Defence Research and Development Canada
Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

Getting smarter at managing avionic software

The results of a two-day requirements elicitation workshop with DTAES

P. Charland
D. Dessureault
G. Dussault
M. Lizotte
F. Michaud
D. Ouellet
M. Salois
DRDC Valcartier

Defence R&D Canada – Valcartier
External Client Report
DRDC Valcartier ECR 2007-097
October 2007

Canada

Getting smarter at managing avionic software

The results of a two-day requirements elicitation workshop with DTAES

P. Charland
D. Dessureault
G. Dussault
M. Lizotte
F. Michaud
D. Ouellet
M. Salois

Defence R & D Canada – Valcartier

External Client Report
DRDC Valcartier ECR 2007-097
October 2007

This page intentionally left blank.

Acknowledgements

The authors would like to thank the 11 participants for their enthusiastic involvement in the workshop. Special thanks are also addressed to Jennifer and Sylvie for helping us in setting up the workshop and running the two days with an iron hand in a velvet glove.

Abstract

Managing avionic software effectively is a challenge with today's tools and resources. The Directorate of Technical Airworthiness and Engineering Support (DTAES) has recognized this and has approached DRDC Valcartier to address the problem from a technological standpoint. The team at DRDC Valcartier has considerable expertise in software engineering but, unfortunately at this point, very little in avionics. To offset this, a series of measures have been undertaken to ramp up this expertise (e.g. training, production of state-of-the-art reports).

The first of these measures was the organization of a two-day workshop with DTAES and its partners to better define their requirements in avionic software management. This document highlights the results of this workshop, held in December 2006. The workshop used the Decision Support Services (DSS) collaborative laboratory, located at the National Defence Headquarters in Ottawa. This laboratory is built on the MeetingWorks toolset to provide each of the 11 participants with his own computer on which to give feedback on the four pre-identified domains (extraction, analysis, visualization, process support). From the outputs of these domains, the main tasks or problematic areas were identified and prioritized. These will be further investigated later, which could lead to relevant research projects and new engineering efforts within DRDC.

Since DRDC Valcartier is a neophyte in this area, this document will not be an all-encompassing list of all avionic software engineering problems. It rather provides a summary of the most important requirements identified at the workshop. As DRDC Valcartier is currently negotiating with DTAES to improve various other aspects related to the air platforms, the document also provides an opportunity to spotlight potential openings for future collaboration to improve the whole avionic engineering process.

Résumé

De nos jours, il est très difficile de gérer efficacement les logiciels d'avionique avec les outils et les ressources disponibles. La Direction de la navigabilité aérienne et du soutien technique (DNAST) a reconnu ce problème et a approché RDDC Valcartier pour l'aborder d'un point de vue technologique. L'équipe à RDDC Valcartier possède une expertise considérable en génie logiciel, mais n'a malheureusement pas beaucoup d'expérience en avionique. Pour pallier cet état de fait, une série de mesures ont été prises pour accélérer l'acquisition de cette expertise (p. ex. formation, production de rapports sur l'état des connaissances).

La première de cette mesure consistait à organiser un atelier de deux jours avec DNAST et ses partenaires pour mieux définir leurs besoins en gestion des logiciels d'avionique. Ce document met en évidence les résultats de cet atelier, tenu en décembre 2006. L'atelier a eu lieu au laboratoire des Services d'aide à la décision, situé au Quartier général de la Défense nationale à Ottawa. Ce laboratoire est basé sur la suite d'outils MeetingWorks et fournissait un ordinateur à chacun des onze participants sur lequel ils pouvaient donner une rétroaction dans les quatre domaines pré-identifiés (extraction, analyse, visualisation, soutien au processus). À partir des résultats obtenus dans ces domaines, les tâches principales ou les régions à problèmes ont été identifiées et priorisées. Celles-ci seront étudiées plus en détail plus tard, ce qui pourrait mener à des projets de recherche pertinents et à de nouveaux efforts d'ingénierie pour RDDC.

Puisque RDDC Valcartier débute dans le domaine, ce document ne constitue pas une liste exhaustive de tous les problèmes d'ingénierie de logiciels d'avionique. Il fournit plutôt un bon résumé des plus importants besoins mentionnés à l'atelier. Comme RDDC Valcartier est présentement en négociation avec DNAST pour améliorer divers autres aspects reliés aux plateformes aériennes, ce document présente aussi une occasion d'illustrer des ouvertures potentielles de collaborations futures pour l'amélioration de l'ensemble du processus d'ingénierie avionique.

Executive Summary

The Directorate of Technical Airworthiness and Engineering Support (DTAES) has recognized that managing avionic software effectively is a challenge and has approached DRDC Valcartier to address the problem from a technological standpoint. The team at DRDC Valcartier has a vast expertise in software engineering, but it currently lacks the necessary knowledge in avionic software. To offset this, a series of measures have been undertaken to ramp up this expertise (e.g. training, production of state-of-the-art reports). The first of these measures was the organization of a two-day workshop with DTAES and its partners to better define their requirements in avionic software management.

This document highlights the results of the workshop, held in December 2006. Four domains were identified prior to the workshop to feed the brainstorming sessions. The outputs for these four domains describe the current situation, as it came out during the two days. Here is a summary:

- First domain: Extraction
  - *Development environment:* The consensus is that more automated design/code generation tools should be used but, currently, a heterogeneous environment is usual. Many compilers are used and managing makefiles is a problem. There are tailored simulators for specific avionic platforms.
  - *Documentation:* Huge paper documents are the norm. This should be changed to automate documentation generation and management as much as possible.
  - *Hardware:* Federated boxes are the norm but there is a will to move toward more open architectures. Bus 1553 and its extensions are critical but there are other busses. New processors are integrated but heat is a big factor.
  - *Human personnel:* There is a discrepancy between what is taught in universities and what is really needed. There is no formal training for software architects. Developers are either contractors or Crown employees (civil & military). Users underevaluate the complexity of the software.
  - *Models:* There is a will to use modern model-driven architecture & design but there is a long way to go. People are afraid of (unproven) new things in safety-critical systems. Real-time variants of UML are gaining momentum.
  - *Source code:* Access to source code is often a problem because of licenses and international regulations. Ada, assembly, (MISRA) C, and (JSF) C++ are most common. Free and Open Source Software (FOSS) is pointing its nose in some areas.
- Second domain: Analysis
  - *Software comprehension:* Design patterns are used with a lack of rigor and become corrupted over time. Anti-patterns are also of interest to find things that should not be there or is bad practice. Finding a good way to explore large software is an issue.
  - *Validation:* Traceability is a big issue, both forward and backward. Validation of the tools is essential but not always performed. There is a need for stricter usage rules enforcers and stricter impact analyses.
  - *Verification:* Models should be used for verification but this is not the norm yet. Schedulability analysis is critical but lacks formal support. Proving timing, concurrency, and assertions properties comes out strong in the list of priorities. Unit and coverage tests seem to be the two main testing techniques.

- Third domain: Visualization
  - *Charts, diagrams, and graphs:* There is a lack of good visualization specific to real-time and embedded systems. Dynamic structures and properties need to be seen. State diagrams are underused and many charts from UML and SysML should also be used more.
  - *User Interface:* There are many good examples of good interface design (e.g. Google and Nintendo Wii). Unfortunately, there are probably more bad examples (e.g. Rational, ARTiSAN). Most tools have big flaws and designers do not show due diligence in this area.
- Fourth domain: Process support
  - *Proportion of effort:* About 30% of the verification process is dedicated to verifying the software itself. The rest is performed mostly on 'paper' (it might be in electronic form).
  - *Target:* Both product qualification and safety certification (airworthiness) are the end goal but the importance of consistency, completeness, and agility was also stated.
  - *Requirements:* Most requirements are stated in natural text, which is a problem as it lacks rigor. DOORS is mandated by the Assistant Deputy Minister (Materiel) (ADM(Mat)). Traceability to requirements is currently very important; some feel too much so.

From these outputs, four pressing problems were extracted and prioritized. These problems provide an excellent starting point for collaborative work with other Canadian laboratories to improve the avionic engineering process for the Canadian Forces and their partners. Here is a summary:

- *Problem 1: Avionic Software Toolbox and Methodologies*
  The keyword here is probably integration. There are many good tools but most of them do not talk to each other. A research project should leverage these tools as much as possible and only develop new tools when necessary. Tools should adapt to the methodologies and not the reverse, which is often the case with current tools. There is also a need to keep on top of new technologies and train the engineers as they and the technology become available.
- *Problem 2: Capture and use requirements in a more proactive fashion*
  In theory, requirements are set in stone; in practice, they evolve and become outdated. There is a need to develop a set of tools and methodologies that would link together the software tools to provide better traceability and different levels of detail.
- *Problem 3: Technology watch and tool categorization*
  The objective is to develop a capability to provide expert advice on existing tools, to benchmark and categorize new ones, and to develop methodologies to fully tap their potential.
- *Problem 4: Quick access to current capabilities*
  There is currently no way to get a newcomer up to speed in all aspects relevant to a particular avionic project within a reasonable and practical amount of time. The problem consists in finding one.

P. Charland, D. Dessureault, G. Dussault, M. Lizotte, F. Michaud , D. Ouellet, M. Salois; 2007; Getting smarter at managing avionic software; DRDC Valcartier ECR 2007-097; Defence R & D Canada – Valcartier.

Sommaire

La Direction de la navigabilité aérienne et du soutien technique (DNAST) reconnaît que la gestion efficace des logiciels d'avionique est un défi et a approché RDDC Valcartier pour aborder ce problème d'un point de vue technologique. L'équipe à RDDC Valcartier possède une vaste expertise en génie logiciel, mais souffre d'un manque de connaissances en logiciels d'avionique. Pour pallier ce problème, une série de mesures ont été prises pour accélérer l'acquisition de cette expertise (p. ex. formation, production de rapports sur l'état des connaissances). La première de ces mesures consistait à organiser un atelier de deux jours avec DNAST et ses partenaires pour mieux définir leurs besoins et gestion des logiciels d'avionique.

Ce document présente les résultats de cet atelier, tenu en décembre 2006. Quatre domaines ont été préalablement identifiés avant l'atelier pour lancer les sessions de remue-méninges. Les résultats de ces quatre domaines décrivent la situation courante, telle que discutée pendant les deux jours. En voici un résumé :

- Premier domaine : Extraction
  - *Environnement de développement :* Le consensus est que des outils plus automatisés de conception/génération de code devraient être utilisés, mais que présentement un environnement hétérogène est usuel. Plusieurs compilateurs sont utilisés et la gestion des fichiers de configuration ('makefile') est un problème. Il existe des simulateurs sur mesure pour des plateformes avioniques données.
  - *Documentation :* Les énormes documents papier sont communs. Ceci devrait changer pour automatiser autant que faire se peut la génération de la documentation et sa gestion.
  - *Matériel :* Des boîtes fédérées constituent la norme, mais il y a une volonté d'aller vers des architectures plus ouvertes. Le bus 1553 et ses extensions sont critiques mais d'autres bus sont utilisés. De nouveaux processeurs sont intégrés, mais la chaleur est un facteur important.
  - *Personnel humain :* Il y a une incompatibilité entre ce qui est enseigné dans les universités et ce qui est vraiment requis. Il n'y a pas de formation formelle pour les architectes logiciels. Les développeurs sont, soit des contracteurs, soit des employés de la couronne (civils et militaires). Les usagers sous-estiment la complexité du logiciel.
  - *Modèles :* Il y a une volonté d'utiliser les architectures et la conception guidés par modèles, mais il y a beaucoup de chemin à faire. Les gens ont peur des nouvelles choses (non prouvées) dans les systèmes où la sûreté est critique. Les variantes temps-réel d'UML gagnent en popularité.
  - *Code source :* L'accès au code source est souvent un problème en raison des licences et des règlements internationaux. Ada, l'assembleur, (MISRA) C et (JSF) C++ sont les plus communs. Les logiciels libres pointent leur nez dans quelques domaines.
- Deuxième domaine : Analyse
  - *Compréhension du logiciel :* Les patrons de conception sont utilisés avec un manque de rigueur et deviennent corrompus avec le temps. Les anti-patrons sont également intéressants pour trouver des choses qui ne devraient pas être là ou qui constituent une mauvaise pratique. Trouver une bonne façon d'explorer un gros programme est un problème.
  - *Validation :* La traçabilité est un gros enjeu, tant en amont qu'en aval. La validation des outils est essentielle, mais n'est pas toujours effectuée. Il y a un besoin d'imposer des règles d'usage de programmation et des analyses d'impact plus sévères.
  - *Vérification :* Des modèles devraient être utilisés pour la vérification, mais ceci n'est pas encore standard. Les analyses d'ordonnancement sont critiques, mais souffrent d'un manque de soutien formel. Prouver les propriétés de synchronisation, de concurrence et d'assertions ressort fortement dans la liste des priorités. Les tests unitaires et de couverture semblent être les deux techniques principales de test.

- Troisième domaine : Visualisation
  - *Diagrammes et graphes :* Il y a un manque de bonnes visualisations propres aux systèmes embarqués et en temps réel. Les structures et les propriétés dynamiques devraient être visualisées. Les diagrammes d'état sont sous-utilisés et plusieurs des diagrammes d'UML et de SysML devraient être utilisés davantage.
  - *Interface usagé :* Il existe plusieurs bons exemples de conception d'interface (p. ex. Google et Nintendo Wii). Malheureusement, il existe probablement plus de mauvais exemples (p. ex. Rational, ARTiSAN). La plupart des outils ont des défauts apparents et les concepteurs ne font pas preuve de diligence raisonnable dans ce domaine.
- Quatrième domaine : Soutien au processus
  - *Proportion des efforts :* Environ 30 % du processus de vérification est consacré à la vérification du logiciel lui-même. Le reste est effectué en grande partie sur 'papier' (peut-être en format électronique).
  - *Cible :* La qualification des produits et la certification de sûreté (navigabilité aérienne) sont les buts finaux mais l'importance de la consistance, de la complétude et de l'agilité a aussi été mentionnée.
  - *Besoins :* La plupart des besoins sont exprimés en langage naturel, ce qui est un problème à cause du manque de rigueur. DOORS est mandaté par le sous-ministre adjoint (Matériels). La traçabilité des besoins est présentement très importante ; certains pensent même trop importante.

À partir de ces résultats, quatre problèmes immédiats ont été extraits et priorisés. Ces problèmes fournissent un excellent point de départ pour des travaux collaboratifs avec d'autres laboratoires canadiens pour améliorer le processus d'ingénierie de l'avionique des Forces canadiennes et de leurs partenaires. En voici un résumé :

- *Problème 1 : Boîte d'outils et méthodologies pour les logiciels d'avionique*
  Le mot clé ici est probablement intégration. Il existe plusieurs bons outils, mais la plupart ne se parlent pas entre eux. Un projet de recherche devrait miser sur ces outils autant que possible et développer seulement de nouveaux outils si nécessaire. Les outils devraient s'adapter aux méthodologies et non l'inverse, ce qui est souvent le cas des outils existants. Il y a aussi un besoin de se tenir à jour avec les nouvelles technologies et de former les ingénieurs à mesure que les technologies et les ingénieurs eux-mêmes deviennent disponibles.
- *Problème 2 : Saisir et utiliser les besoins de façons plus proactive*
  En théorie, les besoins sont coulés dans le béton ; en pratique, ils évoluent et deviennent désuets. Il y a un besoin de développer un ensemble d'outils et de méthodologies qui relieraient ensemble les outils logiciels pour fournir une meilleure traçabilité et différent niveaux de détail.
- *Problème 3 : Veille technologique et catégorisation d'outils*
  L'objectif est de développer une capacité de services conseils experts sur les outils existants, de tester et catégoriser les nouveaux outils et de développer une méthodologie pour profiter pleinement de leur potentiel.
- *Problème 4 : Accès rapide aux capacités actuelles*
  Il n'y a présentement aucune façon pour un nouvel arrivant de se mettre à jour dans tous les aspects d'un projet avionique particulier dans un délai raisonnable et pratique. Le problème consiste à en trouver une.

P. Charland, D. Dessureault, G. Dussault, M. Lizotte, F. Michaud , D. Ouellet, M. Salois; 2007; Pour une gestion plus intelligente des logiciels avioniques; DRDC Valcartier ECR 2007-097; R & D pour la défense Canada–Valcartier.

Table of Contents

1 Introduction

Managing a modern air platform (e.g. a plane) is a challenge with today's tools and techniques. Software engineering is, in general, an immature domain. Precise recipes that work every time have not been developed yet, as may be the case in other engineering domains. The situation is even more complex when avionic software is present as more stringent requirements are needed to ensure safety and security. Recognizing this problem, the Directorate of Technical Airworthiness and Engineering Support (DTAES) approached Defence Research & Development Canada (DRDC) – Valcartier to address this problem from a technological standpoint. That is to say: to survey and evaluate existing tools; to provide advice on existing techniques and methodologies; to develop missing pieces; and to integrate all of the above.

Unfortunately, DRDC Valcartier has a vast knowledge of software engineering in general but very little knowledge of avionic software at this point. In order to fill this knowledge gap, a series of measures were initiated to ramp up the expertise. For example, the production of surveys specific to the real time/embedded avionic software world are in progress (e.g. programming languages, visualization techniques) and training has already started with a four-day UCLA class on digital avionics system that was hosted by DRDC Valcartier.

The first step, however, was better define DTAES' requirements in avionic software management. To this end, a two-day workshop was held in December 2006 at the National Defence Headquarters, in Ottawa. The goal was to better define these requirements with DTAES and its partners and to start learning about the whole avionic software process. It used the Decision Support Services (DSS) collaborative laboratory and the MeetingWorks tool to provide each of the 11 participants with his own computer on which to give feedback. Annex A lists the participants.

Four pre-identified software engineering domains were seeded in the collaborative tool prior to the workshop to organize the brainstorming sessions and get them going. The focus was originally given to validation and verification, the core expertise from the team at DRDC Valcartier, but the workshop ended up with a much wider scope. These four domains are:

Extraction Identify the sources of raw information (e.g. programming language, documentation, tools, personnel).

Analysis Identify the different kinds of analysis (e.g. (anti) design patterns, impact analysis, schedulability, feature location).

Visualization Identify the different kinds of graphs and graphical representations that are commonly used (e.g. charts, diagrams, user interface).

Process support Identify the software engineering artifacts required to perform avionic software verification (e.g. proportion of effort, target, requirements).

The summary of what came out from these four brainstorming sessions is presented in Chapters 3 to 6. Following these four sessions, the fifth and last session consisted in identifying the main tasks or problematic areas in managing avionic software. Four problems came out of the workshop, were prioritized, and are discussed in Chapter 7.

As stated above, DRDC is quite new to avionic software. As such, this document is not an all-encompassing list of all avionic software engineering problems. The reader may feel that obvious matters are missing, but it is only a summary of the problems that were illustrated during the workshop. As DRDC is currently tapped to develop into a long-term resource for avionic software expertise, it is also hoped that this document can serve as a springboard toward collaborative work not only with DTAES but also with other partners.

2 How to Read and Navigate this Report

This document is designed to be read at different levels, from a general overview to more technical details. Entries for the four domains are within a table arranged like this:

Title	The entry itself, sometimes with subtitles
Idea behind this entry	*What's missing?* In describing the situation *as is*, a summary of missing pieces that came out during the workshop.
Summary if the entry is long enough	

This way, a reader interested in the overview can read only the left column when a summary is provided and read the right column only when more details are required. Using the navigational bookmarks provided by the Portable Document Format (PDF) and Acrobat (Reader), it is easy to obtain a quick overview. If further information is required, hyperlinks may be followed to specific references, either on the accompanying CD-ROM or on the Web. Links are color-coded. A blue link points within the report, a cyan link points to a file on the CD-ROM, and a magenta link points to a web page, indicating that access to the Internet is required. To get back from a followed link, use the 'Go to previous view' arrow on the toolbar/menu or ALT-Left arrow on the keyboard. The left/right arrows can be used to go to the previous/next page.

3 First Domain: Extraction

The idea behind the first exercise was to identify the sources of raw information that are available when tackling avionic software problems. Another goal was to identify missing data that could be requested from contractors.

As this was the first exercise, many participants started thinking about other domains at this step. Thus, many of the items were transferred to other categories (mainly the Second Domain: Analysis). Also, the original idea was to focus on existing systems but many participants expressed needs they wished fulfilled. These will be addressed later as part of propositions for future research projects.

Development environment

What types of environment (tools/techniques) are available?

The consensus is that more automated design/code generation tools should be used but, currently, a heterogeneous environment is usual. Many compilers are used and managing makefiles is a problem. There are tailored simulators for specific avionic platforms.

Model-based design: Some contractors already use model-based design and automated code generation, but this does not seem to be the norm. There is a wish to use more formal modeling techniques, such as the Unified Modeling Language (UML) [1] and its derivatives (e.g. RT UML), both internally and as a requirement for contractors but this is not yet the norm either. Such a tool would need to pass airworthiness qualification requirements and adequate training would be required to make sure that everyone uses the tool correctly.

Compilers: A subset of the popular compilers are generally used to ensure a deterministic and safe behavior. Compilers that enforce usage rules, such as Motor Industry Software Reliability Association (MISRA) C [2] or JSF C++ [3], are available but their use is not yet the norm. A certifying compiler[4] that provides direct traceability between source code and machine code would be really useful. Examples of compilers that are currently used include: GCC, Green Hills Ada95, WindRiver's Tornado II and Workbench.

Makefiles: On the CF-18 side, a team of 2 to 3 people are working on the makefile and making sure that the development environment works. RMC does not teach the arcane art of makefiles and neither do most of the other universities (to the authors knowledge). This raises the question: Should they teach this?

Simulator: There are many software/hardware simulators. The challenge is to keep them up to date with the real systems and to make sure that the simulations are realistic and that real-time constraints are maintained. This is usually achieved with hardware-in-the-loop types of simulations, but it becomes quite difficult to demonstrate that the error-handling software will work exactly the same way on the real system.

Development environment (continued)

What's missing? A development environment should make it possible to assess the operational requirements through scenario-based modeling and simulation. This would help in defining the functional requirements and then start its allocation.

Documentation

What types of documentation are available?

Huge paper documents are the norm. This should be changed to automate documentation generation and management as much as possible.

Relevance: One of the key questions is what is the relevance of traditional documentation. Everyone knows that, most of the time, it is out of date. Some even think that the static nature of writing documentation makes it irrelevant by definition when contrasting it with the dynamic nature of the design and the source code. Ideally, documentation generation and management should be as automated as possible from the design/source code. In such a scenario, better test cases would also be generated automatically and would keep their relevance throughout software versions; it would also make it easy to see what has changed between versions.

Specifications: Expressing the requirements for a system is currently usually done textually. The use of natural language is inefficient at best and often ambiguous. There is a need for more formal specifications that can be used to better trace a requirement from design to code and back. Such formal specifications are currently only used in high-criticality systems.

Quality and completeness: Documentation on paper is practically useless; at the very least an electronic and fully indexed version should be made available. The quantity of information is also staggering in most cases. Visualization is key to good documentation and good navigation through it (low word count/high diagram count.)

What's missing? A whole new way of managing documentation that is semi-automatically generated from the model or the source code in order to stay up to date.

Hardware

What types of hardware are in use?

Federated boxes are the norm but there is a will to move toward more open architectures. Bus 1553 and its extensions are critical but there are other busses. New processors are integrated but heat is a big factor.

Types of boxes: Federated boxes are still the norm in legacy systems. There is a wish to move toward a modular open system architecture. The hardware architecture must support multiple levels of security. Proprietary boxes exist but there is a tendency to stay away from them as much as possible and use a more open architecture instead.

Busses: Many types of busses are used but the most common one seems to be 1553, especially with the new notices to the standard that are coming out. Dedicated lines are used in some systems, especially older ones. A wave of faster busses that will go up to 800 Mbps in the next few years are also coming, including well-known information technology standards such as Ethernet IP.

Processors: Processors are becoming faster all the time, so new platforms should be designed for change to adapt to them. However, heat is always a concern in embedded systems, so there is a limit to what can be used. A partial list of currently used processors include: Intel 8086 and 8080, AYK-14. PowerPC is coming for new projects.

What's missing? There is a need to develop better, more realistic, and more omnipresent black box simulations. The useful life of new hardware is becoming shorter. It is therefore critical to change the way we manage it.

Human personnel

What human resources are available to work on the projects or as subject matter experts?

There is a discrepancy between what is taught in universities and what is really needed. There is no formal training for software architects. Developers are either contractors or Crown employees (civil & military). Users underevaluate the complexity of the software.

Educators: As is often the case with universities, there is a wide gap between what they teach and what the industry needs. The industry should encourage their engineers to participate in the composition of classes, especially at the graduate level.

Architects: System, software, and hardware so-called architects need to better communicate in order to ensure that the configuration of an aircraft is known and planned for. A participant noted that there is no formal education to become a software architect. The title can only be earned after years in the trenches. Universities mainly produce coders (undergraduate).

Developers: There are many run-of-the-mills coder but only a precious few gurus. These truly understand the system as a whole and are often better than the architect at explaining it. However, by their usually introspective nature, they normally do not freely participate in reviews and meetings. Developers are a mix of contractors, public service employees, and military personnel. There is a general lack of understanding of DND's airworthiness requirements.

Human personnel (continued)

Users: Pilots, flight engineers, and maintainers are the main users of avionic software. One generic problem is that users tend to think that software is easy. They do not understand the complexity behind it and, thus, why it costs so much and takes so long to develop.

What's missing? There is a need to adapt and adopt modernized processes to better address the problems of avionic software. This is often a problem of people trying to use tried and tested methods to solve new problems for which these old methods are no longer adequate.

Models

What types of (design) models are used across avionic processes?

There is a will to use modern model-driven architecture & design, but there is a long way to go. People are afraid of (unproven) new things in safety-critical systems. Real-time variants of UML are gaining momentum.

Model-driven architecture & design: There is a strong push to move toward a more modern approach where the design of the hardware and the software is done in common. The model can then be used to generate the code automatically. It could also mean that the model itself becomes executable and can be used to run simulations and get user feedback. Some people are afraid of automated code generation as the process as yet to be certified. There is an AERAC-funded project at RMC to study the impact of such an approach on avionic software [5].

Languages: Real-time meta-UML models variations are slowly rising. SysML, one of these UML derivatives, receives a good interest from the industry but is not yet in common use. AADL [6] is another formal language that is being pushed by a portion of the industry.

What's missing? There is a need to use more models in the whole process. It is difficult to make the clients aware of the benefits; hence they do not request enough information from contractors. There is work to be done there, especially on standardization issues.

Source code

What are the different languages that are in use for avionic software?

Access to source code is often a problem because of licenses and international regulations. Ada, assembly, (MISRA) C, and (JSF) C++ are most common. FOSS is pointing its nose in some areas.

Access: In many cases, the Canadian Forces do not have access to the source code as most of the systems are American and fall under International Traffic in Arms Regulation (ITAR). Companies are also rather shy in this area as they are concerned with intellectual property. FOSS [7] is starting to appear in some areas, especially for UAVs.

Languages: Ada (83 & 95) is still in use but is not taught in most places anymore. Assembly is present in many critical systems (e.g. CF-18). C seems to be disappearing to be replaced by C++; this is cause for legitimate concerns on safety aspects as more modern and safe languages are available. Universities are coming back to teaching some C/C++ as they recognize its staying power, particularly for real time and embedded systems. Java is virtually absent from avionic software, but there are some proponents.

Usage rules: MISRA C [2] and JSF C++ [3] seem to be the prevalent subsets for avionic software, with a hint of the seemingly defunct Embedded C++ [8] (i.e. it has not been updated in five years).

What's missing? Because of ITAR and other proprietary solutions, libraries and documentation are often not shared by the contractors. The Government of Canada needs to work on this and maybe adopt a tougher line of conducts with the contractors. There is also a need to better manage the different standards and address their impact.

4 Second Domain: Analysis

The idea behind the second exercise was to identify the different kinds of analysis that need to be performed when tackling avionic software problems. A second objective was to identify useful metrics and patterns.

Software comprehension

What steps are taken to understand the software before reuse/update/upgrade?

Design patterns are used with a lack of rigor and become corrupted over time. Anti-patterns are also of interest to find things that should not be there or is bad practice. Finding a good way to explore large software is an issue.

Design patterns: They are used in principle in the development of avionic software. However, they often erode along the way as the software is maintained. In such a case, an identified pattern may not be the one that was originally intended. This is noteworthy as it may indicate other corruptions from the original design. Finding an anti-pattern, something that should not exist or is considered bad practice, is also noteworthy.

Software reconnaissance: Some tools exist to explore software [9] but they are currently limited. To be useful, such a tool should be as automated as possible and leave only low level abstractions for the human operator to describe and understand.

Validation

What types of validation need to be performed on source code and requirements?

Traceability is a big issue, both forward and backward. Validation of the tools is essential but not always performed. There is a need for stricter usage rules enforcers and stricter impact analyses.

Traceability: There is a strong need to follow a requirement from design to source code and vice versa. There are often emergent requirements that are derived from coding constraints. These also need to be added to the design to complete the traceability loop. Traceability from source code to object code is also a requirement for certification at the highest levels of criticality.

Tools: The tools and techniques used for certification need to be validated themselves to make sure that they fulfill the purpose for which they are used.

Coding standard: There are many compilers that enforce and demonstrate compliance to usage rules (e.g. MISRA, JSF). However, these usage rules are a bare minimum and leave much to be desired in safety and security properties.

What's missing? There is a need to address not only what should go into the requirements but also what happens when something is not there (or how to specify what should not be there). The process should address the deviation and waivers of requirements and be able to analyze their impact on the design, on integration and testing, on verification and validation, and on overall system level functionalities.

Verification

What kinds of proofs are required? What kinds of tests are performed?

Models should be used for verification but this is not the norm yet. Schedulability analysis is critical but lacks formal support. Proving timing, concurrency, and assertions properties comes out strong in the list of priorities. Unit and coverage tests seem to be the two main testing techniques.

Model vs. code: There is a strong push but little support yet to verify models as opposed to source code. A family of new verification tools will be needed both to verify the models and the tools that manipulate these models and generate code. As done with software testing, the model should also be able to include the hardware in the loop. Simulation models in tools such as Matlab/Simulink [10] are used as well. Modeling graphical languages (e.g. AADL [6]) are being explored. Complexity arises in the sense that the learning curve for such languages is quite steep. There are also problems with certification in terms of configuration control inertia (i.e. how dependent is the result from the configuration of the tools?).

Schedulability: Analyzing the schedulability of the whole system is critical. This is often misunderstood and a dangerous rule of thumb is often used. This rule of thumb states that, during tests, if the utilization does not go over 70%, you're ok; the system will not be overloaded. Dependency between components is also a problem as it is often not considered correctly in the schedulability analysis. DMA [11] and RMA [12] are two techniques that are widely used.

Proof: Timing is a paramount issue in real-time systems. The analyst must make sure that there are no external timing impacts when performing a change. The way the system handles the latency of warnings is a certification issue. Concurrency is also critical and preventive analysis using simulation techniques should be used to detect potential deadlocks. The verification of pre-/post- conditions and invariants through assertions is also common but proving that it is handled correctly is a concern. Coupling analysis is required for both data and control flow. Dynamic analysis to gather performance metrics and detect problems, such as memory leaks and partition violations are needed. There is also a need to test for interoperability among systems and provide some sort of proof of independence (partitioning). There is an add-in to RoseRT [13] called Quality Architect that can be used to perform verification and validation. Among other things, it does sequence chart comparison for both validation and regression testing, as well as static race condition checking. It also does automated test stub generation to allow for partial system testing.

Testing: There is a will to use a test first development approach, which means that the test should be written and developed before the actual system. This is not the norm, however. Unit tests can currently be achieved using tools that automatically generate a test harness. Such cases do no usually test for all of the out-of-range conditions and other singularities. Airworthiness certification requires specific levels of structural coverage, based on the assessed criticality of the software. There is a need to test for and track changes in configuration throughout the lifecycle. All test inputs/outputs should be recorded and accessible in order to be able to perform coherent regression testing.

Verification (continued)	*What's missing?* There is a need for a whole family of verification tools that are closely integrated and can integrate hardware in the loop. The 'system' should have an array of available solutions that is customizable to the task at hand and the availability of different inputs. Recognizing that there is no 'one-button-does-it-all' solution, the system should nevertheless be able to greatly speed the work for human analysts. For testing, there should be a way to address the particular problems of developing features in parallel and making sure that tests are coherent for the whole system and not just parts of it. The same set of tools and methodologies would also apply to testing patches. Furthermore, there is a need to adapt testing procedures to perform at different levels of integrity, depending on what standard it is tested against. There is also a need to standardize verification in such a way that the results of different analyses can be compared and amalgamated. Good tools are emerging [14] but do not work well with each others. Finally, the process should be agile because new rules are forthcoming (e.g. RTCA DO-178C/ED12C guidelines, expected in 2009).

5 Third Domain: Visualization

The idea behind the third exercise was to identify the different kinds of graphs and graphical representations that are commonly used when tackling avionic software problems. Another goal was to identify problems or known solutions related to the graphical user interface.

Charts, diagrams, and graphs

What types of graphical representations are used to design and understand an avionic system?

There is a lack of good visualization specific to real-time and embedded systems. Dynamic structures and properties need to be seen. State diagrams are underused and many charts from UML and SysML should also be used more.

Dynamic: There is a need to visualize the real-time aspects of processors and memory. For example, there is a need to visualize dynamic structures and objects to pinpoint memory leaks, partition violations, and potential deadlocks of resources. There is currently some tool support for this but the output is mostly textual, which requires a lot of effort to understand. There is a need for a visualization that would show time and space independence of a system (e.g. ARINC 653 [15]). There are some dynamic visualization techniques in RoseRT such as sequence diagrams from an execution trace and animated state charts.

Static: A state diagram is a powerful and underused technique. It can show unknown/undefined states that should not be present in the software. Time slicing charts, flow charts, and timing diagrams for busses are used as well. A UML activity diagram is considered as the modern flow chart. In fact, the whole gamut of UML diagrams are employed at some point. However, UML diagrams should be tied in with the code and not be used only as documentation as is currently the case. SysML is relatively new but seems to be gathering speed. Call graphs specialized to show coupling among units are useful. There is a need in general for graphs to be more responsive to changes in other graphs. There seems to be a lack of integration both between the graphs themselves and the design/code. There is a need for a clear display that shows test coverage and traceability at different levels. There is also a need to be able to clearly follow data and control flows. There are some good tools that are used in academia (e.g. USE (OCL) [16, 17], Alloy [18]) but not in the 'real' world.

What's missing? Choices for visualization are too broad. There is a need to clearly define what is the minimum set to look at in order to better define the needs. Efforts need to be devoted to finding out what has been done so far. There is an extensive repertoire of work in high-integrity software, including real time and embedded. There is also considerable previous work in this area for generic software engineering [19].

User Interface

Who are the good, the bad, and the ugly of interface design?

There are many good examples of good interface design (e.g. Google and Nintendo Wii). Unfortunately, there are probably more bad examples (e.g. Rational, ARTiSAN). Most tools have big flaws and designers do not show due diligence in this area.

Good: Everyone can probably agree that Google specializes in simple and intuitive interfaces. One fine example is Google Earth and its 3D navigation system. The use of mock up simulators to show the human-machine interface to real users (pilots in this case) is of paramount importance. It can also be used to test maintenance interfaces. Multiple/larger screens are more and more prevalent as they really improve efficiency. The almighty 'undo' button is an absolute essential but it can be improved. Tool should support saving everything so that a diagram and a specific view do not need to be generated anew every time. The gaming industry has some good ideas (and some bad ones). For example, the Nintendo Wii might be on the right track with its new wireless remote that uses player movements as well as more traditional button presses to control the action. Visio is another tool that is commonly used. Total and easy configurability is always a good idea as it can prevent a lot of user frustration.

Bad: Delays between changes in diagrams are still much too long. Rational is recognized as having a counterintuitive interface. It is much too broad and un-integrated in its approach (e.g. Rose vs. RoseRT). ARTiSAN Studio [20] is used in some places but there are problems with the interface. Among other things, it is difficult to navigate through the model to a specific elements; scrolling and zooming are done very poorly; diagrams created automatically are so cluttered that they become unusable; and these things are not easily modifiable (most of the time they are not even configurable!). In fact, layout and edge routing are often problematic as elements get on top of one another and make all of them illegible. Filtering is also a problem as it is usually not simple in the tools. The only way is often to create a whole new diagram. Sometimes, there are no clear indications of the relations between elements of the diagrams. For example, most tools will not warn you that deleting something in one diagram may impact other diagrams. There are often too many ways to do the same thing. In some cases, this may be good, but, most of the time, it only confuses the user.

What's missing? Visualization and graphical user interfaces must pay more attention to human system integration as it is critical. A standard that would allow discussions on the model at any level from managers to bits on the wire is needed.

6 Fourth Domain: Process Support

The fourth exercise was divided into two parts:

- Characterize the software verification process: The idea was to identify the software engineering processes that are required for avionic software verification.
- Identify problematic areas: The idea was to identify specific tasks that are currently problematic in the avionic software process and might require research.

The last item being the main goal of the whole workshop, it deserves the whole following chapter (7: Problematic Areas). The first item is discussed next.

Proportion of effort

How much of the avionic software engineering process is dedicated to verification?

About 30% of the verification process is dedicated to verifying the software itself. The rest is performed mostly on 'paper' (it might be in electronic form).

It seems that about 30% of the avionics verification process is spent on verifying the software. The other 70% consists in validating the paper trail (e.g. requirements, models, test procedures). As always, there is a discrepancy between the software and its documentation. As manual code inspection is often impractical, more attention should be given to the developed material (model or code) or a MDA/MDD approach should be taken. Unfortunately, this is rather uncommon. Test scenarios generally include mostly normal conditions and do not test enough for abnormal ones. Furthermore, about 50% of errors found are due to poor requirements. There is a definite need for tools that will help create more complete and coherent requirements. There is a general tendency to respect the bottom line in the contract rather than proceed with due diligence. For example, unit tests are not a requirement for certification, so they will often not be performed and test will be source-code based rather than requirement based. However, some safety tests are required by the certification process. Static analysis is more used than dynamic analysis, mainly because it does not require the creation of tests, which should be designed to offer good code coverage. However, the explicitness of dynamic tests is often cited as an advantage over the black box that often is statc analysis, even if in practice it generally gives excellent code coverage (both control flow and value domains). Code reviews are usually an informal process. As it is currently impractical for large systems, it only catches syntactic and "pretty printing" problems. Most experts agree that it should be performed more thoroughly as it helps to catch errors early in the process. Documentation reviews are performed for consistency and understanding but are considered insufficient for design assurance and compliance artifact.

Target

What are the goals of software verification?

Both product qualification and safety certification (airworthiness) are the end goal but the importance of consistency, completeness, and agility was also stated.

Safety seems to be the ultimate goal for most applications, but product qualification is also very important. Other goals that were stated and generated additional discussions are: consistency, completeness, and agility. Consistency is very important but requires a Spartan discipline. Generating the documentation automatically from the source code or the model would solve this problem. Completeness is a contractual obligation. Any deviation from the original plan must be justified and signed off. The same goes for correctness. Agility is a worthy goal but it depends heavily on the strength of the original architecture and the level of erosion in the maintenance, as most systems will 'live' for over 25 years. A good architecture from the get-go also ensures a lower learning curve for maintainers. Tracking the changes to the requirements can often give an idea of where the proposed architecture might be deficient.

Requirements

What is the role of the requirements in software verification?

Most requirements are stated in natural text, which is a problem as it lacks rigor. DOORS is mandated by ADM(Mat). Traceability to requirements is currently very important; some feel too much so.

As natural text lacks rigor, interpretation of the requirements is always an issue. A clear glossary is an important factor in making sure that all stakeholders understand the same thing. Text has its place in less critical areas, such as user interface, but more formal representations should be used for more critical systems. Formal methods are useful but generally do not scale to the whole system, although they are becoming ever better (as does systems!) The DOORS project management tool [21] is mandated by ADM(Mat) but is often not used to its full potential or worst, misused. Everything should at least be linked through DOORS (or an equivalent product) to allow complete traceability. There is a disconnection between contractual obligations and due diligence. For example, some feel that the importance of traceability is overstated as, sometimes, it just ensures that bad requirements find their way into every aspect of the development and absent but good requirements are never expressed. Forcing contractors to apply due diligence could help in alleviating this problem.

What's missing? There is a need to develop a process to define clear and precise scenarios that would link requirements to code. This would allow more intelligent testing by better controlling external conditions, including abnormal ones. There is a need for standardization, so that lessons can be learned from previous projects and make sure that everybody speaks the same language.

---

7 Problematic Areas

Nine activities and tasks requiring research and development were originally identified as commonly problematic by the personnel working on military avionic software. Upon further inspection, five of the tasks were subtasks of others, so four problems are described below. A consensus was reached as to their priorities and this is reflected by the order in which the problems are presented, the first one being the highest on the agenda. As can be seen, the level of detail generally drops as we go down the list. This makes sense as the most pressing matter is probably the one that was given the most thoughts. However, other important problems may be identified later from the raw data (Annex B) but this is what came out of the 2-day workshop.

Problem 1: Avionic software toolbox and methodologies

The keyword here is probably integration. There are many good tools but most of them do not talk to each other. A research project should leverage these tools as much as possible and only develop new tools when necessary. Tools should adapt to the methodologies and not the reverse, which is often the case with current tools. There is also a need to keep on top of new technologies and train the engineers as they and the technology become available.

Analysis: Safety analysis should be integrated in the design right from the start and tools should support this. Analyzing the scheduler and making sure that the design works correctly and that timing constraints hold is very important. Coupling analysis is essential but there seems to be considerable room for improvement in tool support. Impact analysis should be combined with other analyses whenever a change is performed on the software in order to make sure that the change respects the overall design and that affected parts are correctly retested. This will ensure that the software does not become so convoluted that it can no longer be maintained, let alone certified. In fact, different 'versions' of the program, from requirements all the way to maintenance, should never be unrelated. Everything should be traceable from one version to the others. Another needed analysis is to develop a way of verifying the compliance of partition/protection integrity (e.g. ARINC 653).

Modeling & Coding: Model driven design and development should be thoroughly investigated as the projected gains are huge. Any modeling tool should at least be able to model concurrency constraints (e.g. mutual exclusion, precedence, release times, completion times). In the current way of things (i.e. not model driven), there is a need to recover a complete architecture from the source code. If successful, this could provide a way to use model driven architecture and design in the maintenance of legacy code. In any case, this could be quite useful in understanding any software, not just avionics. There is a definite incentive for coders to learn what is termed 'defensive coding', so training should be provided. Existing tools and techniques could be leveraged to improve and automate parts of this but integration and adaptation will be required.

Testing: Automating test coverage, as per safety requirements, is not well supported by current tools. For example, they lack in baselining and auto updating as code and requirements evolve.

Problem 1 (continued)

Most tools also offer little to no support in covering not only each line of code but also each of the requirements from a functional perspective. It would also be nice if the tool was able to derive input test cases to cover areas missed by the test coverage. Regression testing is also a problem as it is difficult to make sure that maintenance activities do not cause more problems than they solve. Black box testing could also benefit from more tool support. For example, the development of realistic simulations could be made easier and more standard.

Visualization: Software is becoming bigger and more complex. Navigating through this complexity thus becomes a challenge. Therefore, the toolset should come with good visualization paradigms and a very good, state-of-the art user interface [19].

Problem 2: Capture and use requirements in a more proactive fashion

In theory, requirements are set in stone; in practice, they evolve and become outdated. There is a need to develop a set of tools and methodologies that would link together the software tools to provide better traceability and different levels of detail.

Correctly capturing the requirements on any software project is always a challenge. This is even truer for avionic software as lives depend on it. Currently, there seems to be a tendency to have 'dead' requirements. You set them once and never change them. The problem is that real life is not that static. Requirements will change in time and, as the project progresses, forgotten or imprecise requirements will emerge as people involved better understand the problem.

A set of tools should be able to handle 'live' requirements and allow the triumvirate of requirements, architecture, and code to remain constantly synchronized. This way, it becomes possible, for example, to link the requirements for safety with those for security, making sure that the code for one does not interfere with the code for the other. This will also greatly improve the traceability between the requirements and the actual code that fulfills them. It is also necessary for airworthiness to be able to exhaustively bind functionality to code to prove due diligence. In many cases presently, this seems to be a haphazard and manual process technology-wise. There is room for improvement at the very least.

The tool should also allow the requirements to be expressed at different levels of detail. At first, text can be used to put the ideas in place. As the requirements are refined, the tool should allow (force?) the analyst to use more formal representations. In a perfect world, refining the requirements would eventually turn them into a model, which in turn would generate the code automatically...

Problem 3: Technology watch and tool categorization

The objective is to develop a capability to provide expert advice on existing tools, to benchmark and categorize new ones, and to develop methodologies to fully tap their potential.

Certification standards, such as RTCA DO-178B/C *Software Considerations in Airborne Systems and Equipment Certification*, require evidences showing that the software used in avionics is correct. Because of the complexity involved, verification tools need to be used to assist the analyst in the production of these evidences. The trend seems to be that these verification tools will continue to be more involved in the process and the analyst will have to trust these tools to a greater extent.

Hence, there is a need for making sure that these software verification tools are trustable. Formally proving their correctness would require a white-box analysis (i.e. with the source code). However, because of intellectual property and the trade secrets involved, this would probably not be feasible. Anyhow, the level of complexity of that kind of proof would be staggering, so this is not an approach we would suggest. A black box analysis focusing on the detection of false negatives, with many cleverly designed tests giving a good coverage of the problem space, could give reasonable assurance on the capabilities of these verification tools.

There is a need to develop expertise, methodologies, and tools to better keep track of new techniques and tools as they become available on the market. This includes studying and recommending approaches to deal with new and disruptive technologies to make sure that the Canadian Forces leverage these technologies.

Another aspect of this effort is to be able to provide expert advice on what tools should be strongly recommended in contracts. The flip side is to be able to validate contractors' claim on the tools they propose to use.

A set of benchmarks are thus needed to be able to compare tools and techniques. This effort should leverage solutions developed for other problems in this chapter.

Problem 4: Quick access to current capabilities

There is currently no way to get a newcomer up to speed in all aspects relevant to a particular avionic project within a reasonable and practical amount of time. The problem consists in finding one.

Avionic systems in general and their software in particular are quite complex. It is practically impossible for someone or even a small team to understand every aspect of it. Unfortunately, this is often expected from the people who write the contractual requirements, which leads to a series of problem (e.g. outdated requirements, misunderstandings, etc.) There is a solution that consists in training someone intensively for 10 years whom will be used exclusively for this, chaining him to his desk. Practically, people come and go and a more practical approach could be to develop a set of techniques to develop crash courses illustrating the capabilities relevant to a specific project. This could take the form of a customized 'movie' or strongly directed readings. Some will argue that documentation could fill this role but the documentation is usually quite too large to be considered efficient. The idea is to get the personnel up-to-speed in a few days maximum, ideally in a few hours. This set of tools would also be quite useful to introduce newcomers to any of the teams in the avionic system development. In fact, such a capability would come in handy on any software project.

To address this, a project should start small. For example, developing a prototype to illustrate the capabilities of one specific tool (e.g. OASIS [22]). Learning from this experience, it could move to larger and larger systems and processes to describe eventually the entire set of Canadian avionic capabilities. A stepping stone to go this big would be to develop a scenario of a typical avionic software development project and adapt this scenario to the project under study. This would have the side effect of providing a scenario to test most of the potential tools discussed in this chapter and could serve as a platform to develop expertise.

8 Conclusion

This document presents a thorough summary of the points that were discussed during a two-day brainstorming session on avionic software. As such, it is not a complete overview of all the problems related to this vast field but only a summary of what was discussed during these two days. Four software engineering (sub)domains were identified and seeded into the collaborative tool to get the sessions going:

- Extraction
- Analysis
- Visualization
- Process support

From these four domains, four pressing problems were extracted and prioritized on the second day:

- *Problem 1: Avionic software toolbox and methodologies*
  The keyword here is probably integration. There are many good tools but most of them do not talk to each other. A research project should leverage these tools as much as possible and only develop new tools when necessary. There is also a need to keep on top of new technologies and train the engineers as they and the technology become available.
- *Problem 2: Capture and use requirements in a more proactive fashion*
  In theory, requirements are set in stone; in practice, they evolve and become outdated. There is a need to develop a set of tools that would link with the other software tools to provide better traceability and different levels of detail.
- *Problem 3: Technology watch and tool evaluation*
  The objective is to develop a capability to provide expert advice on existing tools, to benchmark and categorize new ones, and to develop methodologies to fully tap their potential.
- *Problem 4: Quick access to current capabilities*
  There is currently no way to get a newcomer up to speed in all aspects relevant to a particular avionic project within a reasonable and useful amount of time. The problem consists in finding one.

The task is now upon the team in DRDC Valcartier to analyze these results. In the short term, various surveys are underway to address aspects specific to real-time and embedded systems (e.g. C/C++ subsets, current analysis techniques and tools, and visualization techniques). A plan has been put in place to organize specialized training, the first part of that plan was a four-day UCLA class on digital avionics that took place in January at DRDC Valcartier. More training is in the scope for the fall. In the spring, work will begin on feasibility studies on relevant projects identified with the help of the workshop results discussed in this document and the results of the upcoming surveys. The goal is to discuss projects with DTAES in summertime and be ready to propose three-year projects in the fall for kickoff in April 2008. At this point, the doors to collaboration are wide open...

References

1. Object Management Group (2007). Unified Modeling Language: Superstructure. http://www.omg.org/technology/documents/formal/uml.htm. Read the PDF.

2. The Motor Industry Software Reliability Association (MISRA) (2004). MISRA C. http://www.misra-c2.com.

3. United States Department of Defense (2005). Joint Strike Fighter (JSF) C++. http://www.jsf.mil/downloads/down_documentation.htm. Read the PDF.

4. Charpentier, Robert; Salois, Martin; Bergeron, Jean; Debbabi, Mourad; Desharnais, Jules; Giasson, Emmanuel; Ktari, Béchir; Michaud, Frédéric, and Tawbi, Nadia (2001). Secure Integration of Critical Software via Certifying Compilers. In *Information Technology Security Symposium (CSE-CITSS)*, Canadian Security Establishment (CSE). Ottawa, Canada: (CSE). Read the PDF.

5. Shepard, Terry; Kelly, Diane; Smith, Ron; Chisholm, Ron; Jackson, Todd, and Mondoux, Paul (2006). Inspecting Designs in the Context of Model-Driven Development. In *CASCON '06: Proceedings of the 2006 Conference of the Center for Advanced Studies on Collaborative Research*, New York, NY, USA: ACM Press. http://doi.acm.org/10.1145/1188966.1189002. Read the PDF.

6. SAE (2007). Architecture Analysis and Design Language (AADL). http://www.aadl.info.

7. Charpentier, Robert and Carbone, Richard (2004). Free & Open Source Software: Overview and Preliminary Guidelines for the Government of Canada. (External Contract Report DRDC Valcartier ECR 2004-232). Defence Research & Development Canada – Valcartier. Québec (QC), Canada. Read the PDF. 271 pages.

8. The Embedded C++ Technical Committee (2002). Embedded C++. http://www.caravan.net/ec2plus.

9. Charland, Philippe; Dessureault, Dany; Lizotte, Michel; Ouellet, David, and Nécaille, Christophe (2006). Using software analysis tools to understand military applications. (Technical Memorandum DRDC Valcartier TM 2005-425). Defence Research & Development Canada – Valcartier. Québec (QC), Canada. Read the PDF.

10. MathWorks (2007). MATLAB & Simulink. http://www.mathworks.com/products.

11. Tindell, Ken (2000). Deadline Monotonic Analysis. http://www.embedded.com/2000/0006/0006feat1.htm. Read the PDF.

12. Coombes, Andrew (2002). Deadline Timing and OSEK. http://www.embedded.com/story/OEG20021114S0039. Read the PDF.

13. IBM (2007). Rational Rose Technical Developer. http://www-306.ibm.com/software/awdtools/developer/technical. Rose Real-Time (RT) is now part of Technical Developer.

14. Michaud, Frédéric and Carbone, Richard (2007). Practical verification and safeguard tools for C/C++. (Technical Report DRDC Valcartier TR 2006-735). Defence Research & Development Canada – Valcartier. Québec (QC), Canada. Read the PDF. 66 pages.

15. Aeronautical Radio, Incorporated (ARINC) (2006). ARINC Specification 653. https://www.arinc.com/cf/store/catalog.cfm?prod_group_id=1&category_group_id=3.

16. Object Management Group (2006). Object Constraint Language 2.0. http://www.omg.org/technology/documents/formal/uml.htm. Read the PDF.

17. University of Bremen Database Systems Group (2006). A UML-based Specification Environment (USE). http://www.db.informatik.uni-bremen.de/projects/USE.

18. Software Design Group (MIT) (2006). The Alloy Analyzer. http://alloy.mit.edu.

19. Lemieux, François and Salois, Martin (2006). Visualization Techniques for Program Comprehension: A Literature Review. In *The 5th International Conference on Software Methodologies, Tools and Techniques 2006 (SoMeT '06)*, ISBN: 1-58603-673-4, pp. pp. 22–47. Québec (QC), Canada: IOS Press. Read the PDF.

20. ARTiSAN Software (2007). ARTiSAN Studio. http://www.artisansw.com.

21. Telelogic (2007). DOORS: Requirements Management for Advanced Systems and Software Development. http://www.telelogic.com/products/doors/index.cfm.

22. Charland, Philippe; Ouellet, David; Dessureault, Dany, and Lizotte, Michel (2006). Opening up Architectures of Software-Intensive Systems: A Functional Decomposition to Support System Comprehension. (Technical Memorandum DRDC Valcartier TM 2006-732). Defence Research & Development Canada – Valcartier. Québec (QC), Canada. Read the PDF. 94 pages.

List of Acronyms

AADL	Architecture Analysis and Design Language
ADM(Mat)	Assistant Deputy Minister (Materiel)
AERAC	Aerospace Engineering Research Advisory Committee
ARINC	Aeronautical Radio, Incorporated
DMA	Deadline Monotonic Analysis
DND	Department of National Defence
DRDC	Defence Research & Development Canada
DSS	Decision Support Services
DTAES	Directorate of Technical Airworthiness and Engineering Support
FOSS	Free and Open Source Software
ITAR	International Traffic in Arms Regulation
PDF	Portable Document Format
GCC	GNU Compiler Collection
IP	Internet Protocol
JSF	Joint Strike Fighter
Mbps	Megabits per second
MDA	Model Driven Architecture
MDD	Model Driven Development
MISRA	Motor Industry Software Reliability Association
OASIS	Opening-up Architectures of Software-Intensive Systems
OCL	Object Constraint Language
RMA	Rate Monotonic Analysis
RMC	Royal Military College
UAV	Unmanned Air Vehicle
UCLA	University of California, Los Angeles
UML	Unified Modeling Language

Annex A
List of Participants

The following participants acted as subject matter experts:

Name	Organization (position)	Email	Phone	DSS Identifier
Maj Alain Beaulieu	RMC Kingston	beaulieu-a@rmc.ca	613-541-6000 ext. 6196	RMC
Julien Bourdeau	DTAES (5-5)	bourdeau.ja@forces.gc.ca	613-998-7315	DTAES 5-5
Robert Brodie	DTAES (6-4C1)	brodie.rgs@forces.gc.ca	613-991-9560	DTAES 6-4C1
Maj Shawn Durling	WSM DET Mirabel	durling.sh@forces.gc.ca	450-476-4697	WSM Det. Mirabel
Sylvain Fleurant	DTAES (6)	fleurant.sjlj@forces.gc.ca	613-993-1191	DTAES 6
Edison Nascimento	DTAES (6-6-2)	nascimento.eh@forces.gc.ca	613-990-7520	DTAES 6-6-2
Alain Nobert	CAE (CF-18)	nobert.a@forces.gc.ca	450-746-4106	CAE
Carl Sénécal	CAE (CF-18)	senecal.c@forces.gc.ca	450-476-4109	CAE
Maj Ron Smith	RMC Kingston	smith-r@rmc.ca	613-541-6030	RMC
Capt Trevor Stevens	14SES	stevens.tml@forces.gc.ca	N/A	14 SES
Capt Guillaume Vigeant	WSM DET Mirabel	vigeant.g@forces.gc.ca	450-476-4697	WSM

The following representatives from DRDC Valcartier organized the workshop and were present, except for Michel Lizotte:

Name	Email	Phone
Philippe Charland	philippe.charland@drdc-rddc.gc.ca	418-844-4000 ext. 4491
Dany Dessureault	dany.dessureault@drdc-rddc.gc.ca	418-844-4000 ext. 4109
Geneviève Dussault	genevieve.dussault@drdc-rddc.gc.ca	418-844-4000 ext. 4754
Michel Lizotte	michel.lizotte@drdc-rddc.gc.ca	418-844-4000 ext. 4495
Frédéric Michaud	frederic.michaud@drdc-rddc.gc.ca	418-844-4000 ext. 4165
David Ouellet	david.ouellet@drdc-rddc.gc.ca	418-844-4000 ext. 4596
Martin Salois	martin.salois@drdc-rddc.gc.ca	418-844-4000 ext. 4677

And our gracious hosts from DSS:

Name	Position	Email	Phone
Sylvie Jolicoeur-Wojcik	Operator	jolicoeur-wojcik.smc@forces.gc.ca	N/A
Jennifer Walsh	Manager	walsh.jj@forces.gc.ca	613-945-1295

Annex B
Raw Data

This section contains the data that was collected and prioritized for each domain during the two-day workshop. Each item is preceded by the identifier of the person who made that entry. Refer to Annex A for the list of participants and their corresponding identifiers. Items marked with a (Seed) identifier were put in by the team before the workshop to start the process along.

Entries were not edited. They express the personal opinions of the individuals and do not necessarily reflect the views of their respective organization.

B.1 First domain: Extraction

- **(Seed)** Development environment
  - **(RMC)** Safety analysis tools
    - **(RMC)** Development environments should include safety analysis tools that can track the analysis to the various configuration items
  - **(RMC)** Education
    - **(RMC)** We currently use Eclipse and Rational suite of modeling tools to teach (Rose RT, Rational Rose Enterprise,...)
    - **(RMC)** We use debuggers and instrumentation of the code very little as part of the courses we teach for example coverage tools are not a main stay.
  - **(DTAES 6)** Modeling
    - **(DTAES 6)** Now/Future: Assess Operational Requirements via scenario based modeling and simulation to define Functional Requirements and then start its allocation
    - **(WSM)** need to incorporate model driven prototyping into the requirement definition process
    - **(DTAES 6)** Need Model Checkers
    - **(DTAES 6-6-2)** System modeling tools such as MatLab with its Simulink tool box are used to develop real-time control systems design and to provide simulation artifacts for system analysis.
    - **(DTAES 6-6-2)** Modeling graphical languages such as AADL that can provide models at not only system levels but also at hardware and software levels need to be explored.
    - **(DTAES 6)** Need to get client aware of the benefits
    - **(DTAES 6-6-2)** Formal validation of a system model needs to be accomplished before the software design process starts.
    - **(CAE)** For the design part of future project, we could use UML model (at least for documentation).
    - **(DTAES 5-5)** Airworthiness qualification requirements for modeling tools, including a degree of configuration control inertia.
    - **(14 SES)** Problem with most modeling efforts is that it is using a language to communicate system behavior that no one speaks very well. More time must be spent learning how to use the modeling language than anything else.
  - **(DTAES 6-6-2)** Contractors use development model-based design tools with auto-code generators
    - **(DTAES 6)** NDHQ Clients lack knowledge of model-based design and therefore do no request any info/data from con-

tractors who could have otherwise better inform us about the architecture/characteristics of our system

  - **(DTAES 6-6-2)** Verification Environment
  - **(DTAES 6-6-2)** Schedulability analysis tools that perform RMA, DMA, etc. are needed for RT systems
  - **(DTAES 6-6-2)** structural coverage analyzers
  - **(DTAES 6-6-2)** intelligent test case generators
  - **(DTAES 6-6-2)** data flow and control flow analyzers (coupling analyses) are needed
  - **(DTAES 6-6-2)** requirements-based test coverage analyzers are needed
  - **(DTAES 6-6-2)** requirements traceability analysis tools are needed for forward and backward traceability
  - **(DTAES 6-6-2** dynamic analysis tools for RT systems to detect problems like for example memory leakage, partition violations, and gather performance metrics on the processor and memory management schemas are needed.

- **(DTAES 5-5)** General guideline: Fidelity to actual platform installed in aircraft
  - **(DTAES 6)** Need interoperability to be addressed
  - **(DTAES 6)** Need Standardization
- **(DTAES 5-5)** Incentive for avionics industry (currently 90% US based) to align with our strategy
- **(Seed)** Compiler
  - **(DTAES 6)** Contractors use sub-set of standard Compiler (for C++) in order to maintain Deterministic and safe behavior
  - **(WSM)** Developers to use model driven architecture tools to auto generate JSF C++
  - **(CAE)** For most of our development we are using Assembler for 8086, 8080 and AYK-14. I don't know the name of the assembler tools. For a small project
  - **(DTAES 6)** Need guidance or even tools or better standards in order to define minimum characteristics acceptable to clients in order to compile safe code
  - **(CAE)** On a small prototyping project we are using the C++ compiler for GNU.
  - **(DTAES 5-5)** Direct traceability demonstration requirement between source and object for the highest criticality software (Level A)
  - **(14 SES)** Green Hills Ada95 Compiler
  - **(14 SES)** gcc for C code
- **(Seed)** IDE
  - **(CAE)** On a small prototyping project we are using Tornado II IDE and we will eventually switch to Workrkbench, both from WinrdRiver.
- **(Seed)** Libraries
  - **(DTAES 6)** Lack of softcopy i.e. just pdf or word file. Unable to search, extract relevant info, or to trace
  - **(DTAES 6)** Library not share by contractors (designers) to clients (Air Force)
  - **(DTAES 6)** ITAR (USA not sharing info) limits access to information
  - **(CAE)** On our prototyping project we are using an OpenGL library developed by ALT software. We don't have the source code.
- **(Seed)** Makefile
  - **(CAE)** A team of 2 or 3 persons are working on the make files and compilation system.
  - **(RMC)** We do not teach the "art of makefile" do we need to? We just give the students a makefile with pre-made instructions and they only need to include the name of the files at the top, we do not get into the meaning of the other stuff.
- **(Seed)** Simulator

    - **(WSM)** challenge: maintaining concurrency with actual aircraft software
        - **(WSM)** concurrency is easily achieved if the simulator uses the real avionics or emulated avionics. In such a case, the embedded software can be updated before its flight tested. Therefore illuminating the duplication of effort.
    - **(DTAES 6)** Need to develop realistic simulation of black boxes, interfaces, clear I/O,
    - **(DTAES 6)** Should talk about simulation versus simulators.
    - **(DTAES 5-5)** DO-178B Verification objectives achievement via simulation.
    - **(CAE)** In house developed PC based flight simulators and test stations using real avionic hardware.
    - **(RMC)** We use simulators only in the Assembler arena. Nothing else (other than MDD)
    - **(DTAES 5-5)** Mostly lacking in system's error handling demonstration, paramount to the airworthiness paradigm.

- **(Seed)** Documentation
  - **(RMC)** Relevance
    - **(RMC)** A key issue with software doc is - what is the relevance of it? Generally all doc that is not embedded in the product (code or model) is by its nature irrelevant. Doc is static, product is dynamic. Attempts to link the two with tools have invariably failed. Think of other complex architectures (buildings, bridges, airplanes). The architects & engineers do not communicate designs of these creations using static textual documents. In terms, of using any existing documentation to extract design, even if one could do such a thing, it has a very high probability of reflecting an intent or desire that has long since been surpassed by reality (code).
  - **(DTAES 6-4C1)** Specifications - Aircraft, System, Equipment, SW
    - **(DTAES 6)** Need to methods/tools to properly develop them
    - **(DTAES 6)** Need to address impact of standards
    - **(DTAES 6)** Need to use tools to document and of courser trace them back to operational requirements, functional and technical requirements.
    - **(DTAES 6)** tracibility forward & back ward required
  - **(DTAES 6-4C1)** Configuration Management Of Documentation (hard & softcopy)
    - **(DTAES 6-4C1)** Need to track / detect changes through development & life cycle
  - **(Seed)** Completeness
    - **(DTAES 6-4C1)** ICD usually incomplete in details - (e.g. not full w/MIL-HDBK-1553 details - missing precision, bus timing, system timing criteria or expect some e knowledge of User)
        - **(OASIS)** What is ICD?
        - **(DTAES 6-4C1)** Interface Control Document - the link between systems / boxes
    - **(DTAES 6)** Need tools to V&V them
  - **(Seed)** Quality
    - **(DTAES 6-4C1)** Softcopy - not / partially searchable (e.g. scanned PDF image not converted MS Word)
    - **(DTAES 6)** Use of natural language is inefficient
        - **(RMC)** Models could seriously help here,.
    - **(DTAES 6)** Need clear visualization and in a grade fashion (think levels of complexity or top-down or bottom-up)
    - **(CAE)** The electronic version is not always accurate and it is hard to find the required info in the paper version because of the amount of documentation.

    - **(RMC)** The highest quality documents have low word count and high diagram count (as long as they say the same thing).
    - **(DTAES 5-5)** Software Requirements Specification is getting a lot better, obviously mostly on high criticality systems...highly scrutinized for airworthiness,
  - **(Seed)** Quantity
    - **(DTAES 6-4C1)** ICD - 1000+ pages
    - **(DTAES 6)** Overwhelming: how do we visualize the information in a better fashion.
  - **(Seed)** Up-to-date
    - **(DTAES 6-4C1)** ICD Not always with the latest minor fixes
    - **(WSM)** Documentation should be integrated in the development process and be automatically updated as much as possible. (minimize human intervention in the documentation)
      - **(WSM)** Test cases should automatically fill the requirements document and the code should be accessible through an easy graphical interface that represents the structure of the code.
    - **(WSM)** need to improve configuration management across organizations to ensure latest documentation can be identified and accessed

- **(Seed)** Hardware
  - **(DTAES 6)** Architecture
    - **(DTAES 6)** Still using federated system of boxes
    - **(DTAES 6)** Moving toward MOSA : Modular Open System Architecture approach
    - **(CAE)** Architecture must support multilevel of DO-178B (in the same CPU) by using ARINC 653 compliant Operating System
  - **(CAE)** Hardware Platforms
    - **(CAE)** Proprietary Hardware should be avoided
    - **(CAE)** VITA 48/VPX (new VME generation)
    - **(CAE)** Compact PCI
  - **(Seed)** Binary code
    - **(Seed)** Disassembler
  - **(Seed)** Bus
    - **(DTAES 6-4C1)** MIL-STD-1553 (various versions)
    - **(DTAES 6-4C1)** MIL-STD-1760
    - **(DTAES 6-4C1)** Serial RS232, RS422, RS485, CSDB
    - **(DTAES 6-4C1)** AFDX - ethernet
    - **(DTAES 6-4C1)** ARINC 429 & maybe 629
    - **(DTAES 6)** Huge variety of dedicated lines of communication among boxes (old system)
    - **(DTAES 6)** Security of data is critical
    - **(DTAES 6)** Integrity is critical
    - **(DTAES 6)** Use of IT based busses coming up (Ethernet, IPs, etc..)
    - **(CAE)** IEEE 1394 (Firewire) the JSF Bus
    - **(DTAES 6-4C1)** E1553 - next 1553 standard
  - **(Seed)** Communications
    - **(DTAES 6)** Huge requirements for much broader bandwidth i.e. from data and verbal comm. to huge stream of video
  - **(Seed)** Network
    - **(14 SES)** Protocols
    - **(14 SES)** Topology
  - **(Seed)** Processors
  - **(14 SES)** Processor development is progressing at such a rate that I see little or no need to define any particular set of processors that are used. Any solution developed must be able to be modified for a new processor in a relatively short order.
    - **(CAE)** Yes but heat is a concern is embedded system

    - **(CAE)** intel 8086, 8080, AYK-14 and PowerPC (for futur project).

- **(Seed)** Human personnel
  - **(RMC)** Educators
    - **(RMC)** There is a wide gap between the educators and the customers. There is a need for the personnel working in engineering to participate in the composition of the graduating classes.
  - **(Seed)** Architects
    - **(14 SES)** System
    - **(14 SES)** Hardware
      - **(WSM)** need better coordination to ensure aircraft hardware configuration is known and planned for
      - **(DTAES 6)** Military Environments remain harsh and difficult
      - **(DTAES 6)** Obsolescence of HW is getting worse... short life
      - **(DTAES 6)** Need form fit and functions... as we replace old within new H/W... also address the S/W migration
    - **(14 SES)** Software
    - **(RMC)** We use this term in the industry, but as far as I know we have no education/training for such a beast. They only get that designation after years in the trenches. The SE degrees must focus on this objective and stop producing programmers. It is the equivalent of a Mechanical eng school turning out machinists. We need a huge shift in education. We were on our way when the high tech bubble burst - unfortunately enrollments across NA have dropped, and along with it, programs have been rolled back or paused (e.g. at Queen's for instance).
  - **(Seed)** Developers
    - **(14 SES)** There are code monkeys and then there are GURUs, not sure how to tell the difference unless you're in the code with them both. GURUs are quite often better at explaining the system than an Architect, but they generally do not freely participate in design reviews etc.
    - **(WSM)** contractors, public service and military
    - **(WSM)** in general a lack of understanding of DND's airworthiness requirements
    - **(DTAES 6)** Lack of model and simulation involving human in the loop
  - **(Seed)** Users
    - **(DTAES 6-4C1)** Pilots, Flight Engineers, Maintainers
    - **(WSM)** DRDC
    - **(DTAES 6)** Human Systems Integration becoming critical: Tolls are???
    - **(DTAES 6)** Despite being Engineers; customers (NDHQ) lack modernity in their approach.
    - **(WSM)** "Software is easy" mentality is problematic
    - **(DTAES 6)** Very diverse users for the same platform i.e. same software is more and more used by an extremely diverse clientele.
  - **(Seed)** Validator
    - **(DTAES 6)** Need better tools to interpret results from the contractor who did it
  - **(Seed)** Verifier
    - **(DTAES 6)** Customer needs better tools and better preparation for performing this at HQ. Or to better access the results of V&V done by the contractors.
- **(Seed)** Models
  - **(RMC)** Using models not only to produce documentation but also to do evolutionary prototyping as well as the final code.

    - **(RMC)** Too often models are used to try something and then are thrown away. Models that generate working code bring to the forefront the architecture of the system and make it visible. A modification to the code after does not weaken the architecture because it is always present and the code is modified through it.
- **(14 SES)** Flow Charts
    - **(DTAES 6)** Ned a great deal more visual representation: whole family of them
- **(CAE)** AADL
    - **(CAE)** There is some push in the industry for AADL with is a formal language that could used to generate a model
- **(CAE)** SysML
    - **(CAE)** SysML (System Modeling Language) which is based on UML with added System modeling capabilities received a good interest for the industries
    - **(RMC)** This initiative is evidence of the success of UML. The prospect of true sys eng is becoming possible, one where software and hardware co-design is truly possible. In past, allocation was a death sentence - never reversible after day 1 of the project. SysML holds the promise of making this decision transparent to the architect. we are not there yet.
    - **(DTAES 6)** Must marriage software and hardware. See successful approach done by General Dynamics Canada for the Maritime Helicopter Project. Usage of UML with SysML
- **(CAE)** Are systems developers/builders should provide an Executable Model as a deliverable?
- **(Seed)** Ad hoc
- **(Seed)** Data schemas
- **(Seed)** UML
    - **(Seed)** RT UML
        - **(RMC)** Not convinced that the "profiles" of UML are as the meta UML, in so far as, too many profiles may end being the downfall of UML. The evolution of ROOM into a toolset (ObjecTime Developer, then RoseRT, then Technical Developer) was not as significant as the incorporation of the strong aspects of ROOM into the UML 2.0 (structure diagrams, and the central role of sequence and state diagrams)
        - **(RMC)** There is a perception that MDD or MDA is unsafe because a tool automatically generates some of the code.
    - **(Seed)** UML RT
    - **(RMC)** We currently teach MDD for Real-Time software systems using RoseRT an evolution of ObjectTime
    - **(RMC)** There is currently a project that is funded by AERAC to study the impact of MDD on avionic software
    - **(RMC)** UML is fast becoming a defacto standard as more and more students in NA graduate with at least some exposure. The Air Force, imho, has not kept current in this area. Any future SoS will include varying degrees of UML, and UML-based tools. Standards become very important, for this becomes the "drawing set" equivalent in SE.

- **(Seed)** Source code
    - **(RMC)** Does the avionic software producers use pre and post conditions in languages as well as assertions? This is the kind of things for example provided by Eifell
    - **(DTAES 5-5)** Access to source code is NOT at all the norm in currently procured avionics systems; most of these systems fall under ITAR.
        - **(DTAES 6)** ITAR: Can we reverse-engineer object code?
        - **(DTAES 6)** Intellectual Propriety: Huge issue with access
    - **(Seed)** Languages

- **(Seed)** Ada
  - **(14 SES)** There is Ada83, Ada95, and I think 2005 these are fundamentally different languages
  - **(RMC)** We do not teach with Ada anymore, but I believe that it is a language that is worth teaching due to its high engineering value.
  - **(14 SES)** Language used for both AIMP and MHP
    - **(DTAES 5-5)** MHP is mostly C and some Ada 95
  - **(RMC)** Ada has not been taught at RMC for more than 10 years now.
  - **(DTAES 6)** Relatively common for High Integrity Systems but fading away
- **(Seed)** Assembly
  - **(RMC)** Students in Comp Eng get a strong background in assembly, generally on the Motorola 68000
  - **(CAE)** F-18 Missions computers, Store Management Set and Communication Set Controller are written is assembly
- **(Seed)** C
- **(Seed)** C++
  - **(CAE)** Used for prototype development.
  - **(DTAES 6)** Extremely common but huge discomfort from Safety aspect (V&V issues)
- **(Seed)** Java
  - **(CAE)** Some people in the industries tell that Java can become the best language for critical system?
  - **(DTAES 6)** Not common in Avionics at all
- **(RMC)** At RMC we have recently changed the curriculum to teach more than Java. It was recognized that C/C++ has remained an important set of languages, particularly for RT systems.
  - **(RMC)** We teach applied programming using C
  - **(WSM)** Although C remains an important language, the main difference with JAVA is the memory management is left to the users. I believe that today's high fidelity tools should not let the developers manage the memory and manage it by itself or via a manage garbage collector.
- **(14 SES)** Scripting languages (Perl etc)
- **(Seed)** Usage rules
  - **(Seed)** JSF C++
  - **(Seed)** MISRA C
  - **(Seed)** MISRA C++?
    - **(CAE)** I did not think that MISRA C++ exist,
    - **(CAE)** What About EC++ (Embedded C++)
  - **(DTAES 6-6-2)** For the new OASIS toolset what will be the input data entry criteria?
    - **(DTAES 6-6-2)** A compiler error free source code? Or any source code even if it has not been compiled yet?

- **(Seed)** Other
  - **(WSM)** Third party software
    - **(WSM)** need strategies/processes to incorporate s/w generated by different organizations while respecting internal engineering processes and the airworthiness requirements
    - **(DTAES 6)** FOSS becoming common especially with UAVs
  - **(WSM)** Real time philosophy
    - **(WSM)** A switch to priority based scheduling would greatly increase efficiency and would enable other valuable advantages like graceful degradation of the system.
      - **(WSM)** Although priority based scheduling seems harder to verify due to its infinite number of system state, there are mathematical foundations to prove the system scheduability.

    - **(DTAES 6)** Need to clearly define RT as a standard
- **(DTAES 6-4C1)** Network Enabled Operations (NE Ops)
    - **(DTAES 6-4C1)** Not just aircraft level but full network of platforms (future)
    - **(DTAES 6)** Must address linking A/C together, along with other platforms (satellites, ships, vehicules, UUAVs, etc.
    - **(DTAES 6)** SoS: Need to address interoperability among large systems.
    - **(DTAES 6)** Adaptable software to deal with specific threats (think electronic warfare - UDFs)

B.2 Second Domain: Analysis

- **(Seed)** Validation
  - **(DTAES 6-6-2)** Need to check forward traceability between the model and source code; Need to check the backward traceability between source code and the model; Need to identify elements in the model and source code which are missing traceability; Need to identify derived requirements who may not have traceability to higher requirements (e.g. design/architectural decisions).
  - **(DTAES 6-4C1)** Validation of analysis tools - How well do they document the SW for use in Certification?
  - **(Seed)** Model Traceability in Source Code
    - **(CAE)** Model should first to be used to validate customer requirements
    - **(RMC)** ignore
  - **(Seed)** Requirements Traceability in Source Code or Model
    - **(DTAES 6)** critical
    - **(DTAES 6-6-2)** System models require formal validation to system requirements before the real software design starts.
    - **(RMC)** Emphasis must be given to the dark side of requirements
    - **(RMC)** Functional Configuration Audits (part of Config Mgmt) must be supported by the tools.
    - **(RMC)** Allocation of requirements must be highly visible.
    - **(DTAES 6)** Need to address the Deviation and Waivers of requirements and how to analyze their impact on the design, I&T, V&V, and System level functionalities.
  - **(Seed)** Programming rules conformance
    - **(CAE)** Done through code review.
    - **(DTAES 6-6-2)** Need for automated source code checkers for demonstration of compliance to the adopted coding standards.
    - **(DTAES 6)** important in term of standardization for interoperability it impacts our confidence level of product from other customers, other partners, or other contractors.
- **(Seed)** Verification
  - **(RMC)** Model-based versus code-based V&V
    - **(RMC)** Need to examine those features/characteristics that can be verified at the model level vice the code. Move V&V to highest level of abstraction as possible.
    - **(RMC)** In a/c design or modification we verify (for certification for instance) based upon the model (usually drawings and math), and we inspect the product for conformance to the approved design. It should (could) be the same for software.
    - **(DTAES 6)** Family of Verification tools
    - **(DTAES 6)** No single solutions but what are the best tools/methods that best apply for RT application
    - **(DTAES 6)** How do we address verification for huge software programs as found on modern aircraft avionics?
    - **(DTAES 6)** Automation: Need to make it easy and fast
    - **(DTAES 6)** Need to address the hardware in the loop (especially for RT Avionics systems)
  - **(DTAES 6)** Security
    - **(DTAES 6)** Address unique requirements imposed by V&V of classified modules/components/or complete software comments
  - **(DTAES 6-6-2)** Schedulability analysis such as RMA, DMA, etc is very important for RT embedded systems (e.g. tasks with periodic, non-periodic, synchronous, asynchronous, preemptive, non-preemptive, priority-based, calendar-based, concurrency/rendezvous requirements, etc).

- **(DTAES 5-5)** Timeliness and re-usability of verification credentials/artifacts during development.
- **(Seed)** Proof
  - **(Seed)** Timing
    - **(DTAES 6)** Consideration in Real-time is paramount as it affects the quality/values of the running date
    - **(DTAES 6-4C1)** Verification of no external timing impacts by changes critical.
      - **(DTAES 6-4C1)** Latency of warnings a certification issue.
  - **(Seed)** Concurrency
    - **(RMC)** There must be a clear statement on what kind of deadlock resolution mechanism is being used and how the OS supports it.
  - **(Seed)** Stack Usage
  - **(Seed)** Ad Hoc Temporal Logic
    - **(DTAES 6-6-2)** Need detection methods for potential overflow/underflow conditions of counters
  - **(Seed)** Invariants
  - **(Seed)** Value Range
    - **(RMC)** Pre and post conditions with assertions and invariants should be used for this. The language has to be able to support it.
    - **(DTAES 6-6-2)** Need to auto test case generators not only for normal range but also out of range conditions as well as the error handling capabilities for singularities.
  - **(RMC)** Schedulability
    - **(RMC)** often overlooked / misunderstood. The 70% utilization rule of thumb is dangerous, as it usually is invalid in modern (multi-task, highly dependent) systems
    - **(DTAES 6-4C1)** Legacy fleet deterministic RT and future NE Ops priority based scheduling - both task & external interfaces
    - **(RMC)** Response-Time-Analysis approach to analyzing priority-based systems needs to be used in place of utilization analysis
    - **(RMC)** There must be a way to specify exclusion, precedence and deadline constraints and schedule tasks with these constraints. There is a need for a tool to do this.
    - **(DTAES 6-6-2)** Schedulability analysis such as RMA, DMA, etc is very important for RT embedded systems (e.g. tasks with periodic, non-periodic, synchronous, asynchronous, pre-emptive, non-preemptive, priority-based, calendar-based, concurrency/rendezvous requirements, etc).
  - **(RMC)** Dependency
    - **(RMC)** Related to schedulability above, tools are needed that extract and analyze the resource dependencies that exist between tasks in a multi-tasking system. This becomes key to conducting schedulability analysis

- **(Seed)** Detection
  - **(Seed)** Sanity Checks
    - **(WSM)** in-service fault detection is heavily dependent on end-user recognizing and documenting the problem

- **(Seed)** Testing
  - **(RMC)** Test-First Development
    - **(DTAES 6)** Define please
      - **(RMC)** Design a test before you code. It is a form of requirement verification. A concept mainly used in Agile (or light weight) methods.
    - **(WSM)** Probably means that the tests should be written and developed before the actual system. The system is then devel-

oped against the test gradually passing more and more tests until completion.

- **(DTAES 6)** Sequencing of testing
  - **(DTAES 6)** How to address parallel developments of new feature or correction and its combined testing
  - **(DTAES 6)** How do we addressed analysis of code development prior to actually designing it (i.e. lack of engineering discipline)
  - **(DTAES 6)** How do we address analysis of testing for "patches"
  - **(WSM)** require airworthiness considerations to be accounted for at project initiation vice at project acceptance
- **(Seed)** Unit Tests
  - **(CAE)** OASIS could integrate a static unit test analysis to verify for dead/unreachable code, etc.
  - **(DTAES 6)** Module testing of high-integrity S./W varies in accordance with the security and its safety integrity levels. We need grading for such testing
  - **(CAE)** Unit test can be achieved by automated tools that generate test harness
- **(Seed)** Fault Injection
  - **(Seed)** Fuzzing
  - **(DTAES 6-4C1)** validation of Design For Reliability / Failure Modes effects / degraded operation
- **(Seed)** Coverage
  - **(WSM)** F-18 test facilities incorporating code coverage into test environment
  - **(WSM)** F-18 flight test program and operational test and eval conducted by independent organizations who develop their own test procedures. Developers may make recommendations as to what needs to be tested however they do not staff the procedures.
  - **(DTAES 5-5)** Airworthiness certification requires specific levels (types) of structural coverage to be achieved, based on the assessed criticality of the software, mostly by testing (requirements based) or, exceptionally, by analysis.
    - **(RMC)** These levels depend on the criticality of software failure.
  - **(RMC)** Statement and Decision coverage is a good measure of the quality of a test when applied in a black box test environment.

- **(Seed)** Feature Location/Components Identification
  - **(Seed)** Software Reconnaissance
    - **(WSM)** The tool should try to extract as much as it can. However, humans could be used to finalize the more complex low level extraction of the regions of the code the tool cannot extract with high fidelity. The tool could even provide different possible interpretations that a user could confirm by looking at the code.
    - **(DTAES 6-4C1)** Useful for SNAG rectification
    - **(CAE)** Many views are required to understand a system, Static, dynamic etc ..
  - **(Seed)** Concept Analysis
- **(Seed)** Design Pattern Recovery
  - **(WSM)** Anti-patterns identification
    - **(WSM)** recognizing patterns is important but recognizing anti-patterns is also very important. An anti-pattern is something that should be avoided to solve a particular problem.
    - **(RMC)** Patterns are important to recognize, but the architecture of the software normally erodes as the software is maintained. In this sense the pattern observed might not be what we want to find.
- **(Seed)** Clustering

  - **(RMC)** Cross-cutting requirements have to be considered

- **(Seed)** Slicing
  - **(14 SES)** If the tool can provide a graphical representation of a slice, that would be appreciated.

- **(Seed)** Impact Analysis
  - **(RMC)** Encapsulation
    - **(RMC)** The goal of encapsulation (even pre OO) was to bound the impact of change. Highly encapsulated approaches to design such as that found in ROOM and now potentially in UML 2.0 aid in limiting change impact
  - **(DTAES 6)** traceability Analysis
    - **(DTAES 6)** CRITICAL especially with Airworthiness certification
  - **(DTAES 6)** Dependability Analysis
    - **(DTAES 6)** Quite critical especially as the system mature by getting new functions and loosing some too!
  - **(DTAES 5-5)** This is a key aspect of airworthiness as it is extremely desirable to keep the re-certification effort commensurate to the proposed changes.

- **(Seed)** Querying

- **(Seed)** Other
  - **(DTAES 6)** Standardization
  - **(DTAES 6)** Do we have any standards with analysis

B.3 Third Domain: Visualization

- **(Seed)** Dynamic vs. Static
  - **(Seed)** Dynamic
    - **(DTAES 6-6-2)** need to have dynamic analysis tools for visualization of the creation/destruction of polymorphic objects and their shifting behaviors
    - **(DTAES 6)** What is the minimum set of dynamic visualization proposed
    - **(DTAES 6-6-2)** need tools for visualization of the RT aspects of processor and memory management by the software (i.e. RTOS function aspects)
    - **(DTAES 6-6-2)** need tools for detection and visualization of dynamic structures/objects, memory leakage, and potential partition violations in the software
    - **(CAE)** Dynamic visualization that could show the time and space independence of a system (ARINC 653)
    - **(DTAES 6-6-2)** need to detect and visualize real-time aspects/parts of the software (i.e. both model and source code) where there is possibility of unbounded task priority inversions due for example to locked shared resources
  - **(Seed)** Static
    - **(DTAES 6)** Choices are too broad... what is the minimum set to look at.
    - **(DTAES 6-6-2)** test coverage analyzers should provide visualization outputs; same for traceability analyzers
    - **(DTAES 6-6-2)** need structural coverage analyzers that can provide visualization of statement and decision blocks that have / have not been covered by specific test cases
    - **(DTAES 6-6-2)** data flow and control flow visualizations at any user-defined level
  - **(DTAES 6)** Management of information and results
    - **(DTAES 6)** Need database and libraries, get organized!
- **(Seed)** Documentation?
  - **(Seed)** Create documentation using pretty pictures
    - **(RMC)** Using documentation that is in the design tools. Exchanging information in contractor/producer format
    - **(DTAES 6)** Allow for traceability to requirements and design
    - **(RMC)** The only documentation at the software level (vice system) that is not dangerous is documentation that is 100% coupled to the product (i.e., regardless of how it gets generated or where it lives) it is a true representation of the current req't, design, test ...
  - **(Seed)** As opposed to text?
    - **(DTAES 6)** Should allow for some formal methods docs (actual equations)
  - **(DTAES 6-4C1)** Collection of info on a specific change - e.g. "MS Binder" of various analyses
    - **(DTAES 6)** Should think in function of an Integrated Information Environment
  - **(DTAES 6)** Model
    - **(DTAES 6)** Need to see the model used while doing model-driven design
    - **(DTAES 6)** Show clearly the architecture
    - **(DTAES 6)** Enable the merging of new model/features
  - **(DTAES 6)** Formal Approval and its evidence
- **(Seed)** Modeling
  - **(Seed)** Ad hoc
    - **(DTAES 6)** For many, the norm!
  - **(Seed)** Free hand

    - **(DTAES 6)** Quite common for the staff at the HQ but really simplified
- **(Seed)** UML
  - **(Seed)** RT UML
    - **(RMC)** See earlier comments in 1st domain on over-specializing UML
  - **(Seed)** UML RT
  - **(WSM)** need a manageable implementation strategy that will allow for the successful integration of a modeling environment into a legacy software development environment
- **(14 SES)** White board and Photos
  - **(RMC)** This could be used to capture (document) software decisions while using Agile methods
- **(14 SES)** Flow Charts
  - **(DTAES 5-5)** State diagrams
    - **(RMC)** a. does not belong under flow charts, and b) is a powerful, under-used, software engineering tool
  - **(DTAES 5-5)** Time slicing charts - illustrations par diagrams
  - **(DTAES 5-5)** Timing diagrams (bus)
    - **(RMC)** Many RTOS development tools/environments provide this type of add-on, e.g. WindRiver's Tornado suite of tools, and 3rd party tools such as those by TimeWiz (I-Logix)
  - **(RMC)** Activity diagrams (UML) are the modern day flow chart, only richer.
- **(CAE)** Emulators
  - **(CAE)** Current systems (F18) are modelized by emulating the assembly code which provide system behavior at the interface level. (Mil-Std-1553, Display etc ..)
  - **(RMC)** are often as complex as the system, and are an overlooked critical element of RTS development. Usually not built to be maintained, and therefore a good candidate for reverse-architecting.
- **(RMC)** Safety Analysis tools (Visio - extension) FTA, ETA
  - **(RMC)** Need to integrate the safety analysis with the software modules (CSC CSCI or whatever they are called). Not only to identify where the safety concerns are, but also where software is used to mitigate or eliminate the risks.
- **(RMC)** System level representation other than Use Cases.
  - **(RMC)** We need a system level representation not only software based models. System interfaces and safety analysis requires this kind of high level thinking
  - **(RMC)** SysML is relatively new, but has good potential
  - **(DTAES 6)** Add consideration for human in then loop (Human Factors/Human System Integration)

- **(Seed)** Techniques
  - **(Seed)** Layout
    - **(Seed)** 2D
      - **(Seed)** Treemap
      - **(Seed)** Call graph
        - **(DTAES 6-6-2)** control coupling among several units need to be visually identified
      - **(Seed)** Flow diagram/chart
        - **(Seed)** Control
        - **(Seed)** Data
      - **(Seed)** Sequence diagram
        - **(CAE)** The modifications in the class diagrams should be reflected in the sequence diagrams. Rational Rose RT didn't do that when I used it.
      - **(Seed)** Information mural
      - **(Seed)** State chart

- **(DTAES 6-6-2)** State transition diagrams: we need tools to visualize potential unknown/undefined states of the software
  + **(WSM)** Ideally, A good MDA tool applies a correct by construction approach that would not let an undesired state exist.

· **(RMC)** Polymorphism visualization - Sylogistic viewer (Western)

· **(DTAES 6-6-2)** Class diagrams: need to identify classes derived by multiple inheritance

· **(14 SES)** Use case diagram

∗ **(Seed)** 3D

· **(RMC)** Google Earth is a good example of a pleasant experience 3-D tool

- **(14 SES)** Also provides a great example of navigation through a model

∗ **(Seed)** Edges - links between the nodes

· **(Seed)** Routing

· **(Seed)** Organic (rounded)

· **(Seed)** Orthogonal (90 degree)

∗ **(Seed)** Lenses

· **(Seed)** Change information

· **(Seed)** Fish-eyes

· **(Seed)** X-Ray

· **(Seed)** Zoom

- **(CAE)** The features (zoom in packages and sequence diagrams, morphing) demonstrated with the tool developed in collaboration with BC University are pretty interesting.

• **(Seed)** Tools

– **(Seed)** Integration

∗ **(DTAES 6)** None used but desperate to get some in place

∗ **(RMC)** Use of simulators (as in flight simulator) to try new man machine interfaces before doing it on the target platform. We used this with success before.

– **(Seed)** SHriMP

∗ **(RMC)** Cool tool, but why the new notation? A UML like notation would do the same job but it is widely accepted.

– **(Seed)** UML

∗ **(Seed)** ArgoUML

∗ **(Seed)** Edgewater?

· **(DTAES 6)** Proposed suite use ECLIPSE

· **(DTAES 6)** We need to explore the effort deployed there as some industries may end up using it

· **(DTAES 6)** Sponsors are USAF, USN, UK MOD and soon Australia and Canada

· **(WSM)** Since its developed by the original developers that did Object Time, it has a big Object Time flavor. The tool is especially designed for avionics development.

· **(WSM)** need to recognize the international interest and $$ being applied to this environment

∗ **(Seed)** Rational

· **(RMC)** Too broad, there are at least two very different UML-based tools under Rational: Rose and RoseRT.

· **(DTAES 6)** Used for some of our contracts but we do not use it within the HQ

· **(RMC)** RoseRT (originally ObjecTime Developer) unfortunately has the same bad UI as Rose (because of ownership), but has a very robust modeling and code-generation engine. It is worth checking out.

· **(RMC)** RoseRT visualization techniques include sequence chart generation from execution traces, animated state charts

        - **(RMC)** We are currently investigating using Quality Architect to perform V&V
        - **(RMC)** RoseRT add-in Rat' Quality Architect does sequence chart comparison for both verification to spec and regression testing, as well as static race condition checking. It also does automated test stub generation to allow for partial system testing.
        - **(RMC)** Tools should not only allow for modeling and documentation of the design of software but support V&V activities. We should be able to execute models against oracles
    - **(Seed)** Visio
        - **(DTAES 6)** commonly used at NDHQ
        - **(RMC)** Good automation API but requires security certificates to distribute code
    - **(Seed)** Visual Paradigm
    - **(14 SES)** ARTiSAN
        - **(14 SES)** Annoying to move through model to a specific element
        - **(14 SES)** Scrolling and zooming are done very poorly
        - **(14 SES)** Automatic generation of dependency diagrams often results in unusable diagram (cluttered)
        - **(14 SES)** Not easily modifiable
    - **(14 SES)** Rapshody
    - **(DTAES 6)** CORE
    - **(CAE)** UML is often used for documentation only, but it should be connected with the real code and requirements, otherwise after a while the diagrams won't be up to date. It should also stay in electronic format. there is no need to have that information on paper.
- **(DTAES 6)** TCP JSA TP 4 (SE) and TP 4-3 (Safety-critical System)
    - **(DTAES 6)** Huge effort in visualization (and also analysis) tools... see what has been done so far. Extensive repertoire of work in High-Integrity S/W (including RT Embedded S/W)
- **(RMC)** Satisfiability tools - bounded requirements
    - **(RMC)** Tools like Alloy (Bounded satisfiability) and Use (OCL) are available in academia to prove assertions and requirements, but are not used outside universities. Investigate?
- **(CAE)** The perfect tools
    - **(CAE)** This tools shall modelized the static structure , dynamics behavior, define the real-time constraints that must apply. Then generate code that with all the parameters on the different targets platform and obviously be certifiable
- **(DTAES 5-5)** Simulation
    - **(DTAES 5-5)** VAPS
    - **(DTAES 5-5)** Matrix X

- **(Seed)** User interface
- **(Seed)** The good
    - **(DTAES 6)** There is an interest to define such capability within ADM(MAT)
    - **(DTAES 6-4C1)** Multiple screen / computer support (e.g. EBOLA)
    - **(14 SES)** No delays when loading any particular diagram (this is probably not possible)
    - **(DTAES 6)** Need to access the critical components without using a random approach as we can only do a partial review (like Quality Assurance).
    - **(14 SES)** The almighty "UNDO" button
    - **(14 SES)** show those elements that a particular user does not have access to change as separate from those that they can change.

    - **(DTAES 6-4C1)** Support for presentations / workgroups (e.g. store analysis not regenerate real time not every time)
    - **(RMC)** Wii - the point being, look to the gaming industry for good UI ideas, both in terms of user inputs and visualization of output.
  - **(Seed)** The bad
    - **(DTAES 6)** Not used at MDHQ but should... but what should we have
    - **(14 SES)** No obvious indication of the dependency of one element to the rest of the model. (i.e. if I was about to delete a function/whatever from the model, knowing that it was used in 42 different sequence diagrams might have caused me a moments delay before I accepted the change)
    - **(RMC)** too many ways to do the same thing, starts out with good intentions, and always leads to bug nightmares, and thus user frustration.
  - **(Seed)** The ugly
    - **(DTAES 6)** Usually limited to amazingly complicated representation when dealing with real avionics software.
    - **(14 SES)** Placement of dependency lines often cross over elements of the diagrams this obscures the ability to interpret the visualization.
    - **(14 SES)** Filtering what is shown on a particular diagram is often painful if not impossible without creating a whole new diagram.

- **(Seed)** Aspects to consider
  - **(Seed)** Cognitive
    - **(DTAES 6)** Must address the Human System Integration aspects... so GUIs are very important
  - **(Seed)** Computation
    - **(DTAES 6)** High Speed, stop computing capability, etc, are needed
  - **(Seed)** Interaction
    - **(DTAES 6)** May need to compare our visualization tools with those used by a prime contractors and his numerous sub-contractors as each have their own development (and then visualization) environments
  - **(Seed)** Output
    - **(DTAES 6)** Need to define the minimum essential sets of information required to perform our due diligence as professional specialists.

- **(Seed)** Essential requirements
  - **(Seed)** Bidirectional Interface
  - **(Seed)** Editable on the Fly
  - **(Seed)** Filter/Highlight
  - **(Seed)** Highly Scalable
  - **(Seed)** Morphing
    - **(RMC)** Adding a situational map in an optional pane would help with localization
  - **(Seed)** Multiple Views
  - **(Seed)** Saving Views/Tags/Bookmarks
  - **(Seed)** Scrolling & Panning
  - **(Seed)** Search & Query
  - **(Seed)** Thumbnail/Overview
  - **(Seed)** Various and Easily Added/Replaced Layouts
  - **(Seed)** Visual Attributes
  - **(Seed)** Zooming
  - **(DTAES 6-4C1)** Changes between versions / releases - present how has the "picture" changed overall.

- **(DTAES 6-4C1)** "Hard" documentation - reports will be needed for the files / signoffs.
  - **(DTAES 6)** From a legal point of view, professional engineers have to sign their formal paper reports...
- **(DTAES 6)** Easy to learn, Easy to use, understandable by managers of technical team

- **(Seed)** Other
  - **(14 SES)** Must have easy access to control the configuration of the elements and diagrams (CM like clear case etc) built into the viewer.

B.4 Fourth Domain: Process Support

- **(Seed)** SV Applies on
  - **(Seed)** Software - what is the percentage of SV applying to software?
    - **(DTAES 5-5)** 30% applied to software
    - **(WSM)** Since the code and the documentation are usually not consistent, more attention should be given to the developed material. In the case of an MDA tool, verifiers can look at the models.
    - **(WSM)** need traceability to requirements. The assumption the software, or the documentation, is correct may be wrong
    - **(DTAES 6-6-2)** Platform (aircraft/ship/ground vehicle) operational/mission requirements, including its interoperability requirements, interface requirements, including MLS protocols, hardware and software requirements: about 50% of errors found in avionics software are due to incomplete, incorrect, ambiguous, inaccurate, inconsistent, untestable/unverifiable requirements. We need to have better tools for requirements verification and validation.
      - **(DTAES 6-4C1)** We need tools for better requirements definition too then!!!
  - **(Seed)** Documentation - what is the percentage of SV applying to documentation?
    - **(DTAES 5-5)** 70% to documentation - requirements, models, test procedures. Reality of now, requirement to reduce time for documentation
  - **(RMC)** Model driven dev. and not design / under MDD the current 0% should go to 100%
    - **(DSS)** Not to use the model as prototype
    - **(RMC)** By this being absent, I suspect it implies that currently 0% is spent on the SV process using the models; whereas in other eng domains, much of the verification is at the model level.
  - **(DTAES 6)** scenarios
    - **(DTAES 6-6-2)** test cases shall be linked to the various user operational scenarios and external conditions, including abnormal ones.
  - **(DTAES 6)** Operational Environments
    - **(DTAES 5-5)** Too little, too late. Most of the time, trivial
      - **(WSM)** please elaborate
      - **(DTAES 5-5)** ...Flt Test
  - **(WSM)** Test and Support Environment
    - **(DTAES 5-5)** Moving target. General lack of rigor in setting up scenarios.
  - **(DTAES 6)** contractual obligation versus proper due diligence
- **(Seed)** SV Activities
  - **(Seed)** Software testing (e.g. Unit tests) does it apply? How?
    - **(DTAES 6)** Make adjustment to its integrity levels (Safety & Security) i.e. not one approach fits all
    - **(DTAES 5-5)** Unit test currently not a requirement for certification. Value lies in troubleshooting. Typically source code driven, not requirement driven.
    - **(RMC)** There are two main views on testing for safety critical systems. First that of reliability of the software (does it meet the user requirements without failures). Second that of safety. The safety program does not have anything to do with meeting user requirements, but that of the certification agency.
  - **(Seed)** Static Analysis (e.g. Metrics to detect rots) does it apply? How?

    - **(DTAES 6)** Rather common and well understood
  - **(Seed)** Dynamic Analysis (e.g. Code purify, coverage) does it apply? How?
    - **(DTAES 6)** Need improvement to get industry to use them
    - **(DTAES 6)** Add RT consideration to address fundamental characteristics expanded
    - **(RMC)** Need to verify system for race conditions, deadlocks, livelocks.
    - **(DTAES 5-5)** Structural coverage analysis based on requirements testing (dynamic) is a compulsory requirement of DO-178B compliance for the top three criticality levels in airworthiness. Based on tools (SCADE, CodeTest, VectorCast, etc).
  - **(Seed)** Code Inspection (e.g. Code review, walkthrough) does it apply? How?
    - **(DTAES 6)** Totally unpractical by hand except for module level, even when it is done for a specific module, we need better methods
    - **(DTAES 5-5)** An inconsistent activity. Checklists and expected artifacts need to be defined otherwise it will be limited to variable spelling errors and indentation observations. At the other end of the spectrum, they are often design reviews which should have occurred before and address low level requirements (e.g. semaphore implementation)
    - **(CAE)** Code inspections and walkthrough are an important part of S/W design process. They can find bugs early in the process then limit the impact of defects found in the last stage. The ratio cost benefit is high.
  - **(Seed)** Document inspection (e.g. Diagram review) does it apply? How?
    - **(DTAES 6)** Most common effort done for ADM(MAT) organization (engineering), Extensive levels of problems with understanding the verification effort.
    - **(DTAES 6-4C1)** SW documentation must be complete & maintainable. Airworthiness requires that code be fixed when critical airworthiness problems attributed to SW arise the SW documentation will be a key part of the Airworthiness process.
    - **(DTAES 5-5)** Useful for comprehension. At the moment, considered insufficient for design assurance and compliance artifact.
  - **(DTAES 6-6-2)** verification of requirements, design, code, object code, test cases, procedures and test results with traceability and coverage analyses.

- **(Seed)** SV What to look for?
  - **(Seed)** Consistency does it apply? How?
    - **(DTAES 6)** Yes , a must
    - **(WSM)** a must for maintenance activities. Different software developers need to be able to understand what has been implemented. Engineer the implementation.
    - **(WSM)** Either extreme discipline is needed which means a lot of overhead or the documentation is automatically kept up-to-date.
  - **(Seed)** Completeness does it apply? How?
    - **(DTAES 6)** Yes, it is contractual and related to the customer operational requirements
    - **(RMC)** What ever is deviated or waived must be approved and documented. For those changes to the spec they must be covered by Engineering Change Proposals
  - **(Seed)** Correctness does it apply? How?
    - **(DTAES 6)** Same as for completeness ... a contractual obligation
  - **(Seed)** Agility for new development does it apply? How?

    - **(DTAES 6-4C1)** Agility important due to the extended life cycles of DND aircraft combined with mission capability growth and the need to adapt with civil CNS/ATM upgrades of the near to log term future (-> 2025) (CNS/ATM COMM NAV SURV / AIR TRAFFIC MGMT).
    - **(RMC)** The agility of a system depends on the strength of the architecture and the level of its erosion during the maintenance part.
  - **(Seed)** Design keeper (e.g. code comprehension, tracking) does it apply? How?
    - **(DTAES 6)** Most platform, "live" for more than 25 years... well after the designer has retired: code comprehension is paramount in order to be responsive with problem correction and inclusion of new features on top of the existing software
    - **(DTAES 6)** Learning curve for new design corrector must be minimum.
    - **(WSM)** software development needs to be engineered. Creativity and innovation, although encouraged, ultimately needs to be engineered to be consistent with the design

- **(Seed)** SV Requirement artifacts
  - **(Seed)** Vision document (e.g. CONOps) if so, which types?
    - **(RMC)** The simplicity and clarity of user stories (stolen from xP) may be an appropriate form at this level
    - **(RMC)** Field level Statement of Deficiencies
    - **(RMC)** Operational Research reports and simulation of capabilities
  - **(Seed)** Glossary / Common vocabulary does it apply?
    - **(RMC)** All requirements should be written in the language of the domain, so yes a glossary is important
    - **(DTAES 6)** Careful with natural language, lack of rigor
    - **(DTAES 6)** Need standardization for sure
    - **(DTAES 6-4C1)** Interpretation of requirements is a constant issue.
  - **(Seed)** Models / Diagrams (e.g. UML Use Cases) if so, which types?
    - **(RMC)** Use cases or similar techniques are applicable, only if used in conjunction with scenario-based testing.
    - **(RMC)** SysML req't diagram - may be worth checking out??
  - **(Seed)** Text (e.g. System Requirement Specifications, Change Requests) if so, which types?
    - **(RMC)** In the domain of Configuration Management and System Engineering - Proof of Compliance
    - **(DTAES 6)** Regretfully, natural language is used: try to have a mix of formal requirements, requirements language, and a bit of natural language especially wrt GUI and user interface.
    - **(14 SES)** Tracking the changes to the requirements can often give an idea of where the proposed architecture might be deficient
  - **(Seed)** Formal methods does it apply? How?
    - **(RMC)** Yes they do and can but depending if the software can be expressed as a set of Specification Functions (provable)
    - **(RMC)** As mentioned earlier, these techniques do not scale, so they can only be useful within a tightly defined critical software component. The dependency analysis tools may be helpful to ensure that components verified by formal methods are not invalidated by dependency relationships.
    - **(DTAES 6)** Very expensive but the standard to aim for wrt very high integrity software (think safety, security such as with partition control of software defined radio).
  - **(Seed)** Tools (e.g. DOORS) if so, which ones?
    - **(DTAES 6)** Mandated by ADM(MAT) for technical requirements definition and traceability... but not well used or worst misuse for contractual requirements management. Ouch!

    - **(CAE)** Every thing should be linked through DOORS (or the equivalent). A requirement should be linked to its design specification, software/hardware implementation specifications, its test case description, the files (or package, or functions) implementing it and testing it. One tool to find and view the whole thing.
        - **(DTAES 6-4C1)** Requirements traceability starts before it s called up for in a specification back to the SOR, CONOPS, STANAGs, civil TC CARs, FARs, AC, ICAO SARPS, etc.
    - **(DTAES 6)** The most important domain contractually but the one recently badly done (botched) by our engineer and pseudo-engineer at NDHQ. Bad start = bad result

- **(Seed)** SV Analysis / Functional specification artifacts
  - **(Seed)** Models / Diagrams (e.g. UML package diagrams, DFD, UI Mock-ups) if so, which types?
    - **(RMC)** System level architecture design. This is important especially from an ILS perspective.
    - **(CAE)** Model-based requirements is a good artifact for SV
    - **(Seed)** Text (e.g. Software Architecture Design, SARAD) if so, which types?
    - **(Seed)** Tools (e.g. UML case tools) if so, which ones?
        - **(RMC)** integral schedulability analysis tools linked tightly to the "live" product, and not some Prel Design review report based upon an as yet realized design

- **(Seed)** SV Design Specification artifacts
  - **(Seed)** Models / Diagrams (e.g. UML class diagrams, DB schemas) if so, which types?
    - **(RMC)** Sequence diagrams (with/without timing) to verify inter-module behaviors
    - **(RMC)** Hierarchical state machines (diagrams) to verify properties of independent component behaviors
    - **(RMC)** Inheritance trees to illustrate this special type of dependency. Other diagrams such as state diagrams must support inherited views; i.e. what behavior is general/specific
  - **(Seed)** Text (e.g. Software Detailed Design) if so, which types?
    - **(14 SES)** Tech. Notes
    - **(14 SES)** ICDs
        - **(RMC)** A weak area of system engineering in general. Are there modeling approaches to help solve this gap? Has SysML addressed this need?
    - **(Seed)** Tools (e.g. UML case tools) if so, which ones?
        - **(DTAES 6)** Most use model-drive design... allowing easier design spec verification!

- **(Seed)** SV Implementation artifacts
  - **(Seed)** Text (e.g. Unit test results, implementation guidelines) if so, which types?
    - **(DTAES 6)** We need guidelines, check list, and standardization for contractual reasons.
  - **(Seed)** Models / Diagrams (e.g. Files Implementation Directory) if so, which types?
    - **(DTAES 6)** This is where a good customer could use benefits having such tools . There has been so many horror story where implementers end up coding on the fly to make it work, resulting in an improper design and tons of artifacts (see older aircraft)
    - **(RMC)** interoperable / interchangeable models must be mandated
  - **(Seed)** Tools if so, which ones?
    - **(DTAES 6)** Needed but little ideas in mind. bonne chance Certified code-generators

- **(Seed)** SV Traceability artifacts
  - **(Seed)** Text (e.g. spreadsheets) if so, which types?
    - **(CAE)** Traceability artifacts must be store in DB or in a tool that support standard query languages. Text-based documents or spreadsheets are not suitable.
    - **(RMC)** Configuration Management and Logistics Support Analysis Record Databases contain most of this information on the changes and tracking. The LSAR contains the reliability and supportability information.
      - **(RMC)** There is a wrong perception that ILS does not impact software. (Integrated Logistic Support)
    - **(RMC)** Requirements Verification Matrix are normally used to track allocated requirements and the tests that are done to close each paragraph of specifications.
      - **(RMC)** In most projects the RVM and the CM database as well as the LSAR are not linked through a common repository, a big problem.
  - **(Seed)** Tools (e.g. DOORS, Requisite PRO) if so, which ones?
    - **(DTAES 6)** There is a push to use DOORS as Requisite Pro is rather expensive (not just $ but with time, etc.)
    - **(CAE)** The entire artifact should be linked to requirements (or other) through DOORS. One tool to see everything related to a requirement.
  - **(RMC)** Traceability documentation should also include proof of compliance (POC) program
  - **(RMC)** Traceability Justification Certificate
    - **(RMC)** A term just invented today to reinforce the point that in nearly all cases the money wasted on maintaining traceability would be better spend on safety/reliability/... analysis and assurance. In other words what are the very specific objectives of requiring traceability, and in each case we should have to certifying up front that we realize that traceability will not guarantee us anything more that every bad requirement has found its way into every aspect of the development, and that every thing we forgot to specify, but which is critical, is absent as decried.
      - **(RMC)** Agreed, but not easily achieved. The answer lies in making sure the dark side is well evaluated and that the safety hazards are well tracked. These should be the ones that should certainly be traced.
    - **(RMC)** I prefer the term proof of compliance.
    - **(DSS)** Validation (guides a verification)

- **(Seed)** Other
  - **(DTAES 6-6-2)** Auto test case generators and script testing: all input conditions, states, and outputs need to be logged in case we need to repeat some testing to evaluate abnormal behaviors of the software.
  - **(DTAES 6)** Need to find a standard to inform the developer about what the customer want. Also applies to Data Items Deliverables.

B.5 Problematic Areas

- **(Seed)** Identify problems
  - **(14 SES)** Extracting an architecture from source code
  - **(RMC)** Integrate Safety Analysis to Design
    - **(RMC)** Safety mitigation and fault tolerance requirements should be attached to code (module, capsule, ...)
    - **(DSS)** Not only safety but also ideally every aspect of code development. There should never be separate expression of the same part (i.e. the documentation and the code), A system should grow from the use cases that are defined to a testable level and the system that is developed is immediately linked to the use cases through model driven architecture. Finally tests/built in test, models and the code are all different aspects of the system addressing a specific part/level and not different representation of the system.
  - **(DTAES 6)** Need a toolbox of approach to V&V our delivered software product
    - **(DTAES 6-6-2)** Regression testing: need tools to identify what test cases to run to verify that the software has not regressed after implementing a change.
    - **(DTAES 6-6-2)** Test coverage: 1- need requirements-based test coverage analysis tools that identify missed requirements during the testing process; 2- need structural coverage analysis tools that can identify what statements/conditions have not been fully exercised during the testing process.
    - **(DTAES 6-6-2)** Coupling analysis: need tools to check for all the interdependencies among all software modules w.r.t. data and control flow, in order to verify the existing level of coupling among them so that the implementation of future changes/new functionality does not become convoluted, hard to implement/maintain and test.
  - **(DTAES 5-5)** Visualize the lack of "robustness" test cases
  - **(DTAES 6)** How to link up the requirements for safety and those for security in our processes
    - **(DTAES 6)** Development level
    - **(DTAES 6)** Integration and testing
    - **(DTAES 6)** V&V Levels
    - **(DTAES 6)** When under its sustaining Engineering phase
    - **(DTAES 6)** Need to find a way to bring forward the problems encountered (create test cases, ..)
  - **(14 SES)** A means of creating tests that are linked to a requirement completely contained in the SDE
  - **(DTAES 6)** Can we automate our Verification and Validation processes
  - **(RMC)** The ever-growing disconnect between the software development industry and the Air Force software community
  - **(DTAES 6)** How and what can be standardized by the customer (Government)?
  - **(WSM)** lack of software development, test and V&V artifacts from third party software developers
  - **(RMC)** Lack of modeling for concurrency constraints
  - **(CAE)** Need a low effort regression testing method that could apply on a legacy code where functional tests are not available
- **(Seed)** Identify tasks or activities
  - **(RMC)** Model concurrency constraints (mutual exclusion, precedence, release times - earliest latest, completion times earliest latest).
  - **(14 SES)** Taking existing code and requirements developed a detailed architecture document defining its behavior.
    - **(14 SES)** Involves the ability to seamlessly transition between the requirements, the model and the code.

  - **(DTAES 5-5)** Derive missing test cases from Structural Coverage results (e.g. code not hit)
  - **(RMC)** Create a scenario that represents a "typical" avionics software development project
  - **(DTAES 6)** Identify the processes to ensure that the impact of Deviations and Waivers are properly assessed in term of impact and of risks.
  - **(RMC)** Link safety analysis
    - **(RMC)** Perform safety analysis (FTA, FMECA, ETA, SHA,...)
    - **(RMC)** Derive hazard resolution
    - **(RMC)** Take hazard resolution and modify create the design for the right resolution
    - **(RMC)** Create test case for each resolution
  - **(RMC)** Create a complete mock-up of the avionics software project as it could be developed with the latest in modeling notations / tools, make it available as a living artifact.
  - **(DTAES 5-5)** Demonstrate/find compliance of partition/protection integrity (e.g. in support of ARINC 653)

- **(Seed)** Identify needs
  - **(DTAES 6)** Need a suite of software given to our S/W Specialists in projects and in WSM organizations in order to fulfill their mandates under the System Engineering construct of ADM(MAT) i.e. M&AS.
  - **(DTAES 6)** Need a suite of RT tools to address our requirements in V&V?
  - **(RMC)** Replaces on-going training, training that always lags the reality of the state of the practice
  - **(WSM)** need a more effective mechanism to capture end user requirements (S/W Maintenance)
    - **(WSM)** need to get end user sign-off on the implementation earlier in the requirement/design process
  - **(14 SES)** Need a SDE capable of managing requirements, design and code for the maintenance of the CP-140 S/W
  - **(WSM)** need to be inherently generating artifacts that will address the airworthiness requirements
  - **(RMC)** Need to integrate safety analysis tools with design tools and tracking tools
  - **(DTAES 6)** New Edgewater RTEdge development Environment:
    - **(DTAES 6)** I need a BETA site to this and challenge it.
    - **(DTAES 6)** Need to work on a cooperative program with TTCP: Help required
    - **(DTAES 6)** Need to address a few real case problems in context of the SoS problems.
  - **(RMC)** Integrate concurrency model with design tool
  - **(DTAES 5-5)** Need incremental formal verification system baselining approach so that everything is not left to the end and bad news unmanageable.
  - **(DTAES 5-5)** Need to be able to exhaustively bind (circonvene) functionality to code to support required due diligence verification (airworthiness). At the moment, this is considered a task than cannot be done completely and exhaustively.

- **(Seed)** Describe traceability from problems to tasks, to needs
  - **(RMC)** Safety
    - **(RMC)** Problem #2 Tasks #6 Needs #7
  - **(DTAES 6)** We get operational needs in of capability deficiencies, than with Modeling and simulation tools (which we have little), we translate this in term of technical & functional requirements (many being derived): than it is given to our contractors for implementation. I need such a tool, such as process, and its trace across.+

- **(DTAES 6)** Link those tech requirements all the way down to the actual product H/W & S/W: How? Need a family of tools
- **(14 SES)** The need to extract architecture: Problem 1, Task 2,Need 5
- **(RMC)** Problem 8 (knowledge-lag) links to tasks 4 (scenario) & 7 (mock-up), and Need:3 (replaces training)
- **(RMC)** Concurrency
  - **(RMC)** Problem #11 tasks 1 and needs 9
- **(Seed)** Other

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Defence R & D Canada – Valcartier
2459 Pie-XI Blvd North, Qubec, QC, Canada

2. SECURITY CLASSIFICATION (overall security classification of the document including special warning terms if applicable).

UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C,R or U) in parentheses after the title).

Getting smarter at managing avionic software: The results of a two-day requirements elicitation workshop with DTAES

4. AUTHORS
(Last name, first name, middle initial. If military, show rank, e.g. Doe, Maj. John E.)

Salois, P. Charland D. Dessureault G. Dussault M. Lizotte F. Michaud D. Ouellet M.

5. DATE OF PUBLICATION (month and year of publication of document)	6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc).	6b. NO. OF REFS (total cited in document)
October 2007	56	22

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered).

External Client Report

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include address).

DTAES 6
National Defense Headquarters – 101 Colonel By Drive, Ottawa, K1A 0K2

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Specify whether project or grant).	9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written).
15AV40	

10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique.)	10b. OTHER DOCUMENT NOs. (Any other numbers which may be assigned this document either by the originator or by the sponsor.)
DRDC Valcartier ECR 2007-097	

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

(X) Unlimited distribution
( ) Defence departments and defence contractors; further distribution only as approved
( ) Defence departments and Canadian defence contractors; further distribution only as approved
( ) Government departments and agencies; further distribution only as approved
( ) Defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution beyond the audience specified in (11) is possible, a wider announcement audience may be selected).

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

Managing avionic software effectively is a challenge with today's tools and resources. The DTAES has recognized this and has approached DRDC Valcartier to address the problem from a technological standpoint. The team at DRDC Valcartier has considerable expertise in software engineering but, unfortunately at this point, very little in avionics. To offset this, a series of measures have been undertaken to ramp up this expertise (e.g. training, production of state-of-the-art reports).

The first of these measures was the organization of a two-day workshop with DTAES and its partners to better define their requirements in avionic software management. This document highlights the results of this workshop, held in December 2006. The workshop used the DSS collaborative laboratory, located at the National Defence Headquarters in Ottawa. This laboratory is built on the MeetingWorks toolset to provide each of the 11 participants with his own computer on which to give feedback on the four pre-identified domains (extraction, analysis, visualization, process support). From the outputs of these domains, the main tasks or problematic areas were identified and prioritized. These will be further investigated later, which could lead to relevant research projects and new engineering efforts within DRDC.

Since DRDC Valcartier is a neophyte in this area, this document will not be an all-encompassing list of all avionic software engineering problems. It rather provides a summary of the most important requirements identified at the workshop. As DRDC Valcartier is currently negotiating with DTAES to improve various other aspects related to the air platforms, the document also provides an opportunity to spotlight potential openings for future collaboration to improve the whole avionic engineering process.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title).

Directorate of Technical Airworthiness and Engineering Support (DTAES), air force, avionics, software, requirements

Defence R&D Canada

Canada's Leader in Defence
and National Security
Science and Technology

R & D pour la défense Canada

Chef de file au Canada en matière
de science et de technologie pour
la défense et la sécurité nationale

WWW.drdc-rddc.gc.ca