficient resources. Additional capacity should be left for future growth. In addition, subsystems requirements, SW-HLRs, and SW-LLRs could have derived requirements to account for future growth, reuse, task scheduling, etc. For example, derived SW_HLRs describing how often specific threads should run could be defined. SW_LLRs could contain derived requirements specifying size of buffers.

E. Requirements that describe security features

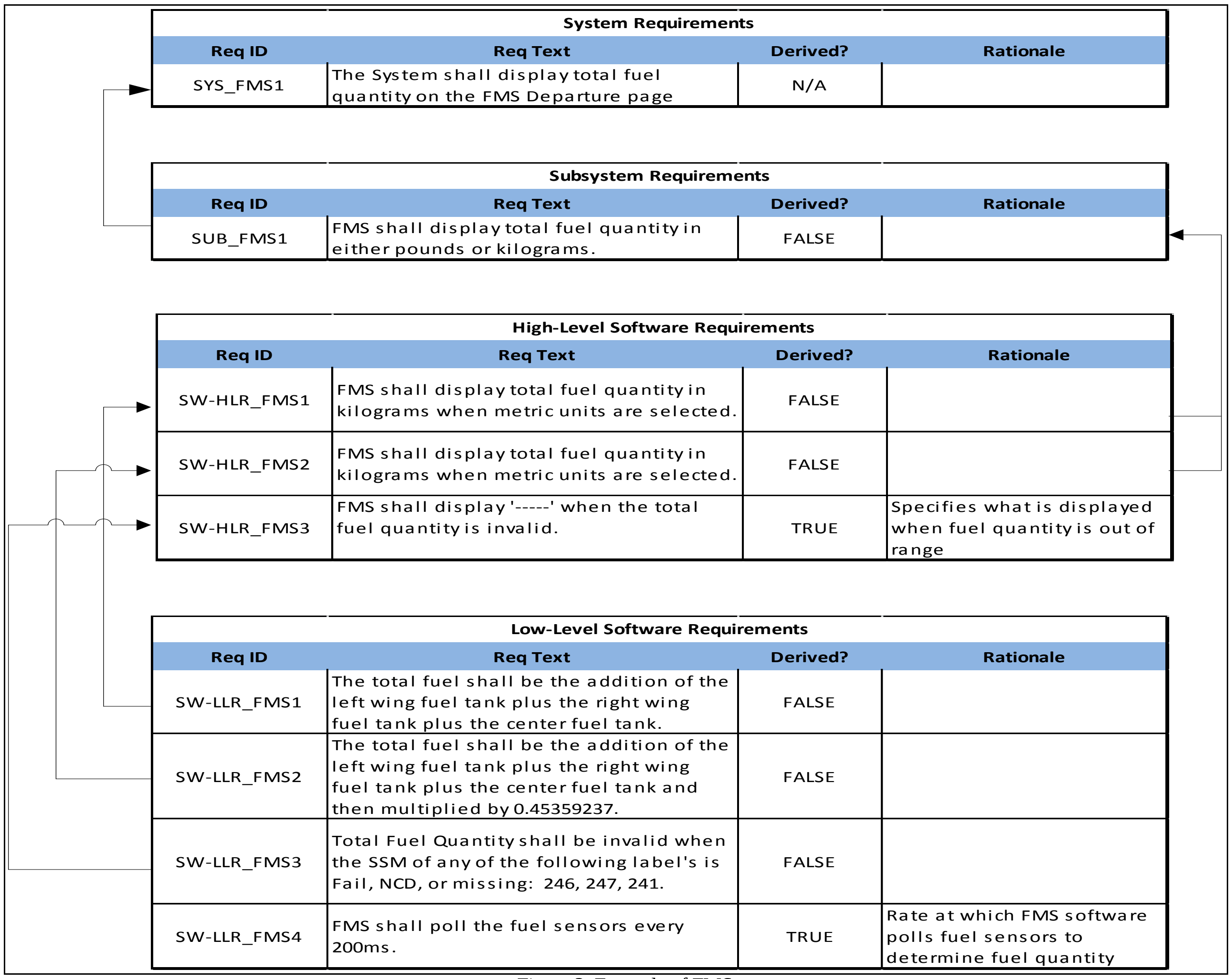
Like external interfaces, requirements describing security features depend on scope. For example, many aircrafts now provide support for ETHERNET for its passengers [15]. The mechanism that isolates ETHERNET traffic from other aircraft communications should be done at the system level or subsystem level. An example of a derived SW_HLR, databases used by an FMS could be encrypted to preserve propriety information. As it is not applicable to the system that such security features are implemented, it would not be appropriate to define such security features in a higher-level document.

F. Requirements that describe system safety availability constraints

Safety requirements concerning the development process include efforts to ensure correctness of the design and correctness of requirements in terms of safety where availability describes the continuity of a function [15]. System safety/availability constraints are expressed in DO-178C through a Design Assurance Level (DAL). DAL A is the most stringent and is designated to functions that could contribute to a catastrophic failure condition (e.g., failure could cause a crash). DAL E is at the other end of the spectrum in which a failure has no impact on the safety of the aircraft. Safety assessments for the aircraft should be conducted and defined at the system level as specified by ARP4754 [7]. Hence, requirements expressing DAL levels should be contained in system documents and not derived in software documents. Derived requirements at any level need to be reviewed by a safety engineer to ensure there is no negative effect on safety or availability of the aircraft (e.g., a failure could cause a loss of the primary flight display).

V. EXAMPLE

Figure 3 illustrates an example of FMS requirements responsible for displaying total fuel quantity beginning at the system level. This example demonstrates some of the difficulty in determining the correct place to capture DRs. SYS_FMS1, a system requirement, is decomposed into one subsystem requirement, SUB_FMS1. This decomposition is represented by an ‘in-link’ into SYS_FMS1. The ‘in-link’ is represtented by a directional arrow from SUB_FMS1 to SYS_FMS1.



System Requirements

Req ID	Req Text	Derived?	Rationale
SYS_FMS1	The System shall display total fuel quantity on the FMS Departure page	N/A	

Subsystem Requirements

Req ID	Req Text	Derived?	Rationale
SUB_FMS1	FMS shall display total fuel quantity in either pounds or kilograms.	FALSE	

High-Level Software Requirements

Req ID	Req Text	Derived?	Rationale
SW-HLR_FMS1	FMS shall display total fuel quantity in kilograms when metric units are selected.	FALSE	
SW-HLR_FMS2	FMS shall display total fuel quantity in kilograms when metric units are selected.	FALSE	
SW-HLR_FMS3	FMS shall display '-----' when the total fuel quantity is invalid.	TRUE	Specifies what is displayed when fuel quantity is out of range

Low-Level Software Requirements

Req ID	Req Text	Derived?	Rationale
SW-LLR_FMS1	The total fuel shall be the addition of the left wing fuel tank plus the right wing fuel tank plus the center fuel tank.	FALSE	
SW-LLR_FMS2	The total fuel shall be the addition of the left wing fuel tank plus the right wing fuel tank plus the center fuel tank and then multiplied by 0.45359237.	FALSE	
SW-LLR_FMS3	Total Fuel Quantity shall be invalid when the SSM of any of the following label's is Fail, NCD, or missing: 246, 247, 241.	FALSE	
SW-LLR_FMS4	FMS shall poll the fuel sensors every 200ms.	TRUE	Rate at which FMS software polls fuel sensors to determine fuel quantity

Figure 3. Example of FMS

SUB-FMS1 is decomposed into two SW-HLRs, SW-HLR-FMS1 and SW-HLR_FMS2. SW-HLR_FMS3 is a SW-HLR DR since it is not traceable to SYS-FMS1. Philisophically, SW-HLR_FMS3 is a DR since it does not decompose SUB-FMS1 but instead describes the behaviour when fuel quantity cannot be displayed. In Figure 3, it is apparent that SW-HLR_FMS3 is a DR since it contains no ‘out-link’ to a higher-level source. In addition, it is a common practice to create a requirement attribute specifying if the requirement is derived or not. This derived attribute is shown in the “Derived?” column for each requirement. Since HLR-FMS3 is a DR, it is also required by DO-178C to include rationale, which is included in the “Rationale” column.

SW-HLR_FMS1, SW-HLR_FMS2, and SW-HLR_FMS3 are further decomposed into SW-LLR_FMS1, SW-LLR_FMS2, and SW-LR_FMS3. Finally SW-LLR_FMS4 and SW-LLR_FMS5 are derived requirements since they are not traceabile to a higher source.

Let us now review each of the derived requirements. SW-HLR_FMS3 is an interesting example of a requirement that contains externally observable behaviour’ as described in Section 4. The caveat is, this externally observable behaviour is not based on an explicit input by the user, but instead a minor fault condition.

Many would argue that it would not be appropriate to describe a requirement such as SW-HLR_FMS3 in the systems requirement document, as it would be too detailed. SW-HLR_FMS3 merely captures the behaviour for a corner fault conditon that should not occur in normal operation.

Using the same argument, SW-HLR_FMS3 may also not be appropriated in the subsystem requirement document. There would be value in including a requirement such as SW-HLR_FMS3 in the subsystem requirements document for other reasons. Through subsystem testing, it is quite common to introduce faults into the system that would reach such fault conditions. By not having the information from SW-HLR_FMS3 stored or otherwise traced to the subsystem document, a subsystem engineer may have diffiuclty to understand why the fault occurred. The engineer may need to consult the domain experts or search for this information in software requirement documents.

SW-LLR_FMS4 is another interesting example of a DR, which relates to a requirement that describes performance, schedulability, and design margin (see Section 4). SW-LLR_FMS4 describes how often the FMS task should read fuel quantity information from the fuel sensors. As long as the value chosen does not impact system performance (e.g., sending data requests every millisecond which could overload a data bus), a detail such as this will be insignificant at the system or subsystem level. Since SW-LLR_FMS4 is a DR, a safety engineer is required to review this requirement to determine if it can impact the safety of the aircraft.

For this DR, it is assumed that the fuel sensor information is published throughout the system at some constant rate. Therefore, there should be no real stakeholders of this information. Hence, this requirement could be considered an implementation detail and placed in a SW-LLR document.

On the other hand, if the fuel sensors were a shared resource among multiple FMS threads, there could be contention. In this case, it will be appropriate to put LLR_FMS4 in the SW-HLR document. For a more complicated example, consider if there are primary and secondary fuel sensors. Would it be considered an implementation detail to fail-over to the secondary fuel sensor data when the data from the primary fuel sensor is invalid or missing?

In summary, this work shows that it is not always a trivial problem to determine the proper placement for a DR. In terms of DO-178C, there is no explicit answer beyond putting the “what” in the SW-HLR and the “how” in the SW-LLR. Therefore, other aspects must also be considered such as ensureing traceability and visibility of information.

VI. CONCLUSION AND FUTURE WORK

This study proposes a set of guidance to for the optimal placement of derived requirements as defined by DO-178C. By deriving a requirement in a document that is too low-level may have the unwanted consequences of hiding information from stakeholders (e.g., engineers). This, in turn, may result in forward and backward traceability problem, which can compromise dependability of safety critical systems.

In future work, apart from creating additional detail in the criteria contained in this paper, validation of these criteria will be carried on. Another related topic for future work would be to create criteria to determine if a LLR correctly decomposes a HHR or if it should be considered a DR. There is more effort involved with the creation of DRs (e.g., creation of rationale, review with safety engineer, additional scrutiny, etc.). Because of this, engineers may be more prone to create a trace to a HLR (indicating decomposition) versus specifying as a DR when there is a gray area. Of course, this is especially dangerous (left uncorrected) as this would bypass a review by a safety engineer.

REFERENCES

[1] Gotel, O. C .Z. and Finkelstein, C. W., "An analysis of the requirements traceability problem," Requirements Engineering, 1994., Proceedings of the First International Conference on Software Engineering , pp. 94-101, April 1994.

[2] Kirova, V., Kirby, N., Kothari, D., and Childress, G., ”Effective requirements traceability: Models, tools, and practices,” Bell Labs Technical Journal, vol. 12, no. 4, pp. 143-157, 2008.

[3] Winkler, S. and Pilgrim, J., “‘A survey of traceability in requirements engineering and model-driven development,” Software & Systems Modeling, vol. 9, no. 4, pp. 529-565, 2010

[4] Ramesh, B. and Jarke, M., “Toward Reference Models for Requirements Traceability,” IEEE Transactions on Software Engineering, vol. 27, no.1, pp. 58-93, 2001.

[5] RTCA. DO-178B/ED-12B. Software Considerations in Airborne Systems and Equipment Certification. RTCA, 1992.

[6] Certification Authorities Software Team (CAST) Position Paper.; “CAST-15: Merging High-Level and Low-Level Requirements”, February 2003.

[7] Marques, J. C., Yelisetty, S. M. H., Dias, L. A. V., and da Cunha, A. M., "Using Model-Based Development as Software Low-Level Requirements to Achieve Airborne Software Certification," Information Technology: New Generations (ITNG), pp. 431-436, April 2012.

[8] “Guidelines for Development of Civil Aircraft and Systems”, EUROCAE ED-79A and SAE Aerospace Recommended Practice ARP 4754A, 2010.

[9] RTCA, 2000, DO-254: Design Assurance Guidance for Airborne Electronic Hardware, RTCA, Inc., Washington, DC.

[10] RTCA DO-178C—Software Considerations in Airborne Systems and Equipment Certification, December 2011.

[11] Qualtech Consulting Inc, “Summary of Difference Between DO-178B and DO-178C", http://www.faaconsultants.com/html/do-178c.html.

[12] Taylor, C., Alves-Foss J., and Rinker, B., “Merging Safety and Assurance: The Process of Dual Certification for Software.” Proceeding of Software Technolgy Conference (STC), Salt Lake City, UT, 2002.

[13] Chaar, J. K., Halliday, M. J., Bhandari, I. S., and Chillarege, R., “In-Process Evaluation for Software Inspection and Test,” IEEE Trans. Software Eng., vol. 19, no. 11, pp. 1055-1070, November 1993.

[14] Reza, H., Jurgens, D., White, J., Anderson, J., and Peterson, J., "An architectural design selection tool based on design tactics, scenarios and nonfunctional requirements," Electro Information Technology, 2005.

[15] Paulitsch, M., Ruess, H., and Sorea, M., “Non-functional Avionics Requirements,” Communications in Computer and Information Science, vol. 17, pp. 369–384, 2009.

[16] Johnson, L. A., "DO-178B, Software considerations in airborne systems and equipment certification", http://www.dcs.gla.ac.uk/~johnson/teaching/safety/reports/schad.html.

[17] Radio Technical Commission for Aeronautics, “DO-178 - Software Considerations in Airborne Systems and Equipment Certification”, Washington, United States, 1982.

[18] Hayhurst, K.,Veerhusen, D., Chilenski, J., and Rierson, L. A. Practical Tutorial on Modified Condition/Decsion Coverage. NASA/TM-2001-210876, 2001.