

Considerations in the Preference for and Application of RTCA/DO-178B in the Australian Military Avionics Context

Squadron Leader Derek W. Reinhardt

Systems Certification and Integrity (SCI)
Directorate General Technical Airworthiness (DGTA)
Bldg L474 (B-2-STH), RAAF Williams, Laverton 3027, Victoria, Australia

derek.reinhardt@defence.gov.au

Abstract

RTCA/DO-178B is the Australian Defence Force's (ADF's) preferred software assurance standard for safety critical and safety related airborne software development. However, RTCA/DO-178B is often the centre of much debate or criticism for several reasons. The absence of mandatory formal methods and static code analysis, the absence of objectives relating to software safety analysis and software safety requirements, and the ineffectiveness of testing regimes are the key focuses of criticism. The assumptions underlying the integrity level definition may also be questionable. Alternatively, there are others that believe that the verification objectives of RTCA/DO-178B are too onerous, and that the fidelity requirements regarding specification and traceability of software requirements are conflicting with common software development practices. This paper examines these criticisms and discusses how they influence the ADF's preference for and application of RTCA/DO-178B. Specific factors such as how RTCA/DO-178B is applied in conjunction with other standards in the ADF framework, the test coverage objectives, use of RTCA/DO-178B as a software assurance benchmark, the use of COTS software and migration issues are also considered.

Keywords: Certification, Software Assurance, Software Safety, Safety Critical, RTCA/DO-178B.

1 Introduction

RTCA/DO-178B (Software Considerations In Airborne Systems and Equipment Certification) is the Australian Defence Force's (ADF's) preferred software assurance standard for safety critical and safety related airborne software development [AAP7001.054]. The Directorate General Technical Airworthiness (DGTA) is the ADF's Technical Airworthiness Regulator/Authority, and is responsible for determining airworthiness design standards for ADF aircraft. Guidance on relevant design

standards, including RTCA/DO-178B, is published within the Airworthiness Design Requirements Manual [AAP7001.054].

The ADF recognises that the Federal Aviation Administration (FAA) processes and standards for software in airborne systems are widely used in the aerospace industry, and accepted by many international airworthiness authorities. In addition many military aviation systems have a civil heritage with software developed using the civil aviation guidelines. Furthermore, major suppliers developing systems for both civil and military aircraft are basing their software development processes on the RTCA/DO-178B guidelines.

However, RTCA/DO-178B is often the centre of much debate or criticism for several reasons. The absence of mandatory formal methods and static code analysis at higher assurance levels is one such example. The apparent absence of explicit requirements or objectives relating to software safety analysis and software safety requirements is another source of debate. Some studies (e.g. [McD01]) have also criticised that the assumptions underlying the integrity level definition are questionable, in that the higher integrity levels may not, of themselves, produce software with lower hazardous failure rates. There are others that believe that the verification requirements of RTCA/DO-178B are too onerous, and that the fidelity requirements regarding specification and traceability of software requirements conflict with common software development practices.

Avoidable software failures, using the approaches prescribed by standards such as RTCA/DO-178B, have already been responsible for loss of life and for major economic losses according to [JTM07]. Cockpit displays plunged into darkness, engines that throttle back during take-off, and contradictory airspeed readings are just some of the problems caused in recent years by inexplicable failures in the software that controls aircraft [Mar08]. For example software has been attributed in the following aircraft related accidents and incidents:

- Software was implicated in the Korean Air jumbo jet crash in August 1997 on Guam, which killed 228 people [Mar 08].

Copyright © 2008, Australian Computer Society, Inc. This paper appeared at the 13th Australian Workshop on Safety Related Programmable Systems (SCS'08), Canberra. Conferences in Research and Practice in Information Technology, Vol XXXII. Tony Cant, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

- In August 2005, a computer in a Boeing 777 presented the pilot with contradictory reports of airspeed [Mar08].
- Faulty logic in the software meant that when computer controlling fuel flow failed in an Airbus A340, the back up systems were not turned on [Mar08].

This paper examines these criticisms and discusses how they influence the ADF's preference for and application of RTCA/DO-178B. In doing so, this paper provides insight into the fundamental principles underlying the RTCA/DO-178B objectives and the common suites of evidence used to demonstrate compliance. Broadly accepted software safety criteria such as requirements validity, requirements satisfaction and requirements traceability are also considered. Where gaps are identified in the coverage of these criteria, then additional discussion is provided on how they are addressed by the broader ADF framework.

2 Background of RTCA/DO-178B

Before examining the criticisms of RTCA/DO-178B, it is firstly necessary to establish some background to RTCA/DO-178B, and the context within which it is applied.

2.1 Relationship to FAA Certification Process and System Safety Assessment

Society of Automotive Engineering (SAE) Aerospace Recommended Practice (ARP) 4754 and SAE ARP4761 are the preferred means of demonstrating compliance to Title 14 within the Code of Federal Regulation (CFR) 25.1309 (colloquially referred to FAR 25.1309), the regulations encompassing system safety for civil aviation certification of transport category aircraft.

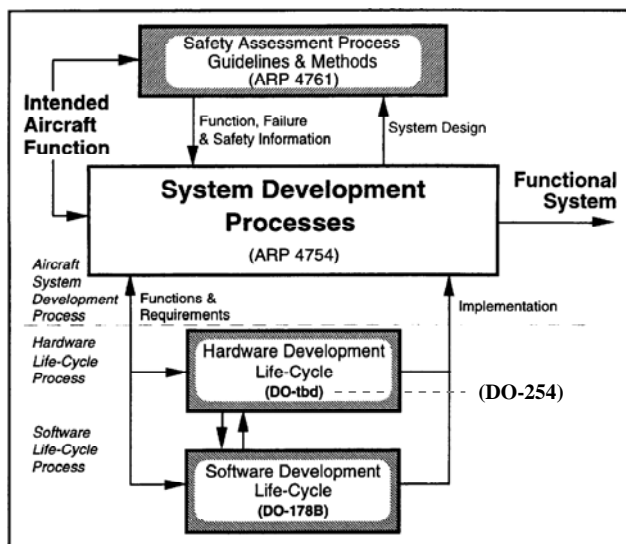


Figure 1: Relationship between ARP4754, ARP4761, RTCA/DO-178B and RTCA/DO-254 [ARP4754]

Similar regulations exist for sports and commuter category aircraft, as well as small and transport category rotorcraft.

ARP4754 and ARP4761 are not intended to be used in isolation, and are typically employed in concert with RTCA/DO-178B for software design assurance and RTCA/DO-254 for hardware design assurance, as presented in Figure 1.

The safety analysis prescribed in ARP4754 (Figure 2), and ARP4761 is centred around the identification, and assessment of functional failure conditions through aircraft and system level Functional Hazard Assessments (FHA) and Fault Trees Analysis (FTA), and system failure conditions through system and sub-system Failure Modes and Effects Analysis (FMEA), FTAs, and other forms of analysis (e.g. Markov analysis, Functional Failure Analysis). In addition, Common Cause Analysis (CCA) is conducted consisting of particular risk analysis, zonal safety analysis and common mode analysis. It is the results of these analyses that levy item and safety requirements onto elements of the system, including software.

To some extent the validity of software safety requirements will be assured through this process, however ARP4754 and ARP4761 have very limited guidance on analysis of software behaviour, and thus it is not possible to rely solely on the functionally oriented software functional failure analysis that is recommended.

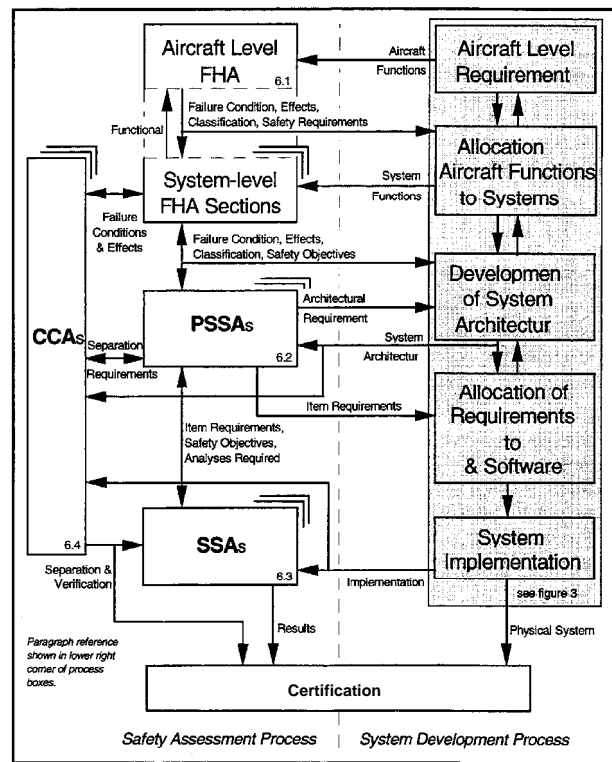


Figure 2: Safety Assessment Process Model [ARP4754]

ARP4754 defines five failure condition severities, and assigns them Design Assurance Levels (DALs), which align to the Software Levels defined in RTCA/DO-178B (Table 1).

| Failure Condition Severity | Definition | | Software Level [DO178B] |
|----------------------------|--|--|-------------------------|
| | | | |
| Catastrophic | Failure conditions which would prevent continued safe flight and landing. | | A |
| Hazardous / Severe Major | Failure conditions which would reduce the capability of the aircraft or the ability of the crew to cope with adverse operating conditions to the extent that there would be: (1) a large reduction in safety margins or functional capabilities, (2) physical distress or higher workload such that the flight crew could not be relied on to perform their tasks accurately or completely, or (3) adverse effects on occupants including serious or potentially fatal injuries to a small number of those occupants. | | B |
| Major | Failure conditions which would reduce the capability of the aircraft or the ability of the crew to cope with adverse operating conditions to the extent that there would be, for example, a significant reduction in safety margins or functional capabilities, a significant increase in crew workload or in conditions impairing crew efficiency, or discomfort to occupants, possibly including injuries. | | C |
| Minor | Failure conditions which would not significantly reduce aircraft safety, and which would involve crew actions that are well within their capabilities. Minor failure conditions may include, for example, a slight reduction in safety margins or functional capabilities, a slight increase in crew workload, such as, routine flight plan changes, or some inconvenience to occupants. | | D |
| No Effect | Failure conditions which do not affect the operational capability of the aircraft or increase crew workload. | | E |

Table 1: Development Assurance Levels [ARP4754] and DO-178 Software Levels [DO178B]

2.2 Structure of RTCA/DO-178B

Annex A to RTCA/DO-178B presents a total of 66 objectives in 10 tables relating to different software lifecycle phases (planning, development, requirements, design, coding and integration, verification), integral processes (configuration management and quality assurance) and liaison with the certification authority (i.e. normally the FAA, or the ADF (DGTA) in the case of

this paper). Table 2 and Table 3 present extracts from these RTCA/DO-178B tables. Note that the inclusion of these tables as an annex to the standard was originally intended as a cross reference into the main body of the standard. Unfortunately, as the main body of the standard provided minimal guidance on the tailoring permitted based on software level, the tables have become used as a defacto compliance checklist.

| | Objective | | Applicability by Software Level | | | | Output | | Control Category by Software Level | | | |
|---|--|--------|---------------------------------|---|---|---|-------------------------------|-------|------------------------------------|---|---|---|
| | Description | Ref. | A | B | C | D | Description | Ref. | A | B | C | D |
| 1 | Source Code complies with Low Level Requirements. | 6.3.4a | ● | ● | ○ | | Software Verification Results | 11.14 | ② | ② | ② | |
| 2 | Source Code complies with Software Architecture. | 6.3.4b | ● | ● | ○ | | Software Verification Results | 11.14 | ② | ② | ② | |
| 3 | Source Code is Verifiable. | 6.3.4c | ○ | ○ | | | Software Verification Results | 11.14 | ② | ② | | |
| 4 | Source Code conforms to Standards. | 6.3.4d | ○ | ○ | ○ | | Software Verification Results | 11.14 | ② | ② | ② | |
| 5 | Source Code is traceable to Low Level Requirements. | 6.3.4e | ○ | ○ | ○ | | Software Verification Results | 11.14 | ② | ② | ② | |
| 6 | Source Code is accurate and consistent. | 6.3.4f | ● | ○ | ○ | | Software Verification Results | 11.14 | ② | ② | ② | |
| 7 | Output of Software Integration Process is Complete and Correct | 6.3.5 | ○ | ○ | ○ | | Software Verification Results | 11.14 | ② | ② | ② | |

Table 2: Extract from Verification of Outputs of Software Coding and Integrations Process [DO178B]

| Legend | Description |
|--------|---|
| ● | The objective should be satisfied with independence. |
| ○ | The objective should be satisfied. |
| Blank | Satisfaction of objective is at the applicant's discretion. |
| ① | Data satisfies the objectives of Control Category 1 (CC1) |
| ② | Data satisfies the objectives of Control Category 2 (CC2) |

Table 3: Symbol Legend for [DO178B] Tables

As illustrated in the Applicability by Software Level column in Table 2, the requirement for the applicant to demonstrate satisfaction of the objectives is tailored based on the software level. For example, Level A requires that all 66 objectives are met, with some additional independence requirements in satisfying some of the objectives. Level B requires that 65 objectives are met, also with some independence requirements in satisfying some of the objectives. Level C requires that 56 objectives are met, and Level D requires only 27 objectives to be met. Each of these objectives is related to one or more sections in the main body of the standard (note the columns referencing paragraph numbers in the main body of the standard).

There are also numerous implicit objectives or requirements specified within the main body of the standard that aren't directly referenced in the tables. In some cases this has generated confusion as to the applicability to software levels. It has also led to some objectives or requirements being overlooked in certain projects where the applicant (or developer), authorised Designated Engineering Representative (DER – an FAA delegation), and the FAA Aircraft Certification Office (ACO) did not hold a complete understanding of the standard and related guidance.

To address these types of issues, and to provide further clarification on the standard, a number of additional documents have been issued to guide the application of RTCA/DO-178B. These are as follows:

- RTCA/DO-248B – Final Report for Clarification of RTCA/DO-178B Software Considerations in Airborne Systems and Equipment Certification
- FAA Order 8110.49 Software Approval Guidelines
- Certification Authorities Software Team – Position Paper Cast 5 – Guidelines for Proposing Alternate Means of Compliance to RTCA/DO-178B

2.3 Intricacies of RTCA/DO-178B

It is a common misconception that RTCA/DO-178B completely specifies the process and activities which must be conducted to demonstrate compliance with the standard. It is true that the objectives of RTCA/DO-178B are closely coupled to the software lifecycle, and that they do not explicitly distinguish between software functional requirements and software safety requirements. It is also true that the fidelity of the objectives presents some challenges for the use of COTS or previously developed software. However, with the exception of three objectives, RTCA/DO-178B permits flexibility in how the remaining objectives are satisfied, provided the means addresses any criteria identified in the main body of the standard, and that agreement has been reached with the FAA through the Plan for Software Aspects of Certification (PSAC) at the commencement of the program.

For the most part, each of the objectives can be broadly classified as contributing to requirements validity, requirements satisfaction and requirements traceability,

all fundamental elements of demonstrating the appropriateness of software. Annex A presents a comparison of the objectives of RTCA/DO-178B against these criteria.

2.3.1 The Prescriptive Objectives – Test Coverage

Three of the RTCA/DO-178B objectives are, however, worded such that they are explicitly satisfied with a prescribed activity and technique, rather than being presented as an outcome. These objectives relate to verification and are as follows:

- Test Coverage of Software Structure (Statement Coverage) is achieved.
- Test Coverage of Software Structure (Decision Coverage) is achieved.
- Test Coverage of Software Structure (Modified Condition Decision Coverage) is achieved.

RTCA/DO-178B states that the objective of this analysis is to determine which code structure was not exercised by the requirements-based test procedures. Note the implicit assumption of testing over static code analysis or formal proof here. Structural coverage analysis may reveal code structure (statements, decisions, and control and data flows (note, some data flows only)) that is not exercised during the requirements based testing, which RTCA/DO-178B states may be the result of:

- Shortcomings in requirements-based test cases or procedures, in which case the requirements based test cases should be supplemented or test procedures changed to provide the missing coverage. If there are significant shortcomings, then the method used to perform the requirements-based coverage analysis may need to be reviewed.
- Inadequacies in software requirements, in which case the software requirements should be modified and additional test cases developed and test procedures executed.
- Dead code, which should be removed and an analysis performed to assess the effect and the need for reverification.
- Deactivated code, in which case analysis and testing should show that the means by which such code could be inadvertently executed are prevented, isolated, or eliminated.

3 Criticisms of RTCA/DO-178B and Addressing the Criticisms within the ADF Framework

As mentioned in the introduction, RTCA/DO-178B is often the centre of debate or criticism for a number of reasons. Ironically, the debate and criticism can broadly be divided into several positions:

- those that believe that RTCA/DO-178B is insufficient; and

- those that believe that RTCA/DO-178B is too onerous.

On the surface, these two positions seem to be at odds with each other. A simplistic view may be that since RTCA/DO-178B is somewhat central to the extremes of criticism, then it is perhaps about right. In addition, it is widely accepted by National Airworthiness Authorities such as the FAA and EASA, and there is evidence that it is effective in this context [NTS06]. However, for the ADF, such arguments, while somewhat compelling, are insufficient on their own, and a more comprehensive justification is required.

To understand the motivation for the criticisms, one must understand the source of the criticisms. The first position is usually levied by academics, researchers or consultants in the high integrity systems and software domain, or formal methods domain. This is not to say that their claims do not have some substance, however it does help to illustrate the source and nature of their criticisms. The second position is predominately levied by systems and software development contractors with primarily a cost and schedule driven imperative. In many cases, lethargy promoted by an ignorance of relevant software engineering practices, and perceived cost and schedule impacts, has been a barrier to many of these contractors adopting rigorous software engineering practices, or keeping pace with improvements in high integrity approaches. Again, it is not to say that their claims are totally unfounded, but it does also illustrate the motivations for their claims.

This section examines these positions and the focuses of criticism on which they are based. It also examines how the ADF framework defined by DGTA addressees each of these criticisms.

3.1 RTCA/DO-178B is Insufficient

In Section 3, it was mentioned that the major source of insufficiency criticisms are usually from academics, researchers, or consultants in the high integrity systems and software domain, or formal methods domain.

The following subsections examine the most common of these insufficiency criticisms.

3.1.1 Absence of Mandatory Formal Methods

RTCA/DO-178B states that formal methods involve the use of formal logic, discrete mathematics, and computer languages (grammar and vocabulary) to provide evidence that the system is complete and correct with respect to its requirements, and a determination of which code, software requirements or software architecture satisfy the next higher level of software requirements. The goal of applying formal methods is to prevent and eliminate requirements, design and code errors throughout the software development processes. Table 4 presents a summary of common formal methods techniques.

| Formal Methods Techniques and Methods |
|--|
| Semi-formal Specification / Formal Specification |
| Formal Development and Formal Verification – proofs of properties or refinement from the specification to a program |
| Theorem Provers – fully formal machine-checked proofs |
| Denotational Semantics, Operational Semantics, Axiomatic Semantics |
| Human Directed Proof, Automated Proof |
| Model Checkers |
| Z Notation, Vienna Development Method (VDM), B Method, Abstract State Machines (ASM), Abstract Machine Notation (AMN). |

Table 4: Formal Methods Techniques and Methods

Firstly, it should be mentioned that RTCA/DO-178B does not prohibit the application of formal methods. Certainly, RTCA/DO-178B does not have explicit objectives that prescribe formal methods, however RTCA/DO-178B does allow formal methods to be proposed as an alternate method for satisfaction of one or more objectives.

In their most thorough application, formal methods could be equivalent to exhaustive analysis of a system with respect to its requirements, within the constraints to which the target hardware and operating environment have been accurately modelled. However, it is these constraints that limit the applicability of formal methods.

3.1.1.1 Application to Problem Domain

Formal methods are not yet universally applicable to the problem domains and technologies used in critical systems. They may also only be only partially applicable to a problem scope. Formal methods are also based on formal languages that are closed, in that they have no inherent real world meaning [KnG07]. A formal language is a collection of symbols and a set of rules for manipulating them. However, to be useful, they are still required to be linked to real world concepts and objects using natural language, a medium that cannot be formally reasoned about or verified [KnG07]. In addition formal language semantics are usually specified around a particular class of representations or problems, and so one formal language is usually insufficient to describe the complete set of required behaviours of most avionics systems.

The general view of the formal methods community is that a model should be specified using a formal language, such as Z or B for example, and then through a process called refinement, refined into implementation (source code) while proving certain properties of interest still hold true. Unfortunately, the types of models that can be built are very much constrained by the formal language that describes it. There are a number of challenges associated with describing real-time avionics systems using such languages, and therefore it may not be possible to apply formal methods to these types of problems in their entirety. However, formal methods are often well suited to small parts of the problem, or implementation.

In addition, there is very limited guidance on determining what might be an acceptable scope of application (or thoroughness) of formal methods, either in the standards that mandate it (e.g. Def (Aust) 5679 and DEF STAN 00-55 (obsolete)), or in the literature. On this basis, formal methods cannot be stated to be universally applicable to all functions, systems and their failure modes; and it would be overly prescriptive to make formal methods mandatory in all cases.

3.1.1.2 Formal Methods and Safety

Because of the constraints on formal models relating to target hardware and operating environment, formal methods does not address a number of significant sources of error that contribute to the safety of systems. In fact, there is little evidence that formal methods alone would have prevented some of the accidents and incidents attributable to software. Safety-related software errors arise most often from discrepancies between the documented requirements specifications and the requirements needed for correct functioning of the system; and misunderstandings about the software's interface with the rest of the system. Software-related accidents have occurred when the software satisfied its specification and when the operational reliability of the software was very high. This is because documented requirements specify behaviour that is not safe from a system perspective, requirements do not specify some particular safety behaviour, or the software has unintended (and unsafe) behaviour beyond what is specified in the requirements.

Formal methods allow us to build an unambiguous model of the system, within certain constraints. It allows us to determine if the model is consistent with itself, and allows us to determine if the implementation conforms to the model. However, formal methods tell us very little about whether the model is right, or safe, unless those properties have been captured in the model. In addition, it doesn't tell us what those properties should be. Unfortunately, most of the important real world interfaces and environmental parameters (particularly for avionics systems) cannot be captured in formal models at this time, or are prohibitively time consuming to do so. For example, the time taken to build these detailed models may be better directed to analysis of safety properties of the system.

Merely prescribing formal methods, as has been done in a number of standards, without prescribing objectives related to the properties of the system that formal methods should seek to prove, goes against the objectives based model adopted by RTCA/DO-178B and is of limited value to developers and certification authorities alike. It also confuses the relevance, applicability and application (or thoroughness) constraints of formal methods.

3.1.1.3 Complementary to Testing

Because there are limitations to the extent to which formal methods can address behaviour on real hardware, in the target environment, formal methods are in their present maturity complementary to testing. Formal

methods also do not provide assurance that these real world behaviours are correct. Testing is still required to demonstrate real world behaviours, on real hardware, in the target environment. On the other hand, it is acknowledged that "testing alone is a completely hopeless way of showing that software does not contain errors" [Mar08]. Therefore the ideal approach is a complementary combination of evidence types: analysis and testing.

Despite the criticism of RTCA/DO-178B regarding testing (refer Section 3.1.3), RTCA/DO-178B objectives are much broader than testing alone. RTCA/DO-178B includes a number of objectives that are routinely satisfied by reviews of relevant analyses, or by the analyses themselves. Should an applicant wish to employ formal methods, then it would predominately support objectives relating to:

- definition and development of requirements (high and low level),
- accuracy and consistency of requirements (high and low level), within the constraints of the formal model,
- traceability of high level requirements to low level requirements to source code (through refinement), and
- source code complying with requirements, within the constraints of the formal model.

All of the test objectives/methods within RTCA/DO-178B are still relevant to a development where formal methods have been applied. There has been some argument to remove the structural test coverage requirements for programs where formal methods have been applied. However, few of these arguments completely address the purposes of structural coverage which includes accounting for compiler generated object code. For example, RTCA/DO-178B states that "the structural coverage analysis may be performed on the Source Code, unless the software is Level A and the compiler generates object code that is not directly traceable to Source Code statements". In this case, additional verification should be performed on the object code to establish the correctness of such compiler generated code sequences. In this context the structural coverage analysis also plays a role in validating the compiler generated code, something that would not be accomplished by the formal methods approach without the addition of additional analysis activities.

Programming language choice plays a role here also, and although not prescribed by RTCA/DO-178B, it does impact the ease by which this objective is satisfied. For example, languages such as C have fairly minimal run-time machines, and as such, provided that compiler optimisations are disabled, then establishing source to object code traceability is relatively straightforward to complete. Unfortunately, languages such as Ada, for which there are many additional analysability (including tools built on formal approaches) benefits over C and similar languages, have quite substantial run-time machines, and thus present numerous challenges, albeit not insurmountable, in constructing this traceability. Hence it has been possible to observe a general

favouritism for languages such as C and C++ over languages such as Ada used for developments under RTCA/DO-178B.

This section has provided some discussion as to why the structural coverage objectives are relevant even with the application of exhaustive source code analysis, as supposed to be provided by formal methods.

3.1.1.4 DGTA Guidance on Formal Methods

DGTA encourages the use of formal methods where it is appropriate, and its application should be proposed and negotiated through the Plan for Software Aspects of Certification (PSAC). However, DGTA also recognises that the safety-related software errors arise most often from discrepancies between documented requirements specifications and the requirements needed for correct functioning of the system; and misunderstandings about the software's interface with the rest of the system. Therefore the extent of application of formal methods needs to be carefully balanced with software safety analysis that might better address the aforementioned concerns. The application of formal methods should not be seen as a 'silver bullet' for software safety, and it is certainly not a substitute for such analysis.

3.1.2 **Absence of Mandatory Static Code Analysis**

Static Analysis is the analysis of source code before it is executed. Techniques include control flow analysis, data flow analysis, symbolic execution, checking the source code against a formal mathematical specification, and checking conformance against a coding standard or language subset. [Bar02] suggests that by using a combination of static analysis techniques, a variety of properties can be guaranteed about a program. An example of a well-known tool that performs such analysis is the SPARK Examiner (for Ada). Other tools include PC-Lint, and Parasoft's Static Code Analysis toolset.

In some respects, the argument for static analysis is linked to the previous section's arguments for formal methods. Those properties of static analysis that relate directly to the refinement process and theorem proving shall not be addressed in this section as the arguments that apply to formal methods apply equally to the associated analysis used for refinement. However, static code analysis can be used to analyse software with respect to additional criteria (control flow analysis, data flow analysis, checking conformance against a coding standard or language subset, etc) as mentioned above.

As with formal methods, RTCA/DO-178B does not prohibit the application of static code analysis. Certainly, RTCA/DO-178B does not have explicit objectives that prescribe static code analysis, however RTCA/DO-178B does allow static code analysis to be proposed as a method for satisfaction of one or more objectives.

In fact there are a number of objectives that can be satisfied in whole or in part by static code analysis. For example, in addition to those relevant to formal methods

(identified in Section 3.1.1.3), static code analysis can be proposed for objectives relating to the following:

- source code is verifiable,
- source code is accurate and consistent
- source code conforms to standards, and
- coverage of data and control coupling.

Tools such as SPARK, also permit the compliance with low level requirements and traceability to low level requirements to be established through the SPARK annotations and hypothesis checking.

It should be noted however that there are significant limitations with applying several important aspects of static analysis retrospectively [SaL06]. This has created significant challenges for some UK MoD programs accepting software from US vendors while trying to apply the framework of Def Stan 00-55 (now obsolete), and its associated prescriptions [SaL06].

DGTA encourages static code analysis to be adopted for all programs that involve the new development of safety critical or safety related software. Static code analysis tools are widely available and affordable. While static code analysis won't find all the errors most related to the safety of the software, there are several benefits that it provides. Static code analysis does permit the programmers and testers to focus greater effort on those activities that relate to identifying requirements validity and satisfaction problems directly affecting the safety of the software. This prevents them becoming overwhelmed in code reviews and testing with identifying inadvertent implementation problems (conventional bugs), which static code analysis tools readily detect.

3.1.3 **Ineffectiveness of MC/DC Testing**

The Modified Condition Decision Coverage (MCDC) objective of RTCA/DO-178B is arguably the most frequently and widely debated requirement of the standard. The concept behind MCDC testing is to place the software in as many situations as possible to see if it behaves inappropriately. At a more detailed level, it attempts to exercise each data flow that directly affects a control flow within the software, in an effort to identify as many faults or errors as possible relating to the control (the logic implementing critical behaviours of the software), and the data that feeds these decision points.

Some studies (e.g. [DuL00], [KaB03], [HaV01], [ViB02]) have been carried out on the effectiveness of MCDC testing, and these studies generally confirm that MCDC testing does find faults (requirements and implementation) that the other testing approaches within RTCA/DO-178B do not find. However other studies, such as the Qinetiq study referenced by [Mar08] found that there was "no significant difference" between MCDC and the Decision Coverage objective at Level B. It is the author's experience that MCDC testing does find faults; but which faults, and how do they relate to the safety of the system is something that requires further assessment?

While finding more faults seems intuitively sensible, there is argument as to whether MCDC is the most effective (time and cost) way of finding these faults, and whether it finds the faults (requirements and implementation) most related to the safety of the system in question [DuL00]. Certainly MCDC testing carried out in ignorance of the application domain and the knowledge of requirements most related to the safety of the system, has reasonable likelihood that it may not actually identify the most critical faults.

If an applicant were to propose an alternate approach that through analysis, perhaps even formal methods and requirements based testing which provided adequate assurance that the errors (requirements and implementation) most related to the safety of the system were addressed, then there may not be reasonable motivation, aside from the explicit requirement in the standard, to achieve MCDC. For example, an applicant might wish to propose Reinforced Condition/Decision Coverage (RC/DC), as developed by [ViB02], in place of MCDC on the basis that it addresses some, but not all, of the criticisms of MCDC. Alternatively, an applicant might propose specific metrics on the completeness of normal and robustness testing of critical control and data flows related to software safety requirements, on the basis that it identifies faults related to the safety of the system. DGTA has always encouraged the use of the PSAC for developers to provide such arguments for alternative approaches.

Much of the argument relating to verification in RTCA/DO-178B centres on the MCDC requirement and its relevance to finding errors related to safety. However, an alternative perspective might help diffuse the debate. For the most part, the objectives associated with identifying behaviours of the software most related to safety are included from Level C (the objective where it is first acknowledged that the software failure modes may be adverse to human safety). At Level C testing should already be addressing the complete suite of:

- normal and robustness testing (of high and low level software requirements – which infers testing of all software behaviours if the low level requirements are adequate),
- error sources associated with the software operating in the target computer environment during software/hardware integration testing,
- inter-relationships and interactions between software components, including adverse ones; and
- known error sources in the implementation of low level requirements.

Therefore, the fact that MCDC testing is not finding a plethora of additional faults is not actually the concern. If the normal and robustness testing has been comprehensive (included from Level C), then it is possible that the gap in MCDC coverage will be small and insignificant. It is the author's opinion that much of the debate about MCDC is misdirected, and should instead be focussed on the adequacy of normal and robustness testing, including the analysis that underpins

it, and variations in the application of this type of testing. MCDC is merely one of the cross checks of the adequacy and comprehensiveness of these other types of analysis and testing.

Therefore, provided the faults most related to the safety of the system are being identified through the combination of analysis and verification techniques embodied by the objectives from Level C upwards to Level A, then the objective of showing the system is acceptably safe can be achieved, irrespective of the criticisms of the effectiveness of MCDC testing.

3.1.4 Absence of Specific Requirements or Objectives relating to Software Safety Analysis and Software Safety Requirements

As previously mentioned, safety-related software failures arise most often from:

- discrepancies between documented requirements specifications and the requirements needed for correct functioning of the system; and
- misunderstandings about the software's interface with the rest of the system

Software-related accidents have still occurred when the software satisfied its specification and when the operational reliability of the software was very high. This is due to:

- requirements that specify behaviour that is not safe from a system perspective;
- requirements that do not specify some particular safety behaviour; or
- software that has unintended (and unsafe) behaviour beyond what is specified in the requirements.

Therefore, the identification and allocation of software safety requirements in the context of the system are fundamental to the realisation of acceptably safe software intensive systems.

Unfortunately, RTCA/DO-178B is not overt or explicit in objectives relating to software safety analysis and software safety requirements. This is not to say they are totally absent though. Carefully managed, RTCA/DO-178B can be used as a framework for assuring the complete and correct inclusion of software safety requirements. Considering both the objectives in RTCA/DO-178B, and the assessment presented at Annex A, then a significant number of objectives actually contribute to requirements validity, including that of safety requirements. The following objectives are most relevant to capturing software safety requirements:

- high-level and low-level requirements are developed;
- derived high-level and low-level requirements are defined;
- high-level and low-level requirements are accurate and consistent;

- high-level and low-level requirements are compatible with target computer;
- high-level and low-level requirements conform to standards;
- high-level requirements are traceable to system requirements.

RTCA/DO-178B also includes structural coverage analysis which may reveal code structure (statements, decisions, and control and data flows) that is not exercised during the requirements based testing; and which RTCA/DO-178B states may be, in part, the result of:

- inadequacies in software requirements, in which case the software requirements should be modified and additional test cases developed and test procedures executed.

However the structural coverage's effectiveness is constrained to those software behaviours that are already in the software, perhaps because the domain knowledge of the developer prompted the inclusion of code, or another part of the requirements and design process broke down with respect to maintaining traceability. The identified inadequacies do not necessarily relate to the complete set of software behaviours that are required to assure the system acceptably safe. Thus although these objectives contribute to requirements validity, including software safety requirements, these objectives are not a measure of completeness or accuracy of the software safety requirements necessary to make the system acceptably safe.

Thus, on their own these objectives and the activities and evidence that are normally used to demonstrate compliance, are not explicit and therefore usually insufficient to demonstrate that the set of identified and allocated software safety requirements is complete and correct. In practice in the civil avionics sector, it is certainly rare that the types of systematic software safety analysis that do identify software safety requirements are proposed by applicants without pressure from certification authorities. Many applicants prefer to rely on the experience and competency of their engineering staff in assuring the completeness of appropriate software behaviours including software safety requirements.

As described in Section 2.1, and further discussed in Section 4.1, RTCA/DO-178B is not intended to be applied in isolation. RTCA/DO-178B is intended to be coupled to a safety assessment process, such as that defined by [ARP4754] and [ARP4761], and used to satisfy FAR/JAR 25.1309. Therefore it is unreasonable to expect that RTCA/DO-178B should be on its own wholly responsible for assuring the completeness and correctness of requirements, including safety requirements.

[ARP4754] describes a process (Figure 2) whereby item safety requirements are allocated to software from preliminary system safety assessments, which have in turn been influenced by identified aircraft failure conditions and system architecture. On this basis, some additional aspects of the identification and allocation of

software safety requirements are addressed, particularly with regards to system and aircraft behaviours. However, such an approach still doesn't address requirements that do not specify some particular safety behaviour, and the software having unintended (and unsafe) behaviour beyond what is specified in the requirements.

In simple terms, RTCA/DO-178B is not explicit enough on its own, and the associated guidance in relevant system safety standards (ARP4754) is also insufficient. In recognising this potential deficiency, DGTA recommends that an IEEE1228 Software Safety Plan be used to document the planned software safety activities [AAP7001.054]. Some programs have elected to include such content in the PSAC instead. Section 4.1 provides further information on how DGTA addresses this deficiency with RTCA/DO-178B.

3.1.5 Assumptions Underlying the Design Assurance Level Definition are Questionable

In general, there is "little evidence" that software of different integrity levels does have failures rates of "integrity level order" [McD01]. There is also wide debate amongst the industry regarding the underlying philosophy and rules for allocating integrity levels. There are significant differences in the specific processes and techniques recommended by different standards. For example [McK06] points out that "Defence Standard 00-55 emphasises the use of formal verification techniques for the highest integrity level, whilst DO-178B concentrates on human review and rigorous testing".

In understanding and addressing this criticism, there are two issues requiring consideration. The first is issues with the consistent application of RTCA/DO-178B between projects which may lead to a perceived variation in the effectiveness of RTCA/DO-178B. The second is with the apportionment, and adequacy of objectives at each level within RTCA/DO-178B.

3.1.5.1 Inconsistent Application of RTCA/DO-178B

[JTM07] notes that consistency in the application of RTCA/DO-178B has been a problem. It is acknowledged that some variation in the interpretation of certain objectives, and the means of demonstrating compliance has led to cases where software was developed without specific recognition of a number of fundamental software assurance principles. For example the author is aware of the following:

- Misunderstanding of the differences between reviews (qualitative assessment of correctness of output of a process guided by a checklist or review aid) versus analyses (repeatable evidence of correctness through detailed examination via systematic means of functionality, performance, traceability and safety) has led to some objectives being presumed to be satisfied solely by reviews, when a combination of analysis and reviews of the outputs was clearly more appropriate.

- The degree to which normal and robustness testing criteria are actually addressed during requirements based testing has been immensely variable, including variability in the focus on the faults identifiable at the various levels of software and hardware integration testing.
- The degree to which the software architecture is appropriate and avoids design constructs that would not comply with system safety objectives has been variable.
- The degree to which software safety requirements are identified and allocated under the objectives identified in Section 3.1.4 has been variable.

Therefore, it is quite possible that most observations regarding software of different integrity levels not having failures rates of “integrity level order” are actually attributable to the inconsistent application of the standard, rather than the definition of the standard itself, and the types of activities that underpin it. This has been addressed in some part by the release of RTCA/DO-248B Final Report for Clarification of DO-178B and FAA Order 8110.49 Software Approval Guidelines. However, inconsistencies still remain.

Unlike the FAA system that has many different Designated Engineering Representatives (DERs) and Aircraft Certification Office’s (ACOs) involve with the certification of software, where the opportunity for variability is great, the ADF has relatively fewer agencies and personnel involved in these aspects of software development and certification. Systems Certification and Integrity (SCI) at DGTA oversees all of the most critical software acquisitions and modifications, including those developed to RTCA/DO-178B. In addition, DGTA publishes guidance in [AAP7001.054] or in the form of DGTA Papers to provide additional clarification where warranted. This contributes to assuring greater consistency in interpretation and application of RTCA/DO-178B, particularly for the most critical software.

3.1.5.2 Apportionment and Adequacy of Objectives

As for the apportionment and adequacy of objectives at each DAL, it is firstly necessary to examine in general terms the construction of RTCA/DO-178B objectives. Table 5 presents the apportionment of objectives in RTCA/DO-178B. Each row should be interpreted as cumulative with the rows above it. For example, Level C consists of those objectives identified as applicable to Level C, plus those for Level D.

The objectives fundamental to assuring the validity and satisfaction of software requirements, including safety requirements, are incorporated at Level C. This is achieved by ensuring the following:

- Correctness, completeness, consistency and traceability is established between system requirements, high level requirements, low-level requirements and source code.
- All behaviours documents in requirements at these different levels of abstraction are verified, including with respect to robustness criteria.
- Any code or requirements which do not meet these criteria are accounted for and resolved.

Beyond this, Levels A and B are targeted at addressing key sources of error in addressing the fundamental objectives (already incorporated at Level C), either with additional rigour, or through complementary objectives/activities. There are not significant gaps in the principles behind the objectives of Level C that Levels A and B address which would be required to address fundamental safety concerns. In simple terms, if the objectives of Level C have been comprehensively addressed, then often the objectives of Level A or B will mainly provide additional evidence to assist with the trustworthiness of evidence presented, rather than further substantiating the relevance and coverage of claims (at least implicitly) being made. For systems with catastrophic and hazardous hazards, it is intuitive that evidence presented should be trustworthy, and be subject to complementary activities and reviews.

The objectives of Level B assure that development staff didn’t make errors (independent assessment) in the critical aspects of the development relating to requirements, including software safety requirements, being valid and satisfied. Further to this, Level A objectives assure (more independence) that development staff didn’t make errors in further additional areas of the development which are known to be the most challenging, and therefore most likely to have been subject to error and the introduction of faults. The differences between statement coverage, decision coverage and MCDC are in some respects almost irrelevant. If the requirements have been completely specified, and the verification of these requirements has completely addressed normal and robustness cases, then actually this analysis would likely find few or no problems.

| Level | Objectives Unique Beyond Lower DALs |
|-------|--|
| D | Planning Process (processes and activities defined) Configuration Management Software Quality Assurance (processes comply with plans and standards, conformity review) High Level Requirements (developed, defined, accurate, consistent, comply with system requirements, traceable, coverage) Low-level Requirements (developed, defined) Source Code (developed) Executable Object Code (integrated and compatible, complies and robust) Verification (coverage of high-level requirements) Certification Liaison Tool Qualification Partitioning Integrity |
| C | Planning Process (process transition criteria, lifecycle environment, development standards, compliance) High Level Requirements (verifiable, conform to standards, algorithms accurate) Low Level Requirements (accurate, consistent, comply with high-level requirements, traceable, coverage, algorithms accurate) Software Architecture (compatible with high-level requirements, consistent, verifiable, conforms to standards) Source Code (complied with low-level requirements and software architecture, conforms to standards, traceable, accurate and consistent) Executable Object Code (complies and robust with low-level requirements) Verification (procedures, results, coverage of low-level requirements, statement coverage, data coupling and control coupling) |
| B | Independence (statement coverage, decision coverage, data and control coupling, executable object code and source code complies with low level requirements, high and low level requirements compliance, accuracy and consistency) High Level Requirements (compatible with target computer) Low Level Requirements (compatible with target computer, verifiable) Software Architecture (compatible with target computer, verifiable) Source Code (verifiable) Verification (decision coverage) Life Cycle Transition Criteria |
| A | Independence (modified condition decision coverage, executable object code robustness with low level requirements, source code complied, with software architecture, accuracy and consistency, partitioning integrity, software architecture is compatible with high level requirements and consistent) Verification (modified condition decision coverage) |

Table 5: Apportionment of RTCA/DO-178B Objectives

DGTA is satisfied that for all software that is safety critical and safety related, RTCA/DO-178B provides an adequate framework for addressing requirements validity, requirements satisfaction and requirements traceability, when applied within the ADF framework. Note though that significant diligence is required to assure that some of the key objectives are satisfied by rigorous means. This is achieved through rigorous certification authority oversight, or through the employment of IV&V agents activating on behalf of the certification authority.

3.2 RTCA/DO-178B is Too Onerous

In Section 3, it is mentioned that the major source of onerousness criticisms are usually from systems and software development contractors with a cost and schedule driven imperative. In many cases, lethargy, promoted by an ignorance of relevant software engineering practices, and perceived cost and schedule impacts, has been a barrier to many of these contractors to adopt rigorous software engineering practices, or keeping pace with improvements in high integrity approaches.

The following subsections examine the most common of the onerousness criticisms within the context of the ADF framework.

3.2.1 Excessive Requirements Specification and Traceability Fidelity Requirements

Many contractors argue that the requirements specification and traceability fidelity requirements of RTCA/DO-178B are onerous and excessive.

DO-178 requires explicit development of and traceability between high level software requirements, low level software requirements (Level A-C), source code (Level A-C), and object code (only for Level A software). This includes the definition and traceability of derived high and low level software requirements, as well as consideration for the relationship between software architecture, functional requirements, and the target computer.

The motivation for this level of fidelity, is not just as a design tool to assist the system and software designers with developing a good design. It is primarily included to ensure that as the requirements are refined towards implementation, each behaviour that constitutes the requirements at each level of abstraction, is systematically accounted for.

But why do all the software behaviours need to be accounted for? For evidence to be presented that all behaviours of the software are acceptable, and do not lead to unacceptable software failure conditions, then at the very least all the behaviours of the software should be explicitly documented. Explicit documentation of

software behaviours then permits reasoning about their suitability for the safety of the system, or at the very least non-interference with the behaviours that are important to the safety of the system.

It is not possible to have confidence that all of the behaviours are acceptable if there is any significant gap between requirements and implementation – i.e. there are software behaviours that are not accounted for. Unfortunately the most rampant perpetrators of excessiveness criticisms of RTCA/DO-178B, often themselves have significant gaps between high level requirements and source code. In some cases the gaps have been so great, that it has not been possible to even retrospectively determine the traceability of requirements to implementation, and account for all the behaviours of the software. In these cases, short of substantial analysis and reverse engineering of the software and requirements (both an expensive undertaking), it is almost impossible for the ADF to successfully complete certification and acceptance.

In some cases intellectual property constraints are suggested as motivation for a design agency to not provide this fidelity of requirements and design disclosure. In other cases, the process of developing critical software is suggested as a creative approach, not itself compatible with such rigour requirements. Both of these arguments are indefensible. If an organisation is not prepared to provide complete disclosure through the definition of requirements and design for all behaviours of their software and show that they are acceptable, then it is not possible for the certification authority (DGTA in this case) to make a determination of the safety of this software, and the software should be considered unsafe. Therefore any approach that does not achieve this should be considered unacceptable. For in-service modifications and upgrades, similar consideration should apply, noting that sometimes there are constraints regarding options for contractors to undertake the work.

Furthermore, the level of explicitness in requirements specification and traceability also becomes fundamental to determining the acceptable extensiveness of verification. This is discussed further in Section 3.2.2.

3.2.2 Excessive Verification Requirements

It has already been established that irrespective of the techniques and methods applied to software development, testing will always be required to gain confidence in the behaviour of the software on the target hardware and in the intended operating environment.

However, many contractors argue that the testing requirements of RTCA/DO-178B are onerous and excessive. These claims are usually directly related to limitations in the contractor's normal and robustness test cases, as well as no structural test coverage assessment (often due to software tool limitations in their existing software and systems laboratories). Such a position is usually closely followed by an unfounded claim that their testing processes are equally as rigorous as those in RTCA/DO-178B.

DGTA's expectation is that the completion of testing (or all verification and validation for that matter) should be determined based on a defensible engineering argument as to why testing is complete, rather than being based around the following factors:

- cost and schedule,
- consensus of program managers,
- broad consensus of the programmers and testers, or
- any other non-engineering based argument.

These factors are certainly drivers for the profitability of any given program, however are not considered sufficient reason for the completeness of verification and validation activities.

RTCA/DO-178B provides a defensible argument as to when testing is complete, by specifying requirements completeness criteria, requirements coverage criteria, coupled with additional implementation related coverage criteria (structural coverage). For requirements-based test case selection, RTCA/DO-178B provides consideration for the extent of normal and robustness tests, as well as the extensiveness of requirements-based low level testing, software integration testing, and hardware/software integration testing. Software faults and failures that affect safety are also addressed in each of these contexts. The structural coverage analysis provides a measure of possible short comings in the requirements-based test cases or procedures, inadequacies in software requirements, dead code, deactivated code, correctness of compiler generated object code that is not directly traceable to code statements (e.g. the run-time machine).

While DGTA is normally prepared to accept alternate approaches to software testing, the argument must consider at least those factors identified in RTCA/DO-178B.

4 Application of RTCA/DO-178B

Section 3 has identified and addressed a number of criticisms of RTCA/DO-178B. In doing so, the reader will note that in order to appropriately address certain issues, additional guidance is required. The following sections provide an overview of some of the key guidance on the application of RTCA/DO-178B, as used by the ADF.

4.1 DO-178B Is Not Applied In Isolation

[AAP7001.054] identifies MIL-STD882C or FAR/JAR2X.1309 as the ADF's preferred system safety paradigms. While both of these paradigms require software to be considered within the system context, in isolation neither provides adequate guidelines for conducting the software safety activities required to assure that the software will execute with an acceptable level of safety. [AAP7001.054] defines a number of key issues, the most relevant of which are the following:

- A Software Safety Program (SwSP) should be established to coordinate hazard identification and

mitigation efforts for hazards with software-related causal factors.

- A Software Assurance standard (e.g. RTCA/DO-178B) is required for the development of all software that is safety related.
- Software process standards (e.g. [MILSTD498], [IEEE12207]) should be applied to the development of software for airborne and related systems.

Noting the issues identified in Section 3.1.4, the ADF promotes the use of IEEE STD 1228-1994, as one means of addressing the gaps commonly found between system safety standards and software assurance standards such as RTCA/DO-178B. It also addresses the aforementioned key issue for a SwSP. IEEE STD 1228-1994, ‘Standard for Software Safety Plans’, provides an industry accepted standard for the preparation and content of a Software Safety Program Plan (SwSPP). The IEEE1228 Software Safety Plan should propose the software safety analyses, and why they are sufficient to identify and allocate the complete and correct set of software safety requirements. Furthermore, the software safety plan should address requirements satisfaction aspects, particularly where safety requirements are difficult to verify in their target environment (e.g. those implemented with exception handlers).

To provide an understanding of the software failure modes that might be relevant, and importantly what architectural considerations and software assurance activities are required to provide evidence of the absence or handling of these identified failure conditions, it is necessary to conduct some form of software safety analysis. There are numerous software safety analysis techniques that could be applied to such a system. Table 6 presents a list of commonly adopted and well known software safety analysis techniques, although it is not intended to be a complete list.

| Software Safety Analysis Techniques |
|---|
| Software Functional Failure Analysis (SFFA) [ARP4761] |
| System Software Failure Modes and Effects Analysis (SSFMEA) |
| Software Fault Tree Analysis (SFTA) – derivation of the weakest preconditions |
| Software HAZOP (Def Stan 00-58 Computer Hazard and Operability Studies CHAZOP) |
| Software Hazard Analysis and Resolution in Design (SHARD) [Pum99] |
| Markov Analysis |
| Petri Net Analysis |
| Software Sneak Analysis (General Dynamics / Boeing) |
| Failure Propagation and Transformation Notation (FPTN) [Pum99] |
| Low-level Interaction Safety Analysis (LISA) [Pum99] |
| What-If Analysis [JSSSC] |
| Safety Critical Path Analysis of Functional Flow Diagrams, Data Flow Diagrams, Call Trees [JSSSC] |

Table 6: Software Safety Analysis Techniques

While it is possible to apply aspects of each of these techniques to analyse software architectures, design and implementation, and indeed the measured application of a number of these techniques would probably be necessary

for the developer to provide sufficient evidence as part of a safety case, it is not in DGTA’s interest to prescribe which techniques must be applied in all cases. It is up to the applicant to provide a compelling argument why the combination of techniques and methods is sufficient to identify and allocate the complete and correct set of software safety requirements. However, this does not prevent DGTA from having preferred approaches.

To achieve certification it is necessary to communicate an understanding to the certification authority of how the software might fail and what might be done to either ensure it can’t or doesn’t fail (failure conditions are absent); or if it can, then verify that it is sufficiently unlikely to fail in a non-benign way (failure conditions are handled). Such understanding should then provide insight into what evidence is required to provide sufficient confidence in these aforementioned properties.

When proposing software safety analyses techniques, applicants should be cognisant of the strengths and weaknesses of the various techniques. While some techniques are useful in a functional context (e.g. SFFA), they are often insufficient on their own to provide confidence of completeness. [AAP7001.054] suggests that guidance on the development and implementation of an effective Software Safety process, including relevant techniques, can be sourced for the US Joint Software System Safety Committee Software System Safety Handbook [JSSSC]. Furthermore, DGTA has found that techniques, such as the SHARD technique [Pum99] (refer Table 6) is particularly relevant to developing the necessary understanding as it considers failure modes, their causes, effects, and potential detection or protection means.

SHARD employs a series of guidewords to classify how the information flows and associated communication events (and associated services) might deviate from their intended forms. These are as follows:

- Omission - Service not delivered.
- Commission - Service delivered when not required.
- Early - Service delivered, but early.
- Late - Service delivered, but late.
- Value - Service delivered, but with incorrect value.

SHARD requires that the system be analysed “backwards” from the outputs (i.e. identify the system level effects first) back towards the inputs. The internal and input deviations are expressed in terms of how they cause or contribute to deviations in downstream items already investigated. Further information on the SHARD technique, and its use as a foundation of a software safety case, can be found in [Pum99] and [Wea03].

In addition to systematic software safety analysis, there are numerous sources of generic software safety requirements that can be used to complement the process. For example, [JSSSC], and [Lev95] are two sources that include detailed lists of relevant software safety requirements.

Fundamental to developing acceptably safe software is the need for the systems engineers and software developers to have adequate domain knowledge of the systems and problems with which they are engaged. DGTA recommends that an assessment of the applicant's software development capability, including domain knowledge, be conducted as part of the tender evaluation and contract negotiation process. Such an assessment will also examine the safety culture within the organisation to determine if the non-systematic (experience based) activities contributing to software safety can be trusted.

4.2 Test Coverage Objectives

According to RTCA/DO-178B, verification of airborne software has two complementary objectives. One objective is to demonstrate that software satisfies its requirements. The second objective is to demonstrate with a high degree of confidence that errors which could lead to unacceptable failure conditions have been removed. Noting that the prescription of activities against these two objectives is scaled based on software level within the standard, it is worth considering the approach in general terms.

The first objective is largely supported through definition of and verification against high-level requirements. Where insufficient disclosure or ambiguities exist within the high level requirements, then refinement and further definition of and verification against is required in translation to low-level requirements. Therefore, it follows that provided the developer can adequately disclose the requirements at the prescribed level of detail, then this objective is relatively straight forward to satisfy.

The second objective, however, is not quite as intuitive. It deals with eliciting properties about the software which don't necessarily follow from the set of already defined high and low level requirements, with focus on those properties that could potentially lead to unacceptable failure conditions. Eliciting these properties permits one of two outcomes: either the behaviour is appropriate, in which case it should be captured in the high and/or low level requirements; or the behaviour is inappropriate, in which case the software design and implementation should be changed to remove the behaviour. RTCA/DO-178B approaches this through prescribing requirements coverage analysis and software structural coverage analysis. Furthermore, establishing requirements traceability from low-level requirements to source code, and to object code, supports providing an understanding of software properties commensurate with this second objective.

Some organisations believe that DGTA does not require structural code analysis, however such an observation does not consider the complete circumstances. There are cases where DGTA has not required structural coverage for some safety related software. However, in these few cases, additional activities have been proposed (as suggested in an example in Section 3.1.3) to compensate for this deficiency. Furthermore, additional oversight has then been applied to the fidelity of requirements, explicitness of traceability and the extent of normal and

robustness testing to ensure that the coverage would be comprehensive anyway. An additional activity to identify dead and deactivated code is also necessary to assure the adequate identification and resolution of such code. Proposals for these types of alternate approaches should be submitted to DGTA through the PSAC, although the preference is to observe the coverage requirements of RTCA/DO-178B. Proposals are critically assessed on a case by case basis.

4.3 Use of RTCA/DO-178B as a Benchmark for Assessing Software Assurance Practices

DGTA is often required to assess software products and their associated development practices as part of finding compliance with software aspects of certification. Despite the prolific use of the RTCA/DO-178B in some sectors, there are numerous software development agencies with which the ADF is engaged, that do not use RTCA/DO-178B in the development of their software. In these cases, how does DGTA assess the acceptability of the software? Furthermore, it can often be confused that DGTA is attempting to apply RTCA/DO-178B to programs where it has not been contracted. This section explains how RTCA/DO-178B is used as a benchmark.

As discussed in Section 2.3 and presented in Annex A, RTCA/DO-178B objectives can be broadly classified under the following software safety criteria: requirements validity, requirements satisfaction, and requirements traceability. These criteria are fundamental to software safety irrespective of the assurance framework imposed or used. Therefore, on the basis that RTCA/DO-178B objectives broadly (with the previously noted exceptions) address these criteria, and provide objectives that elicit the means of addressing these criteria to a greater fidelity, it is possible to use the objectives of RTCA/DO-178B as measures of the fundamental principles of software safety assurance. In particular, RTCA/DO-178B provides clear measures of the adequacy of:

- The fidelity in the specification and traceability of requirements such that there is no significant gap between the required behaviours (as documented requirements specifications and design descriptions) and the executable code (derived from source code) – necessary to understand that all behaviours of the software are appropriate with respect to safety.
- The extent of test based verification of requirements and implementation on the target computer in the intended environment (normal, robustness, coverage) – to determine that the requirements are adequately verified in the target environment.
- Development and review rigour, including requirements for independent review and oversight – to assure that the evidence presented contains an acceptably small number of errors and faults.

4.4 COTS with RTCA/DO-178B

While COTS software should fall within the scope of previously developed software within RTCA/DO-178B, rarely is the necessary evidence available to satisfy the

criteria prescribed by RTCA/DO-178B for previously developed software. Certainly the RTCA/DO-178B criteria for previously developed criteria are strict, but this is not because they are excessive. It is merely that the set benchmarks for previously developed software to have come from a rigorous software development process. However, this assumption is not always compatible with all conceived uses of COTS in safety systems. For these reasons, DGTA may accept alternative approaches to the use of COTS.

For example, DGTA may accept proposals to use COTS within a RTCA/DO-178B development on the basis that the one or more of the following types of properties can be demonstrated (context specific):

- *Partitioning* - partitioning integrity (containment and mediation) can be demonstrated between COTS components and software that provides behaviours necessary for the safety of the system.
- *Non-Interference* - the behaviours of the COTS components are known to a sufficient fidelity and can be shown to not interfere with the behaviours required for the safe operation of the system.
- *Acceptable Behaviours* - the behaviours of the COTS components are known to a sufficient fidelity and are demonstrated to be acceptable with respect to the safety of the system.

Clearly the approach adopted is determined by the intended use of the COTS components, however it also affects the software architectures and other design aspects of the system, or the level of observability (and required analysis of behaviours) of the COTS components. These factors may affect the appropriateness of one COTS component over another, depending on the application. The PSAC provides the means for proposing such approaches to DGTA for assessment on a case by case basis.

4.5 Migrating to RTCA/DO-178B

Many legacy ADF aircraft have been acquired where no software assurance standard (such as RTCA/DO-178B) has been explicitly applied. Instead, software development and lifecycle standards such as DOD-STD-2167A and MIL-STD-498 were inherently relied upon in the original development. Development and lifecycle standards like MIL-STD-498, in isolation do not provide an adequate basis for providing software assurance. Therefore, these standards must be supplemented to provide ongoing confidence in the product commensurate with accepted practices as it is maintained in-service. [AAP7001.054].

For some organisations there may be a commercial advantage in transitioning their practices to RTCA/DO-178B; either because they are seeking to unify processes between military and commercial aspects of their businesses, or because they are motivated to improve their practices inline with international conventions, and also for commercial reasons. Where these commercial imperatives don't exist, then DGTA provides an alternate means of supplementing software assurance activities –

namely the development of a Software Assurance Task Matrix, which is a consultative process with DGTA and shall not be described further in this paper. For further information refer to [AAP7001.054].

The FAA provides guidelines in Notice 8110.49 Chapter 10 on the application of RTCA/DO-178B for changes to legacy systems. The guidelines are intended for application to systems developed to RTCA/DO-178 and RTCA/DO-178A. However, DGTA recognises that a similar approach can be applied to legacy military systems migrating to RTCA/DO-178B. While careful management of the scope of retrospective production of software assurance evidence is required to make this approach economically feasible for small changes, and extensive liaison with DGTA is required to manage expectations of compliance with RTCA/DO-178B objectives, DGTA has assessed the approach as acceptable.

5 Summary

This paper has argued the sufficiency of RTCA/DO-178B when properly applied within its intended context, including additional guidance included in [AAP7001.054]. This paper has also rebutted onerousness and excessiveness arguments often levied at RTCA/DO-178B in ADF programs. In these cases, this paper has argued that the onerousness and excessiveness claims are mostly unfounded in that they are not based on a defensible engineering argument, but instead associated factors, rather than those most relevant to the safety of the system. While the cost effectiveness of the development and acquisition of airborne software is certainly of concern, it is DGTA's belief that RTCA/DO-178B provides part a cost effective framework for developing safety critical and safety related software, while providing sufficient flexibility to deal to alternative approaches.

6 Acknowledgments

I would like to thank Systems Certification and Integrity (SCI) – DGTA staff, particularly Mark Wade, for their input to and review of this paper.

7 References

The following documents, papers and publications are referenced throughout this paper. A number of these documents are not available in the public domain for propriety or confidentiality reasons. Readers wishing to seek further information should direct their queries to the author of this paper, or the relevant standards body.

[8110.49] Federal Aviation Administration, "Order 8110.49 – Software Approval Guidelines", United States Department of Transportation, 03 June 2003.

[AAP7001.54] Australian Defence Force, "Australian Air Publication AAP 7001.054 Airworthiness Design Requirements Manual", June 2006.

[ARP4754] SAE International, "Aerospace Recommended Practice 4754 – Certification Considerations for Highly Integrated or Complex Aircraft Systems", November 1996.

- [ARP4761] SAE International, "Aerospace Recommended Practice 4761 – Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment", December 1996.
- [Bar03] J. Barnes, "High Integrity Software: The SPARK Approach to Safety and Security", Great Britain: Addison-Wesley, 2003.
- [CAST5] Certification Authorities Software Team Position Paper Cast 5 – Guidelines for Proposing Alternate Means of Compliance to RTCA/DO-178B, Federal Aviation Authority, 2000
- [DO178B] RTCA Inc., "RTCA/DO-178B: Software Considerations in Airborne Systems and Equipment Certification", Washington D.C.: RTCA Inc., 1992.
- [DO248B] RTCA Inc., "RTCA/DO-248B: Final Report for Clarification of DO-178B – Software Considerations in Airborne Systems and Equipment Certification", Washington D.C.:RTCA Inc., 2001.
- [DuL00] A. Dupay, N. Leveson, "An Empirical Evaluation of the MC/DC Coverage Criterion on the HETE-2 Satellite Software", Proceedings of the Digital Aviation Systems Conference (DASC), Philadelphia, USA, October 2000.
- [HaV01] K. Hayhurst, D. Veerhusen, "A Practical Approach To Modified Condition/Decision Coverage", 20th Digital Avionics Systems Conference (DASC), Daytona Beach, Florida, USA, October 14-18, 2001.
- [IEEE12207] The Institute of Electrical and Electronics Engineers, Inc. "IEEE Std 12207-1996 Standard for Information Technology - Software life cycle processes, IEEE Standard Board, Mar 1998.
- [IEEE1228] The Institute of Electrical and Electronics Engineers, Inc. "IEEE Std 1228-1994 (R2002) IEEE Standard for Software Safety Plans", IEEE Standards Board, 17 Mar 1994.
- [JSSC99] Joint Software System Safety Committee, "Software System Safety Handbook", Joint Services Computer Resources Management Group: U.S. Navy, U.S. Army, and the U.S. Air Force, Joint Services Software Safety Committee, Joint Services System Safety Panel, Electronic Industries Association, G-48 Committee, December 1999.
- [JTM07] D. Jackson, M. Thomas, L. Millet, Editors, "Software for Dependable Systems: Sufficient Evidence?", Committee of Certifiably Dependable Software Systems, National Research Council, National Academy of Sciences, USA, 2007.
- [KaB03] K. Kapoor, J Bowen, "Experimental Evaluation of the Variation in Effectiveness for DC, FPC and MC/DC Test Criteria", Centre for Applied Formal Methods, South Bank University, London, UK, 2003.
- [KnG07] J.C. Knight, P.J. Graydon, "Engineering, Communication, And Safety", Department of Computer Science, University of Virginia, August 2007.
- [Lev95] N. Leveson, "Safeware: System Safety and Computers", Reading, Mass.: Addison Wesley, 1995.
- [Mar08] P. Marks, "Flight of the Software Bugs", in New Scientist, pp26-26, 9 Feb 2008.
- [McD01] J.A. McDermid, "Software Safety: Where's the Evidence?", Department of Computer Science, University of York, 2001.
- [McK06] J. McDermid, T. Kelly, "Software in Safety Critical Systems: Achievement and Prediction", Nuclear Future, Volume 03, No. 03, 2006.
- [MILSTD498] United States of America, Department of Defense, "MIL-STD-498 Software Development and Documentation", 05 December 1994.
- [MILSTD882C] United States of America, Department of Defense, "MIL-STD-882C System Safety Program Requirements", 19 Jan 1993.
- [NTS06] National Transportation Safety Board, "Safety Report on the Treatment of Safety-Critical Systems in Transport Airplanes", Safety Report NTSB/SR-06/02, Washington, D.C., USA, 2006.
- [Pum99] D.J. Pumfrey, "The Principled Design of Computer System Safety Analyses", Department of Computer Science, University of York, 1999.
- [SaL06] C. Salmon, and C. Lee, "The Certification of Software containing software developed using RTCA DO-178B", Avionics Systems Standardisation Committee, Ref ASSC /12/0013 Issue 3, ERA Report 2006-0036 Issue 3, ERA Project 7D0134809, ERA Technology Ltd, UK, Jun 2006.
- [ViB02] S. A. Vilkomir, J. P. Bowen, "From MC/DC to RC/DC: Formalization and Analysis of Control-Flow Testing Criteria", South Bank University, CISM, UK, 2002.
- [Wea03] R.A. Weaver, "The Safety of Software – Constructing and Assuring Arguments", Department of Computer Science, University of York, 2003.

ANNEX A

**Comparison of RTCA/DO-178B Objectives against
Requirements Validity, Requirements Satisfaction and Requirements Traceability Criteria.**

| RTCA/DO-178B Objective | | Requirements Validity | Requirements Satisfaction | Requirements Traceability |
|---|---|----------------------------------|--------------------------------------|--------------------------------------|
| Software Planning Process Table A-1 | | | | |
| 1-1 | Software development and integral processes are defined. | | | |
| 1-2 | Transition criteria, interrelationships and sequencing among processes are defined. | | | |
| 1-3 | Software life cycle environment is defined. | | | |
| 1-4 | Additional considerations are addressed. | | | |
| 1-5 | Software development standards are defined. | | | |
| 1-6 | Software plans comply with this document. | | | |
| 1-7 | Software plans are coordinated. | | | |
| Software Development Processes Table A-2 | | | | |
| 2-1 | High-level requirements are developed. | ✓ | | |
| 2-2 | Derived high-level requirements are defined. | ✓ | | |
| 2-3 | Software architecture is developed. | ✓* | ✓* | |
| 2-4 | Low-level requirements are developed. | ✓ | | |
| 2-5 | Derived low-level requirements are defined. | ✓ | | |
| 2-6 | Source Code is developed. | | ✓ | |
| 2-7 | Executable Object Code is produced and integrated in the target computer. | | ✓ | |
| Verification of Outputs of Software Requirements Process Table A-3 | | | | |
| 3-1 | Software high-level requirements comply with system requirements. | ✓ | | |
| 3-2 | High-level requirements are accurate and consistent. | ✓ | | |
| 3-3 | High-level requirements are compatible with target computer. | ✓ | | |
| 3-4 | High-level requirements are verifiable. | | ✓ | |
| 3-5 | High-level requirements conform to standards. | ✓ | | |
| 3-6 | High-level requirements are traceable to system requirements. | | | ✓ |
| 3-7 | Algorithms are accurate. | ✓ | | |

| RTCA/DO-178B Objective | | Requirements Validity | Requirements Satisfaction | Requirements Traceability |
|---|---|-----------------------|---------------------------|---------------------------|
| Verification of Outputs of Software Design Process Table A-4 | | | | |
| 4-1 | Low-level requirements comply with high-level requirements. | ✓ | | |
| 4-2 | Low-level requirements are accurate and consistent | ✓ | | |
| 4-3 | Low-level requirements are compatible with target computer. | ✓ | | |
| 4-4 | Low-level requirements are verifiable. | | ✓ | |
| 4-5 | Low-level requirements conform to standards. | ✓ | | |
| 4-6 | Low-level requirements are traceable to high-level requirements. | | | ✓ |
| 4-7 | Algorithms are accurate. | ✓ | | |
| 4-8 | Software architecture is compatible with high-level requirements. | ✓ | | |
| 4-9 | Software architecture is consistent. | ✓ | | |
| 4-10 | Software architecture is compatible with target computer. | ✓ | | |
| 4-11 | Software architecture is verifiable. | | ✓ | |
| 4-12 | Software architecture conforms to standards. | ✓ | | |
| 4-13 | Software partitioning integrity is confirmed. | | ✓ | |
| Verification of Outputs of Software Coding Process Table A-5 | | | | |
| 5-1 | Source Code complies with low-level requirements. | | ✓ | |
| 5-2 | Source Code complies with software architecture. | | ✓ | |
| 5-3 | Source Code is verifiable. | | ✓ | |
| 5-4 | Source Code conforms to standards. | | ✓ | |
| 5-5 | Source Code is traceable to low-level requirements. | | | ✓ |
| 5-6 | Source Code is accurate and consistent. | | ✓ | |
| 5-7 | Output of software integration process is complete and correct. | | ✓ | |
| Testing of Outputs of Integration Process Table A-6 | | | | |
| 6-1 | Executable Object Code complies with high-level requirements. | | ✓ | |
| 6-2 | Executable Object Code is robust with high-level requirements. | | ✓ | |
| 6-3 | Executable Object Code complies with low-level requirements. | | ✓ | |
| 6-4 | Executable Object Code is robust with low-level requirements. | | ✓ | |
| 6-5 | Executable Object Code is compatible with the target computer. | | ✓ | |
| | Executable Object Code is traceable to Source Code (implicit objective) | | | ✓# |

| RTCA/DO-178B Objective | | Requirements Validity | Requirements Satisfaction | Requirements Traceability |
|---|---|--|---------------------------|---------------------------|
| Verification of Verification Process Results Table A-7 | | | | |
| 7-1 | Test procedures are correct. | | ✓ | |
| 7-2 | Test results are correct and discrepancies explained. | | ✓ | |
| 7-3 | Test coverage of high-level requirements is achieved. | | ✓ | |
| 7-4 | Test coverage of low-level requirements is achieved. | | ✓ | |
| 7-5 | Test coverage of software structure (modified condition decision coverage) is achieved. | ✓# | ✓ | |
| 7-6 | Test coverage of software structure (decision coverage) is achieved. | ✓# | ✓ | |
| 7-7 | Test coverage of software structure (statement coverage) is achieved. | ✓# | ✓ | |
| 7-8 | Test coverage of software structure (data coupling and control coupling) is achieved. | ✓# | ✓ | |
| | Inadequacies in software requirements are identified (implicit objective) | ✓# | | |
| | Dead code does not exist (implicit objective) | | ✓# | |
| | Deactivated code cannot be inadvertently executed (implicit objective) | | ✓# | |
| Software Configuration Management Process Table A-8 | | | | |
| 8-1 | Configuration items are identified. | Establishes that the evidence presented is consistent with the delivered executable object code and that any deficiencies identified in satisfying any other verification related objectives are explicitly identified | | |
| 8-2 | Baselines and traceability are established. | | | |
| 8-3 | Problem reporting, change control, change review and configuration status accounting are established. | | | |
| 8-4 | Archive, retrieval and release are established. | | | |
| 8-5 | Software load control is established. | | | |
| 8-6 | Software lifecycle environment control is established. | | | |
| Software Quality Assurance Process Table A-9 | | | | |
| 9-1 | Assurance is obtained that software development and integral processes comply with approved software plans and standards. | Establishes that the evidence for demonstrating requirements validity, satisfaction and traceability was derived from processes agreed with the certification authority | | |
| 9-2 | Assurance is obtained that transition criteria for the software lifecycle processes are satisfied. | | | |
| 9-3 | Software conformity review is conducted. | | | |
| Certification Liaison Process Table A-10 | | | | |
| 10-1 | Communication and understanding between the applicant and certification authority is established. | Establishes unambiguous consensus between the certification authority and applicant regarding provision of evidence for demonstrating requirements validity, satisfaction and traceability | | |
| 10-2 | The means of compliance is proposed and agreement with the Plan for Software Aspects of Certification is obtained. | | | |
| 10-3 | Compliance substantiation is provided. | | | |

| RTCA/DO-178B Objective | Requirements Validity | Requirements Satisfaction | Requirements Traceability |
|---|--------------------------|------------------------------|------------------------------|
| Additional Comments | | | |
| <p>* The development of the software architecture should address both implementation aspects of satisfying the software high level requirements, as well as developing software low level requirements relating to detailed design. On this basis it addresses both validity and satisfaction aspects.</p> <p># Refer to Sections 2.3.1 and 3.1.3 of this paper for further discussion.</p> | | | |