Safety-Critical Software Development: DO-178B

Prof. Chris Johnson,

School of Computing Science, University of Glasgow.

johnson@dcs.gla.ac.uk

http://www.dcs.gla.ac.uk/~johnson

Introduction

- Software design by:
  - hazard elimination;
  - hazard reduction;
  - hazard control.

- Software implementation issues:
  - dangerous practices;
  - choice of `safe' languages.

- The DO-178B Case Study.

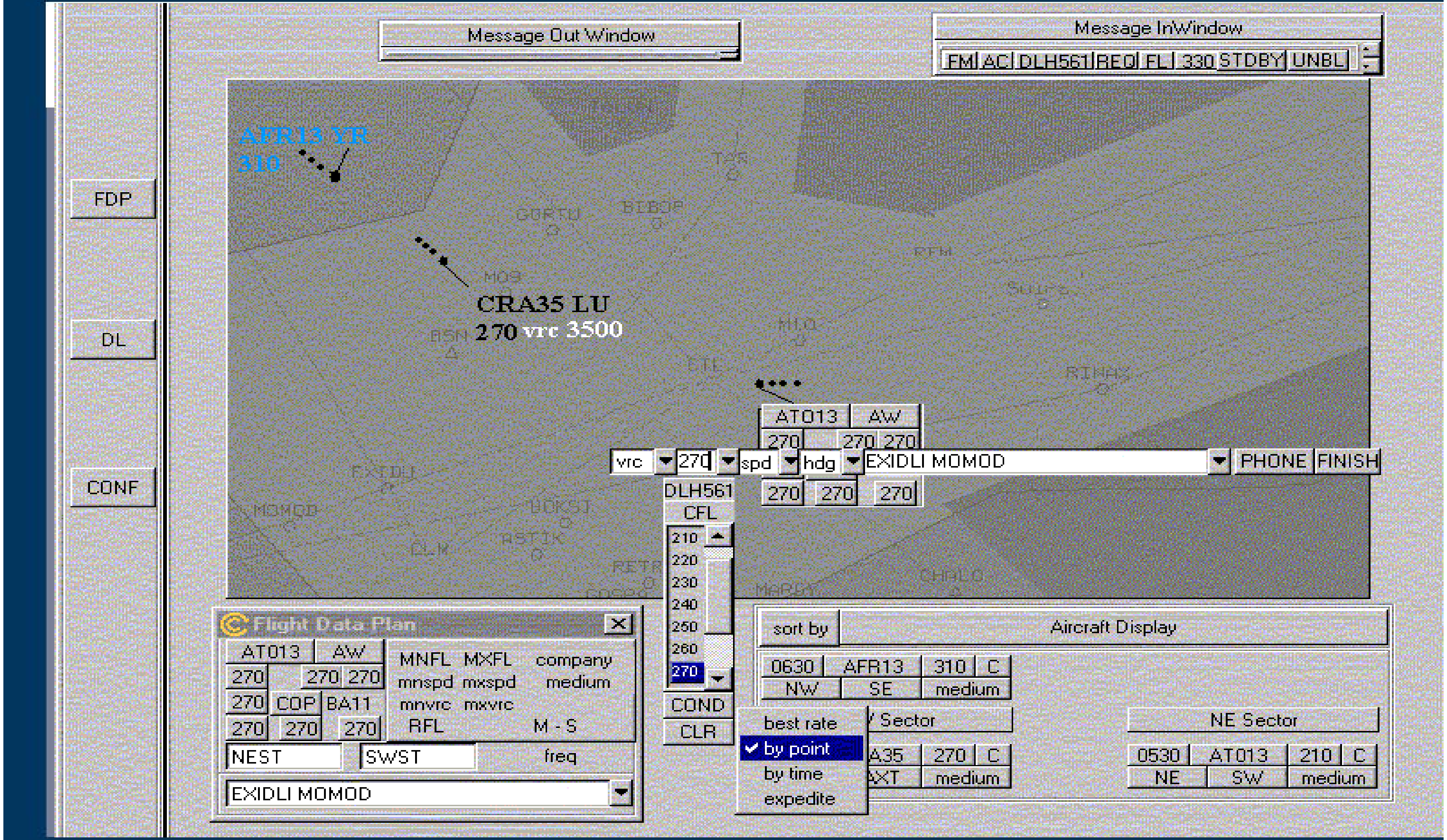
Leveson's Taxonomy of Design Techniques

- Hazard elimination/avoidance.
- Hazard reduction (see 4?).
- Hazard control.
- Hazard minimization (see 2?).

Software Design and Hazard Elimination

- Substitution:
  - hardware interlocks before software.
- Simplification:
  - new software features add complexity.
- Decoupling :
  - computers add common failure point.
- Human Error `Removal' :
  - readability of instruments etc.
- Removal of hazardous materials :
  - eliminate UNUSED code (Ariane 5).

Hazard Elimination: Datalink Example



Message InWindow: FM AC DLH561 REQ FL 330 STDBY UNBL

Software Design and Hazard Reduction

- Design for control:
  - incremental control;
  - intermediate states;
  - decision aids;
  - monitoring.
- Add barriers:
  - hard/software locks;
- Minimise single point failures:
  - increase safety margins;
  - exploit redundancy;
  - allow for recovery.

Hazard Reduction: Interlock Example

This heavy duty solenoid controlled tongue switch controls access to hazardous machines with rundown times.

Olympus withstands the arduous environments associated with the frequent operation of heavy duty access guards.

The unit also self adjusts to tolerate a high degree of guard misalignment.

The stainless steel tongue actuator is self-locking and can only be released after the solenoid receives a signal from the machine control circuit.

This ensures that the machine has completed it's cycle and come to rest before the tongue can be disengaged and machine access obtained.

Software Design and Hazard Control

- Limit exposure.
  - back to ‘normal' fast (exceptions).
- Isolate and contain.
  - Don’t let things get worse...
- Fail-safe.
  - panic shut-downs, watchdog code.

Hazard Control: Watchdog Example

- Hardware or software (beware).
- Check for processor activity:
  - 1. load value into a timer;
  - 2. decrement timer every interval;
  - 3. if value is zero then reboot.
- Processor performs 1 at a frequency
  - great enough to stop 3 being true;
  - unless it has crashed.

Software Design Techniques: Fault Tolerance

- Avoid common mode failures.
- Need for design diversity.
- Same requirements:
  - different programmers?
  - different contractors?
  - homogenous parallel redundancy?
  - microcomputer vs PLC solutions?

Software Design Techniques: Fault Tolerance

- Redundant hardware and software?
- N-version programming:
  - shared requirements;
  - different implementations;
  - voting ensures agreement.
- What about timing differences?
  - what if requirements wrong?
  - costs make N>2 very uncommon;
  - performance costs of voting.
- A340 primary flight controls.

University of Glasgow

Software Design Techniques: Fault Tolerance

- Exception handling mechanisms.
- Use run-time system to detect faults:
  - raise an exception;
  - pass control to appropriate handler;
  - could be on another processor.
- Propagate to outmost scope then fail.
- Ada...

Software Design Techniques: Fault Tolerance

- Recovery blocks:
  - write acceptance tests for modules;
  - if it fails then execute alternative.
- Must be able to restore the state:
  - if failure restore snapshot.
- But failed module may have side-effects?
  - recovery block will be complicated.
- Different from exceptions:
  - Don’t rely on run-time system.

Software Design Techniques: Fault Tolerance

- Control redundancy includes:
  - N-version programming;
  - recovery blocks;
  - exception handling.
- But data redundancy uses extra data
  - to check the validity of results.
- Error correcting/detecting codes.
- Checksum agreements etc.

Software Implementation Issues

- Restrict language subsets.
- Alsys CSMART Ada kernel etc.
- Or just avoid high level languages?
- No task scheduler - bare machine.
- Less scheduling/protection risks

  more maintenance risks;

  less isolation (no modularity?).

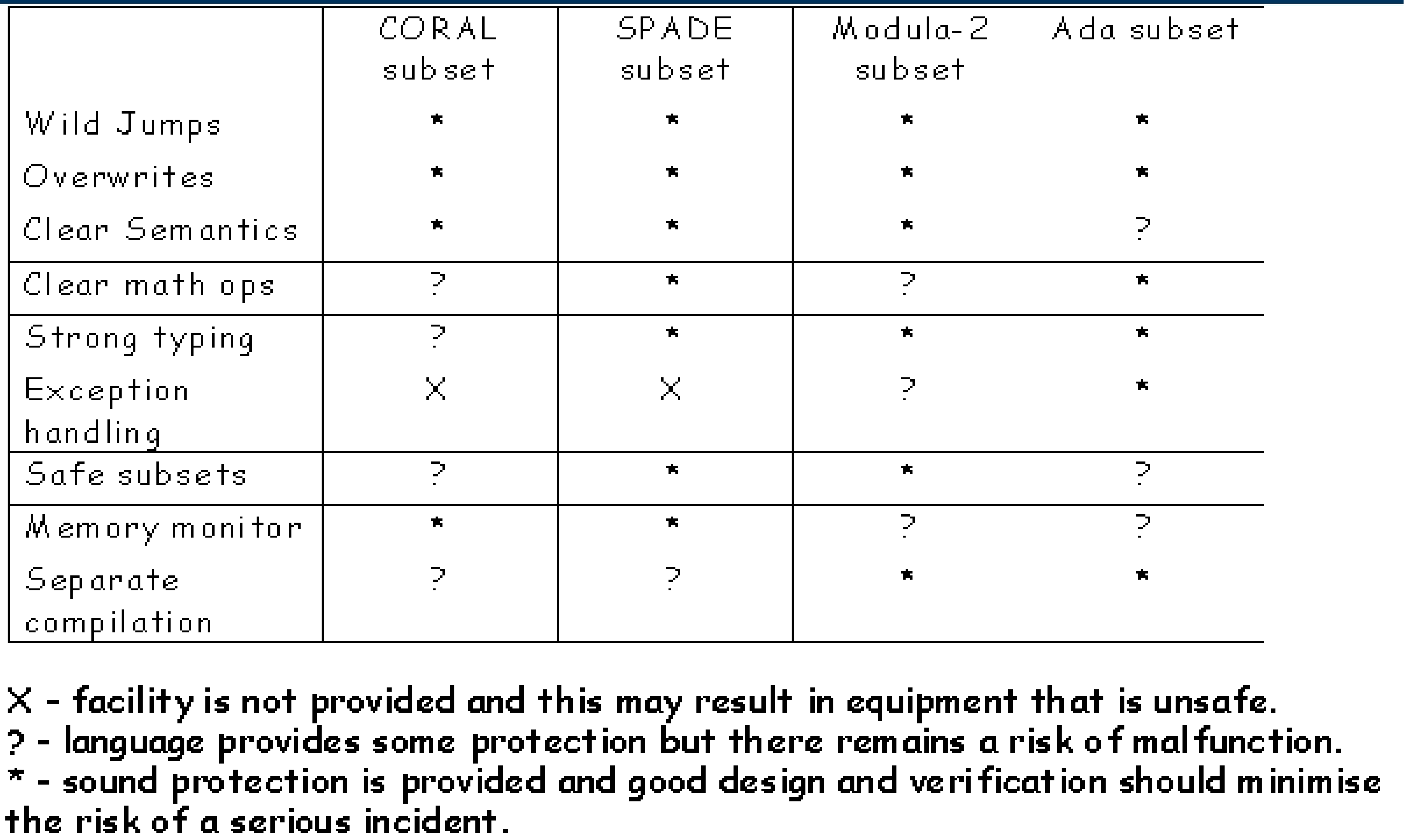
Software Implementation Issues

- Memory jumps:
  - – control jumps to arbitrary location?
- Overwrites:
  - – arbitrary address written to?
- Semantics:
  - – established on target processor?
- Precision:
  - – integer, floating point, operations...

Software Implementation Issues

- Data typing issues:
  - strong typing prevents misuse?
- Exception handling:
  - runtime recovery supported?
- Memory monitoring:
  - guard against memory depletion?
- Separate compilation:
  - type checking across modules etc?

Software Implementation Issues



	CORAL subset	SPADE subset	Modula-2 subset	Ada subset
Wild Jumps	*	*	*	*
Overwrites	*	*	*	*
Clear Semantics	*	*	*	?
Clear math ops	?	*	?	*
Strong typing	?	*	*	*
Exception handling	X	X	?	*
Safe subsets	?	*	*	?
Memory monitor	*	*	?	?
Separate compilation	?	?	*	*

X - facility is not provided and this may result in equipment that is unsafe.
? - language provides some protection but there remains a risk of malfunction.
* - sound protection is provided and good design and verification should minimise the risk of a serious incident.

Software Implementation Issues

- Exoteric languages create training issues?
- General purpose languages & bad habits?
- Ada subset support formal verification?
-
- Meta question:
  - programmer more important than language?
  - or protect all programmers from themselves?

Software Implementation Issues

Here are the results of my recent informal survey of computer languages used in safety-critical embedded systems and other interesting systems. In responses, Ada was by far the most popular language for these systems followed by assembler. There is a list describing 722 Ada projects that is available via ftp from the Ada Information Clearinghouse.

Aerospace:

Boeing:

Mostly Ada with assembler. Also: Fortran, Jovial, C, C++. Onboard fire extinguishers in PLM.

777 seatback entertainment system in C++ with MFC (in development by Microsoft).

757/767: approximately 144 languages used.

747-400: approximately 75 languages used.

777: approximately 35 languages used.

DAINA/Air Force: Aircraft mission manager in Ada.

Chandler Evans: Engine Control System in Ada (386 DOS).

Draper Labs/Army/NASA: Fault tolerant architecture in Ada/VHDL.

Software Implementation Issues

European Space Agency: mandates Ada for mission critical systems.

- ISO (Infrared Space Observatory)
- SOHO (Solar and Heliospheric Observatory)
- Huygens/Cassini (a joint ESA/NASA mission to Saturn)
- Companies involved:
  - British Aerospace (Space Systems) - Bristol, UK
  - Fokker Space Systems - Amsterdam, Holland
  - Matra-Marconi Espace - Toulouse, France
  - Saab - Sweden

Ford Aerospace: Spacecraft in Ada with assembler.

GEOS and INSAT spacecraft in FORTRAN.

(Ford Aerospace is now Space Systems/Loral.)

Hamilton-Standard: (777 air cowling icing protection system in Ada?).

Software Implementation Issues

Honeywell: Aircraft navigation data loader in C.

Intermetrics/Houston: space shuttle cockpit real-time executive in Ada '83 with 80386 assembly

Lockheed Fort Worth: F-22 Advanced Tactical Fighter program in Ada 83

(planning to move to Ada 94) with a very small amount in MIL-STD-1750A assembly. Maintain older safety-critical systems for the F-111 and F-16/F-16 variant airframes primarily done in JOVIAL.

NASA: Space station in Ada. (Sources differed on whether it was Ada only, or Ada with some C and assembler.)

NASA Lewis: March 1994 space shuttle experiment in C++ on 386.

Rockwell Space Systems Div.: Space shuttle in Hal/s and Ada.

Software Implementation Issues

Air Traffic Control:

Hughes: Canadian ATC system in Ada.

Loral FSD: U.S. ATC system in Ada.

Thomson-CSF SDC: French ATC system in Ada.

Land Vehicles:

Delco: Engine controls and ABS in 68C series (Motorola) assembler. C++ used for data acquisition in GM research center. '93+ GM trucks vehicle controllers mostly in Modula-GM (Modula-GM is a variant of Modula-2. A typical 32-bit integrated vehicle controller may control the engine, the transmission, the ABS system, the Heating/AC system, as well as the associated integrated diagnostics and off-board communications systems.)

Ford: Assembler.

General Dynamic Land Systems: M1A2 tank tank software in Ada with time-critical routines in 68xxx assembler. Tank software simulators in C.

Lucas: Many systems in Lucol (Lucas control language). Diesel engine controls in C++. ABS in 68xxx assembler.

Software Implementation Issues

Ships:

Vosper Thornycroft Ltd (UK): navigation control in Ada.

Trains:

CSEE Transports (France): TGV Braking system in Ada (68K). Denver Airport baggage system: This well publicized problem system is written in C++. (A source familiar with the systemsaid the problems were political and managerial, not directly related to C++.)

European Rail: Switching system in Ada.

EuroTunnel: in Ada.

Extension to the London Underground: in Ada.

GEC Alsthom (France): Railway and signal control systems for trains (north lines and Chunnel) in Ada. Subway control systems (Paris, Calcutta, and Cairo).

TGV France: Switching system in Ada.

Union Switch & Signal, Pittsburgh: (Switching system in ?)

Westinghouse Signals Ltd (UK): Railway signalling systems in Ada.

Westinghouse UK: Automatic Train Protection (ATP) in PASCAL.

Software Implementation Issues

Medical:

Baxter: Left Ventricular Heart Assist in C with 6811 assembler.

Coulter Corp.: ONYX hematology analyzer in Ada.

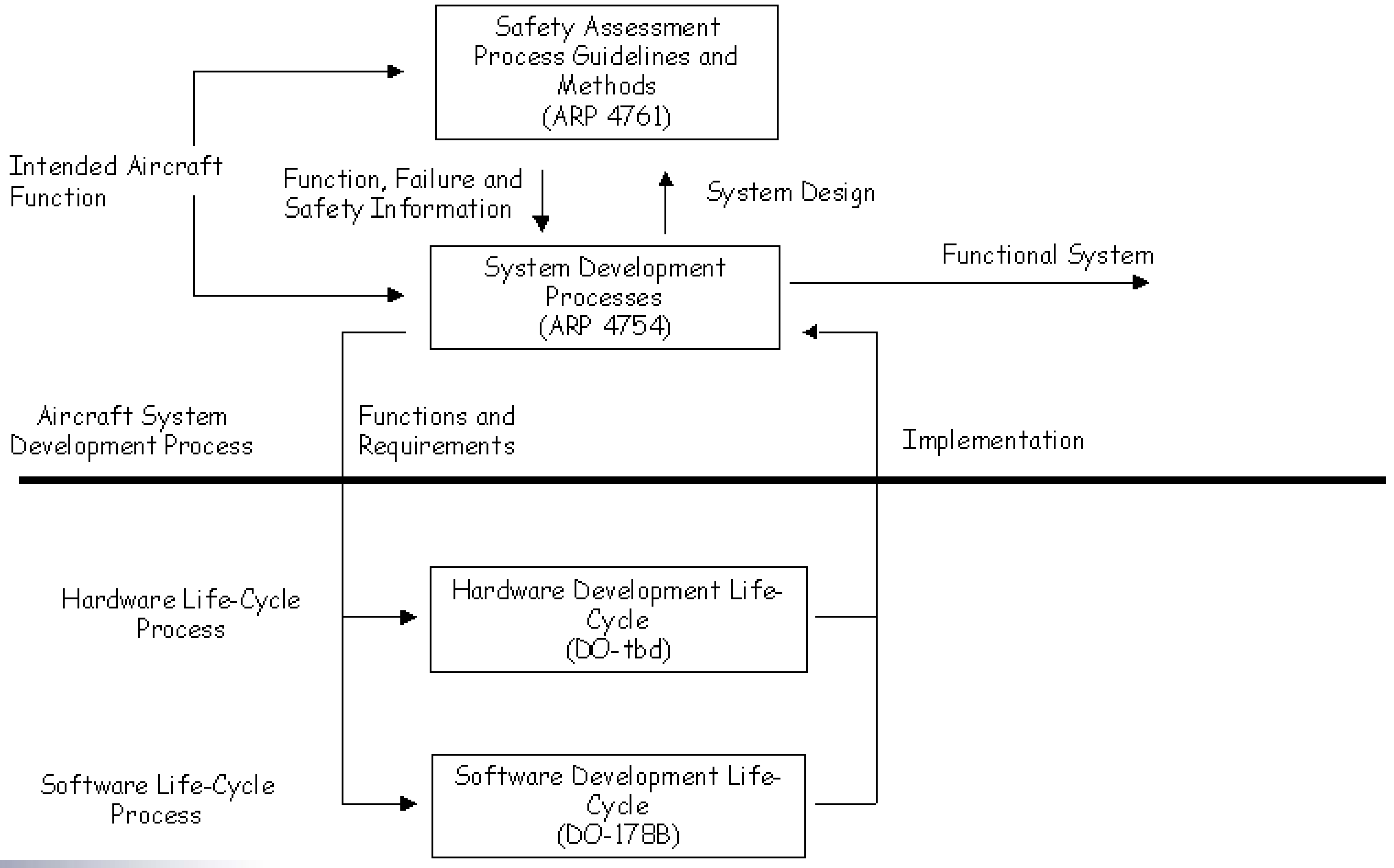
Nuclear Reactors:

Core and shutdown systems in assembler, migrating to Ada

SURVEY METHODOLOGY

I operated under the theory that, with regard to what languages are really in use, the recollections of the engineers themselves are probably the most accurate and open source. In general, I did not have enough sources that I could cross check the information. In cases where I could, the most interesting discrepancy was that companies that thought they had adopted one language as the total solution for all their software designs often had something in assembler or some other language somewhere.

Software Development DO-187B


Safety Assessment Process Guidelines and Methods (ARP 4761)
Intended Aircraft Function
Function, Failure and Safety Information
System Design
System Development Processes (ARP 4754)
Functional System
Aircraft System Development Process
Functions and Requirements
Implementation
Hardware Life-Cycle Process
Hardware Development Life-Cycle (DO-tbd)
Software Life-Cycle Process
Software Development Life-Cycle (DO-178B)

Software Considerations in Airborne Systems and Equipment Certification.

Software Development: DO-178B

- Planning Process:
  - coordinates development activities.
- Software Development Processes:
  - requirements process
  - design process
  - coding process
  - integration process.
- Software Integral Processes:
  - verification process
  - configuration management
  - quality assurance
  - certification liaison.

Software Development: DO-178B

(a) A detailed description of how the software satisfies the specified software high-level requirements, including algorithms, data-structures and how software requirements are allocated to processors and tasks.

(b) The description of the software architecture defining the software structure to implement the requirements.

c) The input/output description, for example, a data dictionary, both internally and externally throughout the software architecture.

(d) The data flow and control flow of the design.

(e) Resource limitations, the strategy for managing each resource and its limitations, the margins and the method for measuring those margins, for example timing and memory.

(f) Scheduling procedures and interprocessor/intertask communication mechanisms, including time-rigid sequencing, pre-emptive scheduling, Ada rendez-vous and interrupts.

Software Development: DO-178B

(g) Design methods and details for their implementation, for example, software data loading, user modifiable software, or multiple-version dissimilar software.

(h) Partitioning methods and means of preventing partitioning breaches.

(i) Descriptions of the software components, whether they are new or previously developed, with reference to the baseline from which they were taken.

(j) Derived requirements from the software design process.

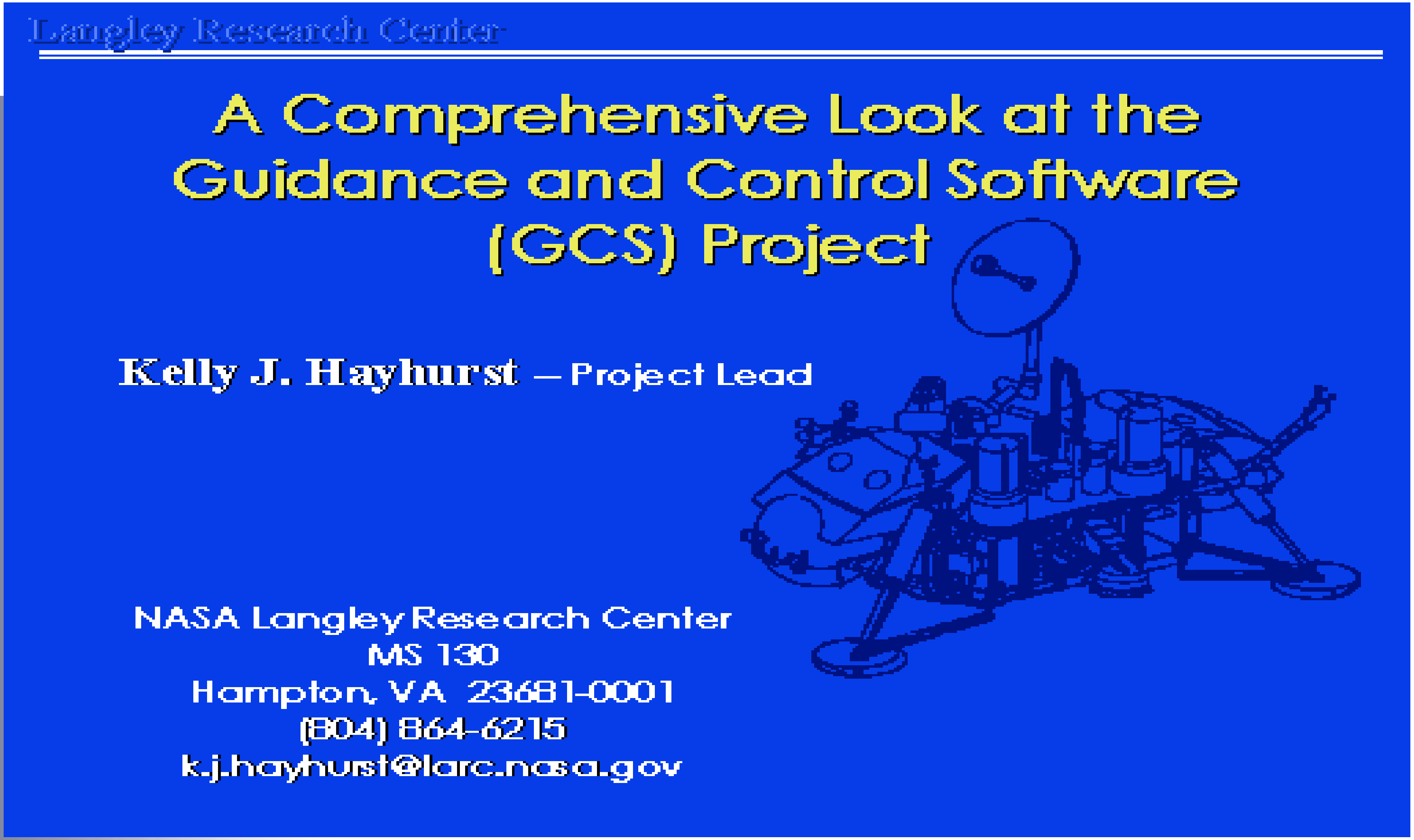
(k) If the system contains deactivated code, a description of the means to ensure that the code cannot be enabled in the target computer.

(l) Rationale for those design decisions that are traceable to safety-related system requirements.

- Deactivated code (k) (see Ariane 5).
- Traceability issues interesting (l).

Software Development: DO-178B - Key Issues

- Traceability and lifecycle focus.
- Designated engineering reps.
- Recommended practices.
- Design verification:
  - formal methods “alternative" only;
  - “inadequate maturity";
  - limited applicability in aviation.
- Design validation:
  - use of independent assessors etc.



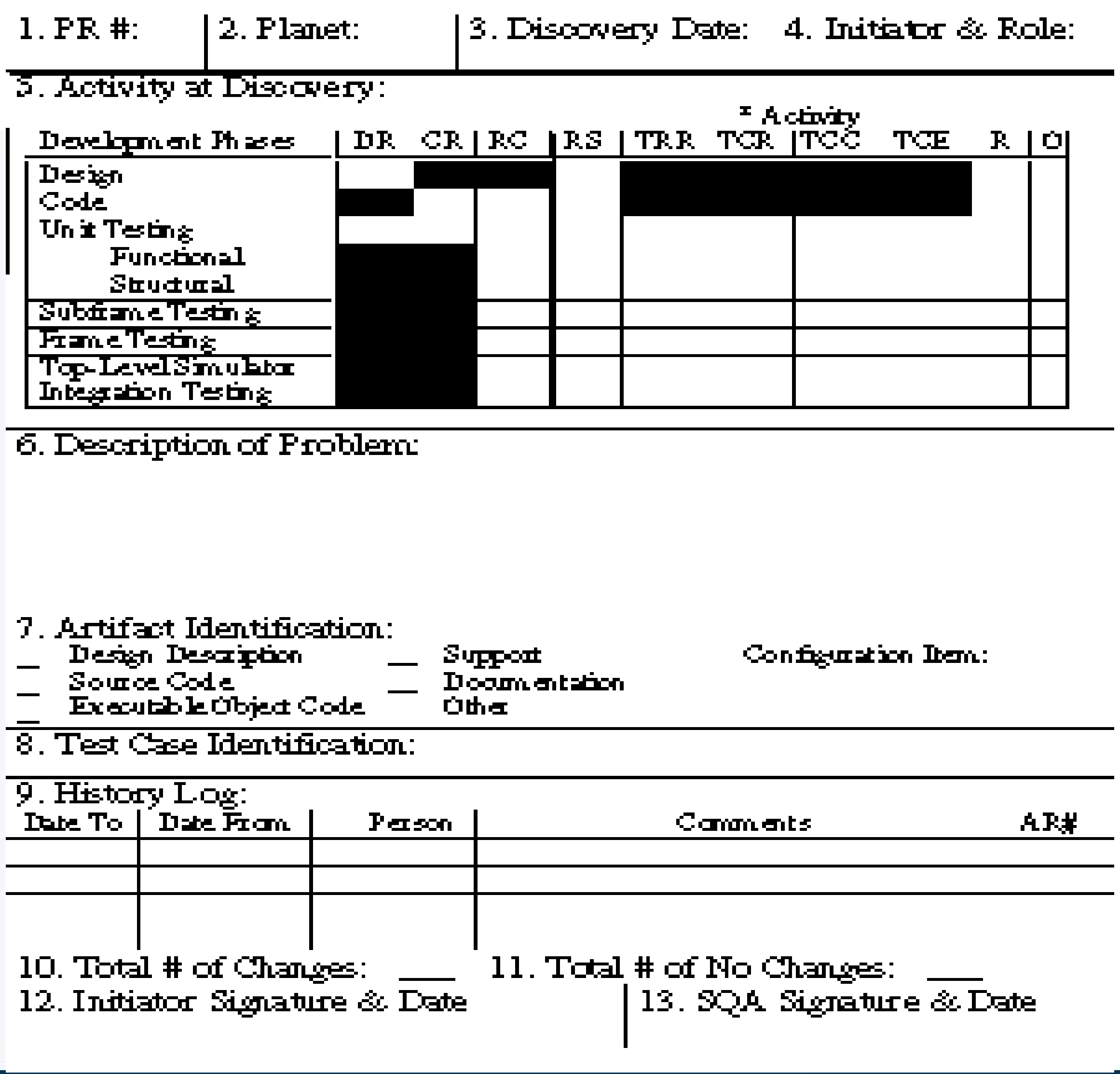
Langley Research Center

A Comprehensive Look at the Guidance and Control Software (GCS) Project

Kelly J. Hayhurst – Project Lead

NASA Langley Research Center
MS 130
Hampton, VA 23681-0001
(804) 864-6215
k.j.hayhurst@larc.nasa.gov

DO-178B - NASA GCS Case Study



1. PR #: | 2. Planet: | 3. Discovery Date: 4. Initiator & Role:

5. Activity at Discovery:

* Activity

Development Phases	DR	CR	RC	RS	TRR	TCR	TCC	TCE	R	O
Design		■	■		■	■	■	■		
Code	■				■	■	■	■		
Unit Testing										
Functional	■	■								
Structural	■	■								
Subframe Testing	■	■								
Frame Testing	■	■								
Top-Level Simulator Integration Testing	■	■								

6. Description of Problem:

7. Artifact Identification:

__ Design Description __ Support Configuration Item:
__ Source Code __ Documentation
__ Executable Object Code Other

8. Test Case Identification:

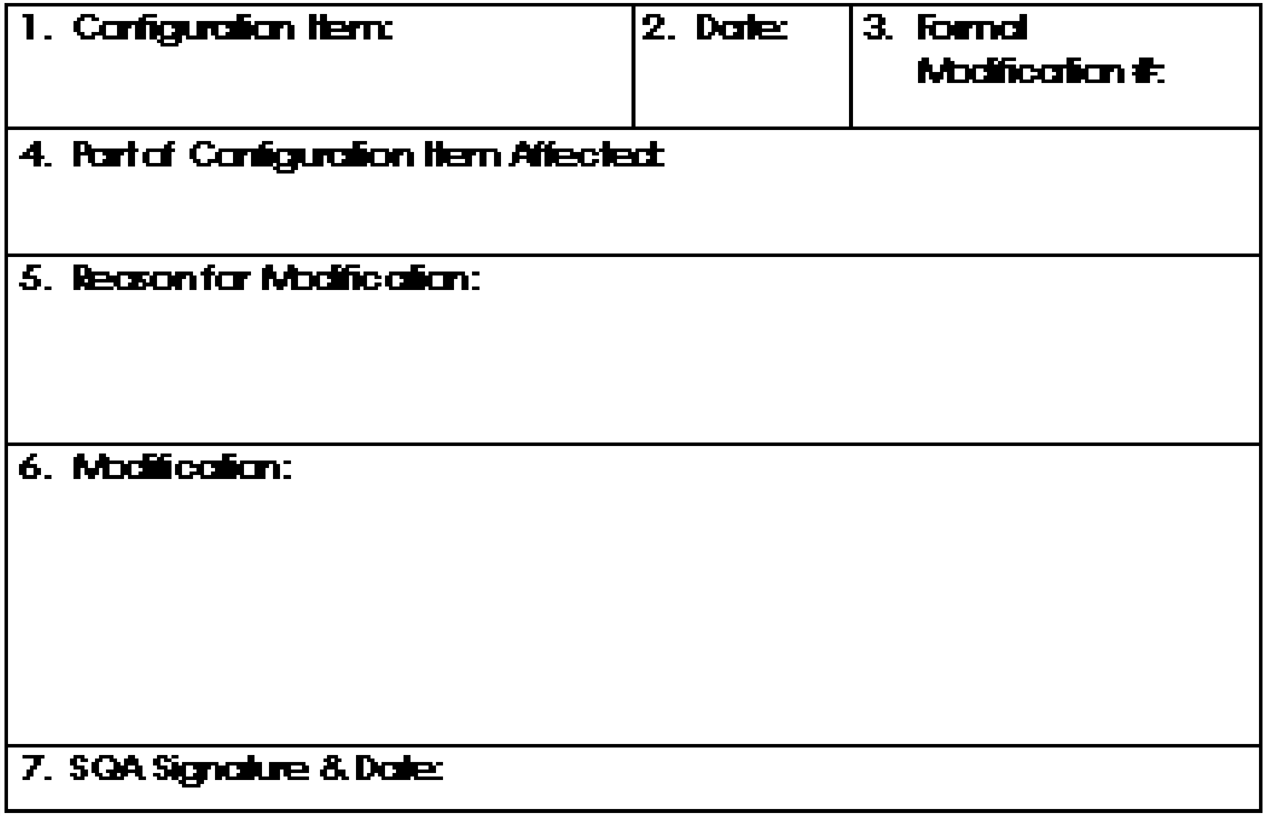
9. History Log:

Date To	Date From	Person	Comments	AR#

10. Total # of Changes: ____ 11. Total # of No Changes: ____

12. Initiator Signature & Date | 13. SQA Signature & Date

DO-178B - NASA GCS Case Study



1. Configuration Item:	2. Date:	3. Formal Modification #:
4. Part of Configuration Item Affected:		
5. Reason for Modification:		
6. Modification:		
7. SQA Signature & Date:		

DO-178B - NASA GCS Case Study

- Project compared:
  - faults found in statistical tests;
  - faults found in 178-B development.
- Main conclusions:
  - such comparisons very difficult;
  - DO-178B hard to implement;
  - lack of materials/examples.

DO-178B Practitioners' View

The difficulties that have been identified are the DO-178 requirements for evidence and rigorous verification... Systematic records of accomplishing each of the objectives and guidance are necessary. A documentation trail must exist demonstrating that the development processes not only were carried out, but also were corrected and updated as necessary during the program life cycle. Each document, review, analysis, and test must have evidence of critique for accuracy and completeness, with criteria that establishes consistency and expected results. This is usually accomplished by a checklist which is archived as part of the program certification records. The degree of this evidence varies only by the safety criticality of the system and its software.

...Engineering has not been schooled or trained to meticulously keep proof of the processes, product, and verification real-time. The engineers have focused on the development of the product, not the delivery. In addition, program durations can be from 10 to 15 years resulting in the software engineers moving on by the time of system delivery. This means that most management and engineers have never been on a project from ``cradle-to-grave.''

DO-178B Practitioners' View

The weakness of commercial practice with DO-178B is the lack of consistent, comprehensive training of the FAA engineers/DERs/foreign agencies affecting:

- the effectiveness of the individual(s) making findings; and,
- the consistency of the interpretations in the findings.

Training programs may be the answer for both the military and commercial environments to avoid the problem of inconsistent interpretation and the results of literal interpretation.

Original source on http://stsc.hill.af.mil/crosstalk/1998/oct/schad.asp

Conclusions

- Software design by:
  - hazard elimination;
  - hazard reduction;
  - hazard control.
- Software implementation issues:
  - dangerous practices;
  - choice of ‘safe' languages.
- The DO-178B Case Study.

Any Questions…