Nationaal Lucht- en Ruimtevaartlaboratorium

National Aerospace Laboratory NLR

NLR-TP-2004-256

Validation, verification and certification of embedded systems

R. Bloomfield (Adelard, United Kingdom)

J. Cazin (ONERA, France)

D. Craigen (ORA, Canada)

N. Juristo (Universidad Politecnica de Madrid, Spain)

E. Kesseler (NLR, the Netherlands)

J. Voas (Cigital, U.S.A.)

This report is based on the Final Report of the NATO Research Task Group IST-027/RTG-009 on the Validation, Verification and Certification of Embedded Systems under the auspices of and approved by the NATO Information Systems Technology Panel of NATO's Research and Technology Organization".
NATO has granted NLR permission to publish this report.

This report may be cited on condition that full credit is given to NLR and the authors.

Customer:	National Aerospace Laboratory NLR
Working Plan number:	AV.1.J.4
Owner:	National Aerospace Laboratory NLR
Division:	Aerospace Vehicles
Distribution:	Unlimited
Classification title:	Unclassified
	June 2004

Approved by author: 2/7/2004	Approved by project manager: w.g	Approved by project managing department: 02/07 2004.

Abstract

This report is the final report resulting from the deliberations of the NATO Research Task Group on the Validation, Verification and Certification of Embedded Systems (IST-027 / RTG-009). The report discusses the important role of embedded systems in both the civil and military contexts. Given the importance, the validation, verification and certification (VV&C) of such systems are of increasing concern. The report discusses the current landscape of VV&C, expected evolution, and also identifies standards of note. The report concludes with various conclusions and recommendations drawn from the task group's deliberations.

Contents

(86 pages in total)

1 Introduction

This final report is the result of deliberations periodically held over a three-year period of the NATO Research Task Group on the Validation, Verification and Certification of Embedded Systems (IST-027/RTG-009).

1.1 Background

The "Terms of Reference" (ToR) for the Research Task Group (RTG), provides the historical background for the group as follows:

"In September 1996, the Flight Vehicle Integration Panel made a proposal to form a Working Group that was approved by RTB in March 1997 under the same title. This group never started due to some difficulties to meet the experts from the appropriate organisms. In addition, RTO was restructured within the same period. The new SCI Panel received the legacy of this activity and decided at its 1998 Fall Business Meeting to ask IST Panel if this activity could be transferred under the IST panel auspices. The idea for a Task Group has been discussed both in IST Panel meetings and via e-mail. In a discussion at its autumn 1999 meeting, a sufficient number of Panel Members supported the formation of a Task Group on the subject for more detailed and substantial work. Embedded systems are ubiquitous in military systems as well as in commercial products used in military applications. From tactical radios to range finders to avionics control units, the Battlespace is filled with digital devices that rely on embedded software for proper operation and functionality. For life-critical and mission-critical systems, extensive testing is required; yet the size and scope of embedded software in today's products precludes exhaustive testing."

The ToR continues by justifying the RTG to NATO as follows:

"Some testing techniques, such as modified condition/decision coverage (MC/DC), have been adopted by government agencies as required methods. Yet there is no body of knowledge that characterizes the types of faults typically found by the various methods and which types of faults might remain undetected. Furthermore, performance testing and stress/load testing are oftentimes overlooked until the product is fielded. Thus, verification and validation are critical efforts for embedded systems. Furthermore, the recent Y2K exercises demonstrated the difficulties many vendors experienced in certifying software. Those Y2K issues related strictly to date and time and yet were quite costly. Expanding the scope to include proper functionality under planned and unplanned usage is important, especially for safety-critical systems; cost-effective and timely methods are needed for certification."

And, finally, the ToR states that the objective of the RTG:

"… is to review the techniques currently used in the software industry to product high quality products; an appropriate number of methods and software life-cycle metrics for systems of relevant complexity should also be examined, particularly those which are supported by fully operational environments. At the conclusion of this activity, the Task Group should deliver a report addressing the following topics:

- Assessment of current technical capabilities and relevance of these techniques and methods to embedded military systems
- Assessment of relative strengths and limitations of these methods
- Assessment of current research trends in testing, formal methods, and requirements traceability
- Specific recommendations for military application of these techniques
- Recommendation for future NATO IST efforts, if relevant, such as Symposium or Workshop."

This report will specifically address each of the above topics in the final section (conclusions and recommendations).

1.2 Meetings

Over the term of the RTG (and the predecessor Exploratory Task Group) nine meetings were held as follows:

- Istanbul, Turkey (October 2000; under the auspices of an Exploratory Task Group);
- Paris, France (March 2001; initial meeting of the RTG);
- Quebec City, Canada (May 2001);
- Warsaw, Poland (October 2001);
- London, United Kingdom (January 2002 [unofficial meeting]);
- Estoril, Portugal (May 2002);
- Cannes, France (October 2002);
- Barcelona, Spain (February 2003);
- Prague, The Czech Republic (October 2003).

These meetings, mostly in conjunction with IST Panel Symposia, provided a means for the key members of the Research Task Group to discuss, in detail, various technical aspects of the VV&C of Embedded Systems. During the Estoril meeting, the IST Panel requested that the RTG consider activities pertaining to NATO's combating terrorism mandate. This resulted in four recommendations being made to the IST Panel of which one was accepted as a follow on activity. The Cannes meeting was held in conjunction with a meeting of the Open Group and, in particular, one of their working groups on Real-time Embedded Systems. This provided an opportunity for interaction between two otherwise mostly independent groups and allowed for some cross-fertilization of ideas.[1] The Barcelona and Prague meetings focused on authoring various aspects of this report and to specifically discuss the conclusions and recommendations.

1.3 Report

This report consists of the following sections:

Introduction	Terms of reference for the RTG.
Embedded Systems Author: Dan Craigen	General discussion; definitions of embedded systems; technology and market trends.
Verification and Validation: Current and Best Practice Author: Natalia Juristo	Definitions of validation, verification, error, fault, and failure; evaluation versus prevention; quality attributes; overview of fault detection techniques.
Verification and Validation: Evolution and Enhancement Author: Jacques Cazin	Avionics systems; General pointers for software partitioning, model checking, static analysis, test case generation from high level specifications, man machine interface and safety analysis.
Standards Author: Ernst Kesseler	Brief introductions to various standards: DO178B/ED12B, DO-278/ED109, IEC61508, AC 120-76, DRD 920, IEC 60880-2, FDA 1252, UK SW01, Common Criteria.
Certification Authors: Robin Bloomfield and Jeff Voas	Discussion on the certification of embedded systems; different kinds of certification; assessment of certification and relevance to NATO requirements; conclusions.
Conclusions and Recommendations Editor: Dan Craigen	Described within the context specified by the Terms of Reference, including various assessments and recommendations.

[1] Two members of the task group, Bloomfield and Craigen also attended the February 2003 Open Group meeting in Burlingame, California.

Appendices	Terms of reference; TAP; Programme of Work; Participants; Combat terrorism recommendations; software certification agencies and services offered; mapping techniques to properties and attributes.

1.4 Acknowledgements

As Chair of the IST-027/RTG-009 Research Task Group on the Validation, Verification and Certification of Embedded Systems, Dan Craigen acknowledges the substantial volunteer efforts put in by five members of the Group: Robin Bloomfield (UK), Jacque Cazin (FR), Ernst Kesseler (NL), Natalia Juristo (SP) and Jeff Voas (US). These members attended almost all RTG meetings and are the material authors of this report. We also acknowledge the efforts of Ann Miller who helped initiate the RTG and was its initial Chair. Dan Craigen edited the report.

2 Embedded Systems

Embedded Systems are ubiquitous. They appear in cell phones, microwave ovens, refrigerators, automobiles and a veritable array of consumer products. Some of these embedded systems have potentially safety or security critical consequences. Embedded systems exist in contexts where failure can be profound. Consider fly by wire, chemical factories, nuclear power plants and even offshore oil wells. Though embedded systems are already ubiquitous, the adoption trajectory of the technology continues unabated. Similar observations exist within the military realm. From smart sensors, software fuses to the evolution of the battle space to being network centric. New projects are almost unfathomable in their scope and extensive usage of embedded systems. For example, the proposed U.S. DDX submarine is effectively a floating software system. It is estimated that the submarine will have 30,000,000,000 lines of code, will make use of 142 programming languages and, obviously, have a huge amount of hardware. How does one even start thinking about how to assess such a system? How do we manage such complexity? Is the technology even there?

Embedded systems are expensive. According to Jack Ganssle,[2] firmware, the software associated with embedded systems, normally cost between US$15-30 per line (from project commencement through to shipping). In the defense realm, with its substantial documentation requirements, costs can range up to US$100 or more. For highly critical applications, of which Ganssle mentions the space shuttle, the cost per line of code approximates US$1,000. Hence,

[2] See http://www.embedded.com/showArticle.jhtml?articleID=17500630, Embedded Systems Programming.

even for a small 5,000-line application, one can easily be talking about projects in the hundreds of thousands of dollars.

The market size for embedded systems is huge. Some estimates place the size of the market at $31,000,000,000, while the general purpose computing market is around $46,500,000,000. The micro-controller market is around $5,000,000,000 with an annual growth rate of around 18%. The growth rate for general-purpose processes is only 10%. With these trends, the embedded systems market will soon be larger than that for general purpose computing. The desktop market is stagnating; the embedded systems market is flourishing.

NATO strategies are continually evolving, but depend substantially on the use of high technology to provide full spectrum dominance. Obviously, given the multinational and interagency components of NATO, and the overall complexity of the battle space, innovation is a vital component of the joint force of the future.[3]

But for all the complexity of defence systems and the ubiquity within the defence milieu, it is a diminishing presence in comparison to the explosive growth of embedded systems in the non-defence realm. A consequence of this diminishing presence means that organizations such as NATO and the U.S. Department of Defense have less impact on evolutionary trends than they have had in the past. The purpose of this report is not to answer the above questions, but to provide the reader with an introduction, overview and assessment of the current and future landscape for the verification, validation and certification of embedded systems. Given the criticality of such systems, how to assess such systems becomes a major concern. Hence, the subject matter of this report.

2.1 What are Embedded Systems?

What are embedded systems? There are numerous definitions, of which a few examples follow:

From http://www.netrino.com/Publications/Glossary/E.html**,** we have

> A combination of computer hardware and software, and perhaps additional mechanical or other parts, designed to perform a dedicated function. In some cases, embedded systems are part of a larger system or product, as in the case of an antilock braking system in a car.

[3] See Joint Vision 2020 (www.dtic.mil/jv2020/jv2020a.pdf) for a sense of how military strategies are expected to evolve.

> In support of the above definition, the following examples of embedded systems were provided: Microwave ovens, cell phones, calculators, digital watches, VCRs, cruise missiles, GPS receivers, heart monitors, laser printers, radar guns, engine controllers, digital cameras, traffic lights, remote controls, bread machines, fax machines, pagers, cash registers, treadmills, gas pumps, credit/debit card readers, thermostats, pacemakers, blood gas monitors, grain analyzers…

From www.y2k.gov/got.html, we have

> … a device that contains computer logic on a chip inside it not independently programmable by the user. Such equipment is electrical or battery powered. The chip controls one or more functions of the equipment, such as remembering how long it has been since the device last received maintenance.

From http://www.embedded.com/wriguide/index.html, we have

> An embedded system is a combination of computer hardware and software, and perhaps additional mechanical or other parts, designed to perform a dedicated function. In some cases, embedded systems are part of larger systems or products, as is the case of an anti-lock braking system.

From http://en.wikipedia.org/wiki/Embedded_system, we have

> An **embedded system** is a special-purpose computer system built into a larger device. An embedded system is typically required to meet very different requirements than a general-purpose personal computer.
>
> Two major areas of differences are cost and power consumption. Since many embedded systems are produced in the tens of thousands to millions of units range, reducing cost is a major concern. Embedded systems often use a (relatively) slow processor and small memory size to minimize costs.
>
> The slowness is not just clock speed. The whole architecture of the computer is often intentionally simplified to lower costs. For example, embedded systems often use peripherals controlled by synchronous serial interfaces, which are ten to hundreds of times slower than comparable peripherals used in PCs.

> Programs on an embedded system often must run with real-time constraints. Usually there is no disk drive, operating system, keyboard or screen.

Sifakis and Bouyssounouse in a paper[4] describing their vision for R&D in embedded systems define embedded systems as:

> Embedded Systems are components integrating software and hardware jointly and specifically designed to provide given functionalities. These components may be used in many different types of applications, including transport (avionics, space, automotive, trains), electrical and electronic appliances (cameras, toys, television, washers, dryers, audio systems, cellular phones), power distribution, factory automation systems, etc.

They go on to note that there is a major trend in information science and technology of the proliferation of embedded systems. It is observed that embedded systems are crucial to, for example, the avionics, automotive, space, consumer electronics, smart card, telecom, and energy distribution sectors. Characteristics of note for embedded systems are ***criticality***, ***reactivity*** and ***autonomy***. Criticality depends upon the consequences of failure. For example, a cell phone failing in Amsterdam may not have severe consequences; however, if your car breaks down in Lapland, the inability to communicate could result in dire consequences. Sifakis and Bouyssounouse note that deviation from the norm can have an impact upon safety, security, mission completion, and on business. In the latter case, they note that business critical systems include services deployed on an embedded infrastructure such as cell phones, power distribution, traffic management, and distributed entertainment. Reactivity pertains to the characteristic that embedded systems must react within the parameters of certain real-time constraints. These real-time constraints can be hard or soft. A hard real-time constraint implies that the expected functionality must be completed within a certain timeframe. Autonomy pertains to an embedded system performing its duties without human input for extended periods of time. Robotic landers on Mars are, by nature, autonomous.

2.2 Trends

In the first part of a Dependable Systems Roadmap,[5] arising from the Accompanying Measure on System Dependability (AMSD) there is an indication of likely trends for embedded systems:

[4] See http://www.softwareresearch.net/site/other/EmSys03/docs/Sifakis.Embedded-EmSys.pdf.

[5] Full report available from http://www.am-sd.org/default/page.gx?_app.page=entity.html&_app.action=entity&_entity.object=KM------000000000000067A&_entity.name=DES_Roadmap-Final_Version.

> "Considering the economies of scale of the enabling semiconductor technologies, the market of safety-critical automotive systems has been identified as the most important market for DES[6] technology, since the automotive market is the largest and the most cost-sensitive market (more than 50 million cars are produced annually). … The envisioned future dominant applications in this growing market (e.g., driveby-wire systems that enhance vehicle stability and thus reduce the number of accidents)... From the dependability technology point-of-view, the leading market is the aerospace market. Although small in comparison to the coming automotive market for dependable embedded systems, it contains mature dependability technology that will be scrutinized in order to determine those aspects that can be transferred to mass-market applications. (Space applications like GALILEO are a unifying link between aerospace and other critical applications like navigation services for air and ground transportation)."

Others have estimated that the market for embedded systems is 100 times the size of that for the desktop market. It is of little surprise that the Information Technology giants (e.g., Microsoft) are moving into the market, especially with the desktop market being now viewed as saturated.

Recently the U.S. National Research Council's Computer Science and Telecommunications Board empanelled a group of researchers under the title "Sufficient Evidence? Building Certifiably Dependable Systems." The scope[7] of the project is described as follows:

> "This project will convene a mixed group of experts to assess current practices for developing and evaluating mission-critical software, with an emphasis on dependability objectives. The committee will address system certification, examining a few different application domains (e.g., medical devices and aviation systems) and their approaches to software evaluation and assurance. This should provide some understanding of what common ground and disparities exist. The discussion will engage members of the fundamental research community, who have been scarce in this arena. It will consider approaches to systematically assessing systems' user interfaces. It will examine potential benefits and costs of improvements in evaluation of dependability as performance dimensions. It will evaluate the extent to which current tools and techniques aid in ensuring and evaluating dependability in software and investigate technology that might support changes in the development and certification process. It will also use the information amassed to develop a research agenda for dependable software system development and certification, factoring in earlier High Confidence

[6] Dependable Embedded Systems.

[7] Quoted from http://www7.nationalacademies.org/cstb/project_dependable.html.

> Software and Systems research planning. It will also investigate ideas for improving the certification processes for dependability-critical software systems."

The output of this group is to be a written report incorporating recommendations, which, we suspect, will be a useful adjunct to the current report.

The mission of The Open Group[8] is

> "… to drive the creation of Boundaryless Information Flow achieved by:
>
> - Working with customers to capture, understand and address current and emerging requirements, establish policies, and share best practices;
> - Working with suppliers, consortia and standards bodies to develop consensus and facilitate interoperability, to evolve and integrate specifications and open source technologies;
> - Offering a comprehensive set of services to enhance the operational efficiency of consortia; and
> - Developing and operating the industry's premier certification service and encouraging procurement of certified products.

Of particular interest to our charter is the Open Group forum on Real-time and Embedded Systems. According the Open Group website the vision of the form is "to grow the marketplace for standardized real-time and embedded systems, through the deployments of standards and associated certification programs." The associated goals of the forum are:

- To be the 'single place' for real-time and embedded systems practitioners to come together for information exchange
- To act as an independent broker to integrate and bring together related Real-Time and Embedded Systems activities
- To develop a series of White Papers identifying the specific needs of Real-Time and Embedded Systems
- To identify priorities for standardization
- To develop test and certification programs to enable the proliferation of standardized real-time and embedded systems.

[8] Quoted from http://www.opengroup.org/overview/index.htm.

2.3 Trends: Conferences/Workshops

In this section, we extract discussion topics occurring in a number of embedded systems conferences and workshops. These topics are suggestive of R&D trends in the space and the current landscape.

Joe Bergmann[9] in his agenda for the February 2004 Real Time and Embedded Systems workshop listed the following topics:

- Focus on Commercial Real-time Environments
- Requirements for Commercial RT Applications to include Avionics, Telematics and Pervasive Computing
- Open Architecture WG
- Commonality of various OA Approaches
- DRM Standards Development
- Security for RT WG
- MILS for Web Services
- PP for Commercial RTOS
- Security for SCADA
- Security for Middleware
- RT Profiles and Certification WG
- Develop RT Certification Profile based on US Navy OACE, FCS SoSCOE, US Army OE
- Safety/Mission Critical Applications
- Specification Development for XML Tags for Traceability
- Safety/Mission Critical RT Java WG
- Ratify Business Plan
- Specification Development SC RT Java
- Mission Critical JSR
- Potential New Items
- High Assurance Systems
- Software Assurance Issues
- Quality of Service Software Issues for RT Environments
- Applicability of OOT in Safety Critical Environments
- Database Requirements for RT
- Procurement issues concerning adherence to Open Systems, Open Standards and Certification

[9] Agenda lodged at http://www.opengroup.org/rtforum/doc.tpl?CALLER=documents.tpl&gdid=4043.

The April 2004 Formal Foundations of Embedded Software and Component-Based Software Architectures (FESCA) Conference[10] encouraged submissions on formal techniques that aid reasoning, analysis and certification of embedded software and component-based software architectures. The call for papers particularly focused on:

- component interoperability,
- contractually used components,
- interface compliancy (interface-to-interface and interface-to implementation),
- compliancy with synchronization constraints,
- temporal properties including liveness and safety,
- security/access rights, further non-functional properties (reliability, performance, timeliness),
- formal methods and dependability,
- formal design methodologies and modeling techniques,
- formal methods for embedded real-time software,
- formal methods and pervasive computing.

The call also was interested in tools and techniques pertaining to the area and explicitly noted::

- logic-based approaches using interactive or automated theorem proving (e.g., B, Z, PVS, Coq),
- concurrency models (e.g., process calculi, refinement calculi, state machines, Petri-nets),
- type theory based reasoning of correctness, component composition frameworks.

In November 2004 the second ACM Conference on Embedded Networked Sensor Systems (SenSys 2004) is scheduled to occur in Baltimore, Maryland, U.S.A. According to the call for papers[11] the purpose and scope of the conference is as follows:

> “SenSys 2004 introduces a high caliber forum for research on systems issues in the emerging area of embedded, networked sensors. These distributed systems of numerous smart sensors and actuators will revolutionize a wide array of application areas by providing an unprecedented density and fidelity of instrumentation. They also present novel systems challenges because of resource constraints, uncertainty, irregularity, and scale. SenSys design issues span multiple disciplines, including wireless communication, networking, operating systems, architecture, low-power circuits,

[10] See http://www.csse.monash.edu.au/fesca/ for further details.

[11] See http://www.cis.ohio-state.edu/sensys04/ for further details.

> distributed algorithms, data processing, scheduling, sensors, energy harvesting, and signal processing. SenSys seeks to provide a cross-disciplinary venue for researchers addressing these networked sensor system design issues."

Topics of note at SenSys 2004 are:

- Network protocols for sensor networks
- Operating system and middleware for sensor networks
- Applications of distributed sensor networks
- Sensor network testbed measurements and benchmarks
- Distributed database processing in sensor networks
- Distributed algorithms for sensor networks
- Novel sensor node hardware and software platforms
- Sensor network planning and deployment
- Energy management in sensor networks
- Adaptive topology management
- In-network processing and aggregation
- Data storage in sensor networks
- Distributed and collaborative signal processing
- Distributed actuation, control, and coordination
- Localization in time and space
- Distributed calibration in sensor networks
- Simulation and optimization tools
- Security
- Fundamental limits and tradeoffs
- Robustness
- Algorithms for self-configuration, maintenance, stabilization
- Programming and validation methodology

Security of embedded systems is to be discussed in the August 2004 workshop on Cryptographic Hardware and Embedded Systems (CHES 2004)[12] to be held in Cambridge, Massachusetts, U.S. As described in the call for papers the focus of the workshop is:

> "… on all aspects of cryptographic hardware and security in embedded systems. The workshop will be a forum of new results from the research community as well as from the industry. Of special interest are contributions that describe new methods for

[12] See http://islab.oregonstate.edu/ches/start.html for further information.

> efficient hardware implementations and high-speed software for embedded systems, e.g., smart cards, microprocessors, DSPs, etc. We hope that the workshop will help to fill the gap between the cryptography research community and the application areas of cryptography."

The topics of CHES 2004 include but are not limited to:

- Computer architectures for public-key and secret-key cryptosystems
- Efficient algorithms for embedded processors
- Reconfigurable computing in cryptography
- Cryptographic processors and co-processors
- Cryptography in wireless applications (mobile phone, LANs, etc.)
- Security in pay-TV systems
- Smart card attacks and architectures
- Tamper resistance on the chip and board level
- True and pseudo random number generators
- Special-purpose hardware for cryptanalysis
- Embedded security
- Device identification

Another major upcoming (as of February 2004) embedded systems conference is EMSOFT 2004,[13] the 4th ACM International Conference on Embedded Software, to be held in Pisa, Italy, in September 2004. The primary objective of this conference is to advance the science, engineering, and technology in embedded software development. The conference website notes the following sample areas of interest:

- System design and integration methodologies
- Programming languages and software engineering
- Operating systems and scheduling
- Hardware/software interfaces and system-level design
- Models of computation and formal methods
- Compilers and execution time analysis
- Networked embedded systems
- Sensor networks and distributed wireless architectures
- Hardware/software co-design and systems-on-chip

[13] See http://www.emsoft.org/ for further information.

- Middleware and QoS management
- Communication protocols and fault tolerance
- Applications on embedded control and multimedia

The 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2004)[14] to be held in Toronto, Canada, May 2004 has three special tracks of note and are described as follows on the conference website:

Real-time Infrastructure and Development: This thrust continues from previous years with focus on embedded and real-time systems that exhibit significant timing constraints. Papers should describe significant contributions to the fundamental infrastructure, system support, or theoretic foundations for real-time computing. Topics include all of those associated with real-time computing platforms and development tools and techniques, such as real-time resource management, real-time operating systems, security, real-time Java, middleware, real-time CORBA, secure real-time systems, support for QoS, novel kernel-level mechanisms, power-aware real-time systems, real-time software component models, model-based development, QoS-aware design, real-time system modeling and analysis, formal methods, scheduling, and performance feedback control.

Real-time control: New this year is an explicit track on the role of control in real-time computing, and the interaction between computing and control systems. Topics cover the use of real-time control methods within infrastructures as well as end-user applications, including but not limited to the interaction of feedback control and scheduling, nonlinear and uncertain real-time systems, modeling and simulation of performance control, computational models and languages for control applications, resource-constrained control or resource-aware control, temporal robustness, robotics, embedded and hybrid systems, and hybrid control.

Embedded Applications: We invite papers on industrial and other real-time and embedded applications. The focus of this track is on contributions associated with systems that are actually deployed in commercial industry, military, or other production environments, including automotive, avionics, telecom, industrial control, aerospace, consumer electronics, and sensors. Papers in this area include, but are not limited to challenges, requirements, model problems, and constraints associated with various application domains, use of real-time and embedded technologies in meeting particular system requirements, performance, scalability, reliability, security, or other assessments of real-time and embedded technologies for particular application

[14] See http://www.cs.virginia.edu/rtas04/ for further information.

domains, mining of architectural and design patterns from applications, and technology transition lessons learned.

These are only a few of the conferences, workshops and symposia directed towards embedded systems subjects. The extent of the meetings pertaining to embedded systems is highly suggestive of the importance and increasing ubiquity of the technology. Many R&D issues remain, even though deployment of embedded systems continues unabated. The Verification, Validation and Certification of Embedded Systems is crucial. The next four sections of this provide an overview and commentary regarding, in order, current and best practice of verification and validation, possible evolution and enhancement of verification and validation, standards, and certification of embedded systems. These sections provide the reader with a solid perspective on the technical landscape for the VV&C of Embedded Systems. We end the report with our conclusions and recommendations.

3 Verification and Validation: Current and Best Practice

In this section we discuss current and best practices in the verification and validation of embedded systems.

3.1 Definitions of Validation and Verification

There are numerous definitions of verification and validation. We present two well-known definitions: An IEEE standard definition and one given by Barry Boehm, an old but very important definition within the software engineering community. According to IEEE,[15] *Verification* can mean the

- Act of reviewing, inspecting, testing, checking, auditing, or otherwise establishing and documenting whether items, processes, services or documents conform to specified requirements.
- Process of determining whether the products of a given phase of the software development life cycle fulfil the requirements established during the previous phase;
- Formal proof of program correctness.

Validation is defined as the evaluation of software at the end of the software development process to ensure compliance with the user requirements. Validation is, therefore, “end-to-end” verification.

[15] IEEE Standard Glossary of Software Engineering Terminology. ANSI IEEE Std, 1983. IEEE Computer Society.

According to Boehm:[16]

- *Verification.* Am I building the system right?
- *Validation.* Am I building the right system?

In the definition for validation, note that it aims to compare a system with its requirements. Therefore, it has to do not only with quality attributes, such as functionality and reliability, but also includes non-functional requirements.

3.2 Definitions of Error, Fault and Failure

A distinction has to be made between terms that have a host of definitions, which can lead to confusion about their meaning. This refers in particular to the difference between error, fault, and failure. IEEE[17] makes the following difference:

- *Error*. People make errors when they reason incorrectly to try to solve a problem.
- *Fault*. An error becomes a fault when it is written (included) in any of the developed software products.
- *Failure*. Failures occur when a software system does not behave as desired (does not do what it should do), which reveals a fault in the software.

The term *defect* will be generally used to refer to any of these concepts.

3.3 Evaluation vs. Prevention

As conceived by Beizer,[18] the actions for producing quality systems can be generically divided into two groups: preventive and corrective. The main difference between the two is that preventive actions define a series of activities to be performed *a priori*, that is, during software construction, to ensure that the product output is not of poor quality, whereas corrective actions take place *a posteriori*, that is, once the software product has been constructed, when its quality is evaluated. In this case, if the value of software output is lower than desired, improvements to the software product (or the software construction process) are proposed. Preventive actions include software development good practice guidelines. Corrective actions encompass the activities proper to software evaluation.

Here, we talk about evaluation of quality attributes, not about prevention (design for, how to build for)

[16] Software Engineering Economics. Prentice Hall. January 1994.

[17] IEEE Standard Glossary of Software Engineering Terminology. ANSI IEEE Std, 1983. IEEE Computer Society.

[18] B. Beizer. Software Testing Techniques. International Thomson Computer Press, second edition, 1990.

As Harrold[19] claims, evaluation is a highly important process, as it is directed at assuring software quality. Evaluation works by studying the software system to gather information about its quality. According to Juristo,[20] evaluation involves examining the developed product (code or other intermediate products) under evaluation and judging whether it meets the desired quality level (if it does, development can continue; otherwise, the evaluated product should be reworked to raise its quality).

3.4 Quality Attributes

Quality is a very broad concept. Therefore, the best way to deal with it is by subdividing it into sub-concepts, which are named quality attributes or criteria. Currently, there is not a universally accepted set of quality attributes for a software system. However, one of the most popular is the one proposed by ISO.[21] Table 1 shows this set.

The rest of this section reviews many of the techniques that evaluate each of the attributes that appear in Table 1. The titles of the following sections take their name from the quality attribute that evaluate and/or the name of the techniques that evaluate them.

3.5 Fault Detection Techniques (Functionality Validation)

The products obtained during software development can be evaluated according to two different strategies: Static analysis and Dynamic analysis (testing).

These two strategies are often confused and both are mistakenly grouped under the term *testing.*[22] However, there are significant differences between static and dynamic analyses of a product. The difference between the two strategies is best understood by making a distinction between the aspects of the software products to be evaluated.

On the one hand, there are static criteria, which are related to visible properties with the system at rest; on the other hand, there are dynamic criteria, which are related to properties that are only manifest when the system is operational. Static analysis can be used to evaluate static criteria and involves examining the product under evaluation at rest. Therefore, static analysis looks for faults. Dynamic analysis, or testing, can be used to evaluate dynamic criteria, which means that it examines the result of operating the system as opposed to the product directly. The dynamic analysis of a software product implies execution, as only by studying the result of this execution

[19] M.J.Harrold. Testing: A roadmap. In Proceedings of the 22nd International Conference on the Future of Software Engineering, pp 63-72, Limerick, Ireland, May 2000. IEEE Computer Society/ACM Press.
[20] N. Juristo. Editor's introduction. Knowledge Based Systems, 11(2):77-85, October 1998.
[ISO, 1998] ISO Standard ISO9126-1998.
[21] ISO Standard ISO9126-1998.
[22] A. Bertolino. Guide to the knowledge area of software testing. Software Engineering Body of Knowledge, February 2001.

is it possible to decide whether or not (or to what extent) the quality levels set for the dynamic aspects judged are met. Therefore, dynamic analysis looks for failures. The task performed by a system during execution for the purposes of running a dynamic analysis is what is known as workload or simply test cases.

At present, the executable software product *par excellence* is code (although there are some executable specification and design languages, their use is not very widespread). Therefore, any product obtained during development (including code) can be evaluated by means of static analysis. However, dynamic analysis (or testing) almost exclusively evaluates code.

CHARACTERISTICS AND SUBCHARACTERISTICS		**DESCRIPTION**
Functionality		**Characteristics relating to achievement of the basic purpose for which the software is being engineered**
	Suitability	The presence and appropriateness of a set of functions for specified tasks
	Accuracy	The provision of right or agreed results or effects
	Interoperability	Software's ability to interact with specified systems
	Security	Ability to prevent unauthorized access, whether accidental or deliberate, to programs and data
	Compliance	Adherence to application-related standards, conventions, regulations in laws and protocols.
Reliability		**Characteristics relating to capability of software to maintain its level of performance under stated conditions for a stated period of time**
	Maturity	Attributes of software that bear on the frequency of failure by faults in software
	Fault tolerance	Ability to maintain a specified level of performance in cases of software faults or unexpected inputs
	Recoverabiltiy	Capability and effort needed to re-establish level of performance and recover affected data after possible failure
	Compliance	Adherence to application-related standards, conventions, regulations in laws and protocols
Usability		**Characteristics relating to the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users**
	Understandability	The effort required for a user to recognize the logical concept and its applicability
	Learnability	The effort required for a user to learn its application, operation, input and output
	Operability	The ease of operation and control by users
	Attractiveness	The capability of the software to be attractive to the user
	Compliance	Adherence to application-related standards, conventions, regulations in laws and protocols

Efficiency		**Characteristics related to the relationship between the level of performance of the software and the amount of resources used, under stated conditions**
	Time behavior	The speed of response and processing times and throughput rates in performing its function
	Resource utilization	The amount of resources used and the duration of such use in performing its function
	Compliance	Adherence to application-related standards, conventions, regulations in laws and protocols
Maintainability		**Characteristics related to the effort needed to make modifications, including corrections, improvements or adaptation of software to changes in environment, requirements and functional specifications**
	Analyzability	The effort needed for diagnosis of deficiencies or causes of failures, or for identification parts to be modified
	Changeability	The effort needed for modification fault removal or for environmental change
	Stability	The risk of unexpected effect of modifications
	Testability	The effort needed for validating the modified software
	Compliance	Adherence to application-related standards, conventions, regulations in laws and protocols
Portability		**Characteristics related to the ability to transfer the software from one organization or hardware or software environment to another**
	Adaptability	The opportunity for its adaptation to different specified environments
	Installability	The effort needed to install the software in a specified environment
	Co-existence	The capability of a software product to co-exist with other independent software in common environment
	Replaceability	The opportunity and effort of using it in the place of other software in a particular environment
	Compliance	Adherence to application-related standards, conventions, regulations in laws and protocols

Table 1. Software Quality Characteristics and Attributes—ISO 9126-1998 View

3.5.1 Testing Techniques

There is sometimes some confusion as to the testing process and it is mistakenly thought that a testing technique outputs faults, when it really outputs test cases. On other occasions, one finds that the testing process in routine practice boils down to executing the software as many times as deemed necessary (or as often as there is time to), and testing with randomly selected inputs. Neither the generated test cases, nor the stop testing condition are ever reported. Neither is a proper estimate of the time and resources that will be necessary to run the tests ever made.

However, in order to properly test a system it is necessary to follow a well-defined process. Here, we show the one contemplated by ESA: PSS-06.[23]

1. **Test planning**. The purpose of this activity is to identify the characteristics or quality attributes of the software to be tested, the rigour with which they are to be tested, any characteristics not to be tested, and the software elements that are to be tested. Also resources (personnel, tools, etc.) will be allocated to each of the tasks to be performed, and the testing techniques to be used will be selected.
2. **Test design**. For each test identified in the above activity, the selected testing techniques will be applied and the generated test cases will be identified.
3. **Test case specification**. For each generated test case, the elements it affects must be identified, and both the inputs required to execute the test case and the expected outputs and resources (software or hardware) required to run the test cases must be specified. Also the test cases will be ordered depending on how they are to be executed.
4. **Test procedure definition**. For each generated test case, the steps to be taken to correctly execute each generated test case must be specified.
5. **Test procedure execution**. Each generated test case will be executed according to the associated test procedure defined.
6. **Analysis of the results**. For each test procedure executed, the version of the software elements involved in the test, as well as the characteristics of the environment in which the test was run must be referenced. A description will be given of test execution (author, test starting and finishing time), as well as the results of the test (success or failure).

The testing process (the six aforementioned steps) is considered as one of the most costly development processes,[24] sometimes exceeding fifty per cent of total development costs. Bearing in mind the testing process defined above, it can be found that one of the many factors that influence the cost of testing is the number of designed test cases (step 2). This means that the more test cases are generated, the longer it will take to specify and execute the tests and, therefore, the more costly they will be. It is well known that exhaustive testing, or the execution of all the possible combinations of input values is infeasible. For example, it will take as many test cases as there are possible combinations for summing any two numbers to test a program as simple as a sum of two numbers and this would consume precious time. Therefore, the tests will be run on a relatively small set[25] previously selected from all the possible cases. The choice of test cases is of utmost importance, not only because the resulting set has to be reduced to the minimum, but also because this set must allow the software system to comprehensively

[23] ESA software engineering standards PSS-05-0. Software Engineering Standards ISSN 0379 4059, European Space Agency (ESA), ESTEC, Noordwijk, The Netherlands, February 1991. ESA Board for Software Standardisation and Control (BBSC).

[24] B. Beizer. Software Testing Techniques. International Thomson Computer Press, second edition, 1990.

[25] B. Beizer. Software Testing Techniques. International Thomson Computer Press, second edition, 1990.

demonstrate the aspect under evaluation. If this aspect cannot be sufficiently exhibited using the selected set of test cases, the tests will have failed to achieve their objective, because they will not have examined the software properly.
Testing techniques actually focus on the choice of a set of test cases from all the possible test cases, an activity that conforms to the third point of the process explained above. This means that the problem of choosing test cases for a software project is actually the selection of one or more testing techniques.

We now discuss various testing techniques.

The techniques for functionality validation that will be presented here are:

- Black–box testing (also called Functional testing)
- Equivalence classes and input partition testing.
- Boundary value analysis.
- Error guessing.
- White-box testing (also called Structure based testing)
- Control flow analysis.
- Data flow analysis.
- Cause consequence diagram.
- Other techniques

3.5.1.1 Black Box Testing (Functional Testing)

Equivalence Classes and Input Partition Testing: The idea behind equivalence partitioning is that testing is more effective if the tests are distributed over all the different possibilities rather than testing similar possibilities. Input values which are treated the same way by the software can be regarded as being in some sense equivalent (they probably follow the same execution path for example). To choose good test cases, it is better to chose one from each equivalence partition than to choose all from one partition. This basic philosophy can be applied at different levels of testing; it is probably better to exercise each menu option once than to leave some unused and concentrate all testing on only a few. (This must be taken with a “grain of salt,” because in some cases it may be better to do the reverse, for example, if only a few options are critical to users and testing time is severely limited).

For example, if the numbers 0 to 99 are valid, then any of those 99 numbers should be handled in exactly the same way by the software. If the program works correctly for the value of 7, for example, it will probably also work correctly for 2, 35, 50, etc. All of the values 0 to 99 are within the same equivalence class or equivalence partition.

The values less than 0 form another equivalence partition: that of the invalid values below the valid partition. The values 100 and above form a third invalid partition above the valid partition. Choosing the three test values of –25, 50 and 150 is more likely to reveal an error than the values of 49, 50, and 51. (But see Boundary Value Analysis below).
This is an over-simplification to illustrate the basic principle, and makes the assumption that an input variable is independent of other variables, which is rarely the case in practice. Equivalence partitions may also be dependent on previous input or data values stored, or on some other context factor. For example, an order quantity of 0 to 99 might be valid in isolation, but if it were referring to items packed in sets of six, not all values would be valid.

Boundary Value Analysis: The values that lie at the edge of an equivalence partition are called "boundary values." One of the most common types of coding error is for the boundary of the equivalence partition to be "out by one." In addition to choosing one value from each partition, chose values on each side of each partition boundary. The boundaries in the example are where the invalid partitions meet the valid partition: one boundary is between –1 and 0, the other is between 99 and 100. Those boundary values, -1, 0, 99 and 100 can determine whether the processing is correct for the boundaries. If the loop control is out by one, then for example, 99 might be regarded as invalid, or 100 as valid.

Again, this is an oversimplification, assuming that a variable is linearly independent from any other variables. Domain Testing is the formalization of this type of partition strategy.

Boundary Value Analysis is more prescriptive than Equivalence Partitioning, since any value in a partition is regarded as equally valid as a test case for equivalence partitioning, but the particular values for the boundaries are determined by those boundary values.

Equivalence partitions and boundary conditions can exist on output values, as well as on input values. Test cases should be devised to attempt to achieve both valid and invalid output values, and to test at the output partition boundaries.

There can also be "hidden" boundaries, which should also be tested for, since thy often result in unexpected failures. Hidden boundaries are related to internal structure, for example, the disc-block transfer size. If 256 characters fill a block, test with 256 and 257 characters.

For real rather than integer values, the boundary values would include the boundary value and small ranges on each side of that value. For integer values, the boundary can be thought of as being between two values, or on one of the values. Some variants of boundary value analysis would result in three values, on, above and below the boundary.

Error Guessing: A good tester often has a feel for where the tricky errors might be lurking, and can guess at the best-input cases to find those errors. This intuition is based on previous experience or on common assumptions made by developers. Examples of good test cases are: division by zero, an empty file, record, or field, negative numbers, alphabetic character for numeric field, decimal point, embedded comma, minimum and maximum sizes. The test suite should contain any such test cases which are suggested; if test cases are considered unnecessary because they would "never happen," that is likely to be a good test case.

3.5.1.2 White (Glass) Box Testing (Structure-Based Testing)

Control Flow Analysis: Control flow techniques are based on the internal control of the software. Although the function of the software is used to determine what the predicted outcome is for a given input, the choice of test cases is driven by looking inside the "box," and choosing test items to exercise the required parts of the control. Control Flow testing is most effectively used as a way to evaluate the effectiveness of functional testing, by analysing the structural coverage gaps have been identified, then control flow techniques are used to derive test cases to exercise the parts of the control which have not yet been exercised.

Data Flow Analysis: Data flow testing is concerned with what happens to data elements in a program. A variable is "defined" when a value is stored into it, such as in an assignment or a read or input operation. (Note that "defined" is not the same as "declared" when the variable name, without a value, is made known to the compiler, although some declarations are also definitions.) A value is "used" when the value determines some processing, such as in a condition of an IF statement, or when it is printed as output. The tests are derived so that adequate definition-use associations are exercised by test cases, or existing test cases are evaluated to determine the coverage achieved in terms of data flow elements. See data flow testing and also Beizer.[24]

Cause Consequence Diagrams: Cause-effect graphing is a systematic way of organising combinations of inputs (causes) and outcomes (effects) into test cases.

There are different variations of this technique; some involve constructing a "logic diagram" using logical operators such as "And" and "Or," and then constructing a decision table from the diagram.[26] Other variants omit the logic diagram and go directly to the decision table.

The cause-effect analysis involves examining a specification, optionally constructing a Boolean network to express the effects as related to the combinations of causes, eliminating any

[26] G.J.Myers. The Art of Software Testing. Wiley Interscience, 1979.

redundant combinations, and constructing a decision table summarizing the conditions and actions. Test cases are then derived from the decision table, and can also take boundary conditions into account. Eliminating different inputs, which result in the same output, and then combining for “do not care” entries can often simplify the cause-effect graph.

3.5.2 Other Techniques

There are other techniques, which are now a matter of research. These are:

- Error seeding.
- Performance modelling.
- Prototyping/animation.
- Process Simulation.
- Techniques for certain kinds of systems (GUI’s, COTS, real-time systems, etc.).

3.6 Static Techniques

Static techniques for functionality validation include:

- Inspections.
- Walk-throughs.
- Reviews.
- Audits.
- Other techniques.

3.6.1 Inspections

Inspections are one of the most effective, yet lowest-technology, quality assurance techniques that can be applied to software development at all stages of the life cycle. Inspections have the obvious benefit of locating errors in code or other documentation. Fagan also viewed them as a contributor to disciplined development. By requiring inspections at various points in the development life cycle, software engineers not only improved the quality of the work products involved, but also gained valuable data on defect injection and resolution.

The completeness of a software product is most often determined by testing. Inspections can also contribute to the determination of when a product is ready for shipment. In Fagan’s original data, design and code inspections located 82% of all errors in a specific product. Acceptance tests and actual use by a customer for six months revealed zero defects. If an organization maintains records of inspection results and all other defect identification methods, it can determine the average percentage of errors located by the process and thus indicate when a product is ready to move on to the next step in development.

Inspections are a team activity. Four people can accomplish most inspections: the producer of the item to be inspected, a moderator to facilitate the process, and two technically competent inspectors. One of the persons also acts as a recorder. The inspection is preceded by a period of preparation, usually not less than one hour, by each member of the group. The inspection itself (a meeting) is usually limited to two hours, since longer duration tends to reduce the efficiency of the team. After the inspection comes a period of follow-up, beginning with a report and ending with the closure of open items such as the disposition of major defects.

3.6.2 Walkthroughs

A Walkthrough is a static analysis technique in which a designer or programmer leads members of the development team and other interested parties through a segment of documentation or code, and the participants ask questions and make comments about possible errors, violation of development standards, and other problems. A Walkthrough is conducted, in many cases, with participants who are non-peers.

A principal distinguishing factor of a walkthrough and a formal inspection is that the inspection is always a peer review. A moderator leads it whereas a reader or presenter will lead the walkthrough and the collection of anomalies is carefully structured to capture statistical evidence of the effort.

3.6.3 Reviews

A review is a process or meeting during which a work product, or a set of work products, is presented to project personnel, managers, users, customers, or other interested parties for comment or approval.

A formal review is typically one that is required by a contract commitment, which is usually invoked through the application of a standard. The implication is that it is a contractual milestone witnessed by the customer, and denotes the completion of certain activities such as detailed design. Another view is that there is certain obligation that needs to be met or satisfied.

Informal reviews are those that are held which are not contractually required, such as technical interchange meetings, where the purpose is one of providing a periodic interchange of information.

3.6.4 Audits

An audit is much like an inspection, but tends to be more defined according to purpose and is usually conducted using defined criteria. An example is the physical configuration audit, In the general form an audit is an independent examination of a work product or set of work products to assess compliance with specifications, standards, contractual agreements, or other criteria.

3.6.5 Other Techniques

Other techniques include:

- Sneak circuit analysis.
- Symbolic execution.

3.7 Failure Detection Techniques (Reliability Validation)

There are numerous techniques related to reliability validation including:

- Control flow monitoring:
- Watchdog.
- Dynamic logic.
- Signature analysis/memorised executed cases.
- Exception handling.
- Run time automatic detection
- Data monitoring:
- Assertions/plausibility checks.
- Integrity checks/detection codes.
- Defensive programming.

3.8 Usability Validation

Techniques for usability validation include:

- Expert reviews
- Heuristic evaluation.
- Inspections (conformance, consistency, and collaborative usability).
- Walkthroughs (pluralistic, cognitive).
- Usability tests
- Thinking aloud.
- Measured performance
- Field usability testing
- Follow-up studies of installed system
- Questionnaires and surveys.
- Interviews.
- Focus groups.
- Logging actual use.
- User feedback.
- Other techniques

3.8.1 Expert Reviews

Heuristic Evaluation: Heuristic evaluation is performed to identify the usability problems of a system, so that they can be attended to as part of an iterative design process. It involves having a small set of evaluators examine the interaction design and judge its compliance with recognised usability principles (the heuristics). It can be used as a complement to usability testing with users, since it usually reveals different kinds of usability problems than usability testing.

Inspections (conformance, consistency, and collaborative usability): Inspections have a long history in software development. The goal of all inspections is to find defects. Usability inspections are aimed at identifying usability defects. The object of inspection may be a finished product, a design or a prototype. Usability inspections refer to systematic processes for inspection, as opposed to heuristic evaluation, which is a less formal usability assessment technique. When different stakeholders perform the inspection in a collaborative effort, it is called collaborative usability inspection. In this case, the review process is a team effort that includes software developers, end users, application or domain experts and usability specialists, collaborating to perform a thorough and efficient inspection. There are two variants of inspection, which have a specific focus: consistency inspections and conformance inspections.

Walkthroughs (pluralistic, cognitive): A pluralistic usability walkthrough is a collaborative process involving users, developers and other stakeholders, where all participants are expected to play the role of users. The participants evaluate the interaction design by trying to perform a given task, and they stop at each step to have a group discussion about its usability. The goal of the technique is coordinated empathies to help developers to put themselves in the shoes of users.

Cognitive walkthrough is a technique for evaluating user interfaces by analysing the mental processes required of users. Like heuristic evaluation, the results are based on the judgement of the cognitive walkthrough analyst, instead of on results with real users. The difference is that it is focused on specific tasks, instead of on assessing the usability of the system as a whole.

In a cognitive walkthrough, correct sequences of actions are analysed, asking if users will actually follow them. The cognitive walkthrough analyst identifies problems by tracing the likely mental processes of a hypothetical user. The analysis considers matters like user background knowledge that influence mental processes but are not part of the user interface. The technique aims to identify likely usability problems in the user interface and to suggest reasons for these problems. Cognitive walkthroughs were developed for systems that can be learned by exploratory browsing, but they are useful even for systems that require substantial learning.

3.8.2 Usability Tests

Thinking Aloud (constructive interaction, retrospective testing, critical incident taking, and coaching method): Thinking aloud is a technique for performing usability tests with users. The evaluator asks participants (users) to talk out loud while working during a usability testing session, indicating what they are trying to do, or why they are having a problem, what they expected to happen that did not, what they wished had happened, and so on. By verbalising their thoughts, test participants enable the developer to understand how they view the system, and this helps to identify major user misconceptions.

The strength of thinking out loud is on qualitative data and not on performance measures. The idea is to get the user's impression while using the system to avoid later rationalisations. The aim of this kind of testing is to detect the parts of the dialogue that are more problematic from a usability point of view, along with the real causes of the problems.

There are some variants of this technique: constructive interaction, retrospective testing, critical incident taking and coaching method.

Measured Performance (Usability Specifications Measurement): Performance measurement through usability testing is used for assessing whether usability goals set in usability specifications have been met. It can be used as well for comparisons with competing products. Performance is measured by having a group of users perform a predefined set of test tasks while collecting time and error data. When the test is performed in a special room prepared for usability testing, it is called laboratory usability testing. A laboratory is usually composed of two rooms separated by a one-way mirror:

- The evaluation room where participants carry out the tests and the main evaluator gives instructions; and
- The control room, where additional evaluators and other members of the development team can observe the test, without disturbing the test participant.

Usual equipment for a usability laboratory includes a video camera to record the screen, another one for recording the participant, tools for software logging and monitors to show in the control room what is happening in the evaluation room.

The opposite to laboratory testing is field testing, where the system is taken to the user environment instead of taking the participant to the system, and the usability test is performed in the user organization.

Field Usability Testing (direct observation, beta-testing, video recording, verbal protocol): Individual users may be directly observed doing specially devised tasks or doing their normal work, with the observer making notes about interesting behaviour or recording their performance in some way, such as timing sequences of actions. This is called direct observation. When the observation takes place in the user organization, it is called field usability evaluation.

Video recording can be either an alternative to direct observation or a backup for what happens in a usability evaluation session. For field usability evaluation, audio recording can be useful as well to record the user comments.

3.8.3 Follow-up Studies of Installed Systems

Questionnaires and Surveys: Questionnaires are used to determine a user's subjective satisfaction with the system. Measuring user satisfaction provides a subjective (but, nevertheless, quantitative) usability metric for the related usability attribute. Some usability specifications will be related to user satisfaction, and questionnaires are the way to check whether the level specified for this attribute has been reached. Questionnaires are usually administered to usability test participants after the test has taken place, so they can give their opinion about specific parts of the user interface and about the overall system.

When questionnaires are distributed to a lot of users, they are called surveys. While questionnaires issued to usability test participants may contain questions about specific parts that have been used in the test, surveys usually gather opinions on more generic issues. Additional information that is usually collected has to do with individual user characteristics, such as background (age, gender, education), experience with computers, familiarity with specific features (virtual reality, macros, shortcuts), and so on.

Interviews (structured, flexible): Interviews involve having an interviewer read questions to a respondent and writing down the responses. After usability testing the evaluator may interview the participant to get the user's subjective opinion, instead of letting the participant fill in a written questionnaire. Interviews are more flexible, since the evaluator may ask follow-up questions that not were in the script.

Focus Groups: Focus groups are a somewhat informal technique that can be used to assess user needs and feelings, after the system has been in use for some time. Focus groups often bring out spontaneous reactions and ideas from users through the interaction between the participants and have the major advantage of allowing some group dynamics and organizational issues. Focus groups are especially appropriate for limited user communities.

Logging Actual Use (time-stamped keystrokes and interaction logging): Logging involves having the computer automatically collect statistics about the detailed use of the system. It is mainly used to collect information about the field use of a system after release, but it can also be used as a supplementary method during usability testing to collect more detailed data. It is unobtrusive, so it does not interfere with the user's normal usage of the system.

When the actual use of the system is logged, this information is particularly useful because it shows how users perform their actual work and because it is relatively easy to automatically collect data from a large number of users working under different circumstances. Typically, an interface log will contain statistics about the frequency with which each user has used each feature in the system, and the frequency with which various events of interest (like, for example, error messages) have occurred.

When undertaking a major redesign for a system that has been in use, it is very helpful to rely on interaction log information to guide the redesign effort.

User Feedback (online or telephone consultants, online bulletin board or newsgroups, user newsletters and conferences): Once the system is in use, the user community is the best source for information on the usability weaknesses of the system. User Feedback can be collected by giving them access to special electronic mail addresses, network newsgroups, or bulletin boards. Users can send their complaints and requests for change or improvement.

3.8.4 Other Techniques

Other techniques include:

- Experimental Tests
- Predictive Metrics
- Acceptance Tests
- Cooperative Evaluation
- Participative Evaluation

3.9 Efficiency Validation

Techniques related to efficiency validation include:

- Avalanche/stress testing.
- Performance requirements.
- Other techniques.

3.9.1 Avalanche/Stress Testing

Stress testing or load testing is done to check for problems that may occur when the system is subjected to input values or environment conditions that exceed their expected range during normal operation of the system. Stress testing of the software that controls a telephone switch capable of handling up to 1024 calls, for example, might include gradual increase of the number of calls up to and beyond 1024, as well as simultaneous initiation of 1024 calls.

3.9.2 Performance Requirements

This is done to determine if the system meets or exceeds specified performance criteria, such as response-time to user commands and system throughput. Performance testing of the telephone switch would measure the time required to establish a new connection under various load conditions of the switch, including no existing calls, 100 existing calls, and 1023 existing calls.

3.9.3 Other Techniques

Other technique that has not been taken into account here is the Response Timings and Memory Constraints.

3.10 Maintainability Validation

Techniques related to maintainability validation include:

- Regression testing.
- Other techniques.

3.10.1 Regression Testing

When existing (tested) software is changed, additional testing is required in tow areas: testing for expected changes and testing for unexpected changes. The altered or additional software should be tested for expected changes, to check that the new expected results are correct; this is depth testing. Software, which has not been changed, should be tested for unexpected changes or side effects; this testing is called regression testing.[27] This is because one must be concerned with ensuring that the software has not regressed (gone backwards) to an erroneous state (linguistically it should be called “anti-gression” testing, but it is known as regression testing).

3.10.2 Other Techniques

Other techniques include:

- Impact Analysis
- Reverify Changed Software Modules

[27] G.J.Myers. The Art of Software Testing. Wiley Interscience, 1979.

- Reverify Affected Software Modules
- Revalidate Complete System
- Software Configuration Management
- Data Recording and Analysis

3.11 Portability Validation[28]

Portability requirements may require the software to be run in a variety of environments. Attempts should be made to verify portability by running a representative selection of system tests in all the required environments. If this is not possible, indirect techniques may be attempted. For example if a program is supposed to run on two different platforms, a programming language standard (e.g., ANSI C) might be specified and a static analyser tool used to check conformance to the standard. Successfully executing the program on one platform and passing the static analysis checks might be adequate proof that the software will run on the other platform.

4 Verification and Validation: Evolution and Enhancement

In this section we discuss the possible evolution of verification and validation technologies as they pertain to embedded systems.

In the area of embedded systems, more specifically in the context of avionics systems, the main constraint is safety. Safety is traditionally achieved by combining architectural design with approaches allowing the verification of functional, safety, and real time performance of software and systems in such a way that the behavior of the system is always predictable.

In the last few years, avionics systems have evolved in a significant way as it may be observed at the two major aeronautics manufacturer Boeing and Airbus: the trend is to replace classical approaches to avionics by Integrated Modular Avionics (IMA).[29] This evolution may be schematically described as follows:

- In classical avionics, embedded systems are made of one computer for each embedded function (E.g., flight command system, autopilot, or flight management system). Whenever

[28] Description from ESA software engineering standards PSS-05-0. Software Engineering Standards ISSN 0379 4059, European Space Agency (ESA), ESTEC, Noordwijk, The Netherlands, February 1991. ESA Board for Software Standardisation and Control (BBSC).

[29] Useful links on Integrated Modular Avionics may be found at http://www-users.cs.york.ac.uk/~philippa/IMA_LINKS.html.

two functions had to communicate with each other, a physical connection is drawn between their support computers.

- In IMA, a set of computers provides generic resources to a set of functions. They are linked to each other by means of an avionics embedded network. In this context, functions now share computer resources (e.g., computational or memory) and communication resources.

This new orientation gives rise to a set of new complex problems which were solved *de facto* by the architectural choice of classical avionics:

- Two functions sharing a given resource must be isolated from each other, especially if these two functions do not have the same level of criticality.
- End-to-end performance of function connection must be predictable; this has consequences both on the use of shared computer resources and on the communication medium.

So major problems in the area of Verification and Validation of embedded systems are concerned with:

- Middleware, the central point being that the operating system allow functions to share the resources.
- Application software, which are to be run on a specific computer resource.
- Communication systems.
- Man-Machine Interfaces, which have specific concerns. Indeed, the use of digital display, and the complexity and variety of information the pilot is to be provided with, demand special care.

Finally IMA opens the path toward modular verification and validation and the way to take it into account in a certification process.[30]

Whatever the properties to be controlled may be, there are two classes of approach to verification and validation. They are either

- Based on human operation and expertise and may be qualified rather empirical even though they are computer aided.
- Automated and may be based on formal notation tools and methods to produce, or to check the properties of the embedded software.

[30] For further discussion, see http://techreports.larc.nasa.gov/ltrs/PDF/2002/cr/NASA-2002-cr212130.pdf.

An Architecture Description Language (AADL), devoted to Avionics systems, is currently in the process of standardization under the SAE authority. Although such a language is rather syntactic it provides an open framework, which is a relevant place to incorporate formal approaches.[31]

4.1 Software Partitioning

The problem of partitioning in software applications is a central one in the context of an IMA system. An exhaustive overview has been made some years ago by John Rushby,[32] in which formal approaches used in the area of computer security are shown to apply also to this specific area. Its is worth noting that the results take into account both space and time partitioning and are therefore useful for functional and performance analysis.

4.2 Model checking

Model checking has proved to be a relevant approach to master functional and behavioral properties of embedded software and numerous success stories are documented. Nevertheless, the approach does suffer from the well-known problem of combinatory explosion. So tentative work is being conducted to reduce this drawback by combining model checking with other formal approaches. In the area of avionics embedded systems, the European project Safeair plans to use bounded model checking, symbolic execution and proved automated code generation and to map it onto an IMA architecture in an Avionics Systems Development Environment.[33]

4.3 Static Analysis

Static analysis by means of abstract interpretation has been extensively explored in the European Daedalus project. The aim of the project is to use this technique to check various properties of embedded software. Among them,

- Precision control of floating point arithmetic.
- Timing analysis of parallel asynchronous programs.
- Modular analysis of program composition.

The results apply to avionics software and give excellent perspectives for functional properties checking, but also for time and performance properties.[34]

[31] For further information, see http://www.cse.wustl.edu/~cdgill/RTAS03/published/SAEAADLRTASv1.pdf.

[32] For further information, see http://www.csl.sri.com/~rushby/papers/faaversion.pdf for general results and http://www.tttech.com/technology/docs/formal_verifications/Rushby_2001-03-Partitioning_in_TTA.pdf for results pertaining to time triggered architectures.

[33] For further information, see http://www.safeair.org/.

[34] For further information, see, for example, http://www.di.ens.fr/~cousot/projects/DAEDALUS/synthetic_summary/index.shtml.

4.4 Test Case Generation from High Level Specifications

Model-checking based approaches are a way to produce test cases in a fairly simple way:

To test a property **P** *on a system amounts to check that* **not (P)** *always holds.*

as it is not the case a model-checker should provide a counter-example which may be interpreted both as a test sequence for *P* and the oracle value. On this topic, see Paul Blake's publication list.[35] This approach is implemented in tools like GATEL[36] based on the synchronous language LUSTRE. As far as this language is underlying the SCADE workshop used in Airbus, to avionics domain is likely a relevant application. Once again, this supposes a means to master complexity and combinatorial explosion. A possible way is to combine results with a statistical model-checking approach.

4.5 Man Machine Interface

Man Machine Interface Verification an Validation give rise to a specific class of problems in the context of avionics systems as it implies to take into account man in the loop perception and interaction. A variety of properties have to be checked in relation with both ergonomic and functional aspects of the interaction. As a consequence, a usual practice in the definition of MCDU (Multipurpose Control and Display Units)[37] is to implement mock-ups from the system specification and to make an intensive use of simulation before coding. This time consuming activity, which amounts to testing applied at the mock-up level to validate the specification, does not alleviate to test the real system after it has been implemented. Provided that a formal specification equivalent to the mock-up implementation (with respect to interaction properties) be at disposal, two improvements may be proposed. The first one consists in replacing testing by proofs on the specification and may be envisaged for safety and liveness functional properties. The second one is to use the specification, which has been validated by simulation thanks to the mock-up, for deriving test cases to be applied to the real system. Both of them may rely on model checking (as introduced for test case generation). Concerning the provision of a formal specification, abstract interpretation techniques may be applied to the source code of the mock-up to extract the part related to interaction.[38]

4.6 Safety Analysis

Safety analysis has been traditionally addressed in the context of aeronautics systems by techniques based on Fault Tree Analysis (FTA) and Failure Mode Effect Analysis (FMEA).

[35] Available at http://hissa.nist.gov/~black/Papers/.

[36] For further information, see http://www-drt.cea.fr/Pages/List/lse/LSL/Gatel/index.html.

[37] For further information, see http://flyreal.free.fr/English/pfd_en.htm and http://www.cas.honeywell.com/ats/products/fms.cfm.

[38] For further information, see http://www.cert.fr/en/dtim/publis/CVSI/1999/dsvis99.ps.

These techniques were complemented, at the level of system analysis, by means of testing and simulation tools to support FTA.[39] This industrial practice may obtain benefit from using formal methods to link FTA with system specification.[40]

5 Standards

In air transport, the safety rules and regulations have evolved separately for the various services. Consequently for any integrated system many, not necessarily harmonized standards apply. All discussed standards share the notion that software has to be classified according to the system hazards the software failure would cause or contribute to. This information is obtained from Functional Hazard Analysis (FHA) plus (Preliminary) System Safety Assessment (P)SSA. Based on this information, the software providing the service will be classified. For each class a number of standard specific requirements have to be satisfied. Usually, an independent authority checks compliance with the requirements and approves complying products as fit for use.

This section will concentrate on the software part, in compliance with the Terms of Reference for the NATO task group. Some verification and validation aspects of the following standards are discussed:

- DO-178B/ED12B, from the airborne avionics domain.[41]
- DO-278/ED109, for CNS/ATM (or ground plus satellite) systems.[42]
- IEC 61508, for general application, being mandated for a/o European railway industry. It evolved from process industry.[43]
- As yet unnamed standard from the Eurocontrol EATMP software task force. This standard has the targets the same area as DO-278/ED109.
- Electronic Flight Bag AC 120-76, aiming at COTS devices in aircraft with many differing applications.
- DRD 920, an EGNOS program specific standard for positioning systems.[44]
- IEC 60880-2 for software in the nuclear industry.[45]

[39] A useful overview can be found at http://www.faultree.com/.

[40] See http://www.cert.fr/esacs/documentation.html for more details.

[41] DO-178B / ED12B, *Software Considerations in Airborne Systems and Equipment Certification*, RTCA & EUROCAE, 1992.

[42] DO-278/ED109 *Guidelines for the communication, navigation surveillance, and air traffic management (CNS/ATM) systems software integrity assurance,* RTCA & EUROCAE, March 2002.

[43] IEC 61508 *Functional safety: safety related systems,* 7 parts, http://ww.iec.ch, December 1998.

[44] *GNSS-1 Programme implementation phase, EGNOS software engineering standard*, to be obtained from the EGNOS programme office, August 1999.

[45] *Software for computers important to safety for nuclear power plants, Part 2, software aspects of defence against common cause failures, use of software tools and of pre-developed software*, http://ww.iec.ch, December 2000.

- FDA 1252 dedicated to software in medical devices.[46]
- UK SW01, for software safety assurance in Air Traffic services, similar in reach as DO-278/ED109.[47]
- Common Criteria, for applications with security concerns, started as a USA-European military harmonization, but as ISO standard is extending its reach.[48]

5.1 DO-178B/ED12B

For all software in an aircraft[49] this standard applies. Based on the FAR/JAR AC-25-1309 the following five software levels are defined by DO-178B. For convenience in Table 2 the FAR/JAR failure frequency definition per flight hour is included.

Level	System failure	Frequency description	FAR/JAR definition
A	Catastrophic failure	Extremely improbable	... < 10^{-9}
B	Hazardous / Severe major	Extremely remote	10^{-9} < ... < 10^{-7}
C	Major failure	Remote	10^{-7} < ... < 10^{-5}
D	Minor failure	Probable	10^{-5} < ...
E	No effect	Not applicable	-

Table 2 DO-178B/ED 12B overview

Detailed requirements are provided for each level. As it is not possible to measure actual failure rates at the required low rates, strict process guidance is provided. Complying with this process is considered sufficient. The excellent air transport safety record up to date does not repudiate this assumption. Many consider DO-178B as the toughest standard in industry.

Commercial off-the-shelf (COTS) products are officially allowed. However no requirements are waived. Consequently, only COTS products that have been developed specifically taking all DO-178B requirements into account can be used. Note that various navigation services[50] take DO-178B into account but deviate on details. The same holds for Galileo, due to its general domain funding. These examples demonstrate that for large COTS services air transport requirements will be taken into account but full compliance is not realistic.

[46] *Guidance for FDA reviwers and industry guidance for the content of pre-market submissions for software contained in medical devices*, http://www.fda.gov/cdrh, May 1998.

[47] CAP 670 *ATS Safety Requirements*, UK CAA http://www.caa.co.uk/docs/33/CAP670.pdf, April 1998.

[48] Common criteria (August 1999), *Common criteria for security evaluation, Version 21*. CCIMB-99-03, 3 parts http://www.commoncriteria.org/cc/cc.html, also known as ISO/IEC 15408.

[49] DO-178B / ED12B, *Software Considerations in Airborne Systems and Equipment Certification*, RTCA & EUROCAE, 1992.

[50] See for example

- *Wide Area Augmentation System description*, http://gps.faa.gov/programs/index.htm, 2003.
- *European Geostationary Navigation Overlay System description* http://www.esa.int/export/esaSA/navigation.html, 2003.
- *MTSAT Satellite- based Augmentation System description* http://www.mlit.go.jp/koku/ats/e/mtsat/miss/03.html, 2003.

Certification involving new software techniques, such as object-orientation, tends to be troublesome for the first applicant trying to certify it, as DO-178B tends to lag behind the current state-of-the-art in software engineering.

Certification is required for each nation where an airline wants to acquire an aircraft for civil use. Airbus has obtained its initial thirteen type certifications over the last ten years from the JAA, complemented by another thirteen from the FAA plus 130 from other nations. Boeing obtained 200 certificates in the same period, after the initial certification.[51] This should be improved.

5.2 DO-278/ED109

For ATM ground systems the USA has produced DO-278.[52] This standard extends DO-178B. Table 3 provides an overview of the six Assurance Levels (AL) defined in DO-278. Note that neither a definition of the assurance levels nor an indication of the allowed failure frequency is provided.

The use of COTS is acknowledged. Processes are defined for planning, acquisition, verification, configuration management and quality assurance of COTS. It has to be ensured that unused capabilities of COTS do not adversely effect the ATM system. An important extension to DO-178B is that COTS service experience may be used obviating the need to apply a DO-278 compliant development process for some assurance levels. However, the restrictions on service experience are quite severe. The information on service experience is included in Table 3. One year means that for a continuous period of 8760 hours of representative use no failure may have been encountered. Additionally all in-service reports, i.e. including those from other users, have to be evaluated for their potential adverse effects on the ATM system.

DO-178 level	DO-278 assurance level	COTS service experience
A	AL 1	Not allowed
B	AL 2	Negotiate with approval authority
C	AL 3	One year
	AL 4	Six months
D	AL 5	Typically not needed
E	AL 6	Not applicable

Table 3 DO-278/ED109 overview

[51] Holderbach, H.*Type certification of commercial aircraft call for enhanced international rules*, ICAO Journal 2, 2001.

[52] DO-278/ED109 *Guidelines for the communication, navigation surveillance, and air traffic management (CNS/ATM) systems software integrity assurance,* RTCA & EUROCAE, March 2002.

5.3 IEC 61508

From the general domain IEC 61508[53] is available. Four Safety Integrity Levels (SIL) are defined. It states that only for hardware the safety integrity can be quantified and assessed using reliability prediction techniques. For software qualitative techniques and judgements have to be made. It explicitly states that failures rates lower then 10^{-9} per hour, i.e. level A according to DO-178B or DO-278 AL1, can not be achieved for complex systems, which include every programmable system. Part 7 aims to provide an exhaustive list of techniques for each process phase, including recommendations on their use (or avoidance) for each SIL. It is possible to certify COTS for a certain level, when a IEC 61508 compliant development process is followed. An independent party does certification. Table 4 provides an overview of the 4 SIL levels.

SIL	Failure probability per hour (systems active > once per year)	Failure probability per demand (systems active < once per year)
4	$10^{-9} \leq \ldots < 10^{-8}$	$10^{-5} \leq \ldots < 10^{-4}$
3	$10^{-8} \leq \ldots < 10^{-7}$	$10^{-4} \leq \ldots < 10^{-3}$
2	$10^{-7} \leq \ldots < 10^{-6}$	$10^{-3} \leq \ldots < 10^{-2}$
1	$10^{-6} \leq \ldots < 10^{-5}$	$10^{-2} \leq \ldots < 10^{-1}$

Table 4 IEC 61508 Safety Integrity Levels

Using service experience is allowed, but in practice hardly possible for higher SIL levels. An example from the standard states that for a SIL 1 system, 95% confidence in correct functioning requires 300 hours of relevant service experience. For a SIL 4 system, 99.5% confidence requires 690,000 years of service experience.

5.4 Eurocontrol EATMP software task force

Eurocontrol has set up the "EATMP safety management software ad-hoc task group" for the European Civil Aviation Conference (ECAC) area. Based on the Eurocontrol Safety Regulatory Requirement,[54] a software standard is being drafted. This standard will combine DO-178B, IEC 61508, and the Capability Maturity Model[55] into a combined safety and quality assurance document. Software classification provided in Table 5 is not yet fixed. It is based on ESARR4, while inserting an additional level identical to DO-178B "level B." The requirements on the evidence that needs to be provided depend on the assurance level.

[53] IEC 61508 *Functional safety: safety related systems,* 7 parts, http://ww.iec.ch, December 1998.

[54] *ESARR 4 Software in ATM Systems,* Eurocontrol, http://www.eurocontrol.be/src/html/deliverables.html, October 2002.

[55] Paulk, M. C., C. V. Weber, S. M. Garcia, M. B. Chrissis, M. W. Bush, (February 1993), *Key Practices of the Capability Maturity Model, Version 1.1, Software Engineering Institute*, http://www.sei.cmu.edu/cmm/obtain.cmm.html.

Software assurance level	ESARR4 severity (Class, effect)		ESARR 4 occurrence likelihood	Software occurrence likelihood (operational-hour)
1a	1	Accidents	Improbable	N/A
1b		DO-178B level B	N/A	DO-178B Extremely Remote $10^{-9} < \ldots < 10^{-7}$
2	2	Serious incidents	Remote	$10^{-6} < ... < 10^{-5}$
3	3	Major incidents	Occasional	$10^{-5} < ... < 10^{-4}$
4	4	Significant incidents	Probable	$10^{-4} < ... < 10^{-3}$
5	5	No immediate effect on safety	N/A	N/A

Table 5 Eurocontrol EATMP software assurance levels

5.5 Electronic Flight Bag AC 120-76

The electronic flight bag is a (COTS-based) platform that supports many independent applications. Traditionally every aircraft application has its own hardware unit. The electronic flight bag can be part of the aircraft so DO-178B applies. However, it can also be used outside the aircraft, so a special document, FAA AC120-76[56] is available. The electronic flight bag could be a portable device like a PC or PDA, but it can also be installed in an aircraft. The electronic flight bag is classified as:

- Class 1 portable COTS device without data link connectivity to the aircraft and not connected to the aircraft power system or an aircraft mount;
- Class 2 portable COTS device mounted to the aircraft, can connect read-only to the aircraft data link and can connect to airline AOC information in receive/transmit mode;
- Class 3 installed equipment and consequently DO-178B applies in full.

For Class 1 and 2 equipment DO-178B compliance is not required. To illustrate the usefulness of Class 1 and Class 2, AC 120-76 lists 64 sample applications for Class 1 and 17 for Class 2. Class 2 mentions Internet connectivity, which fits well with the TALIS services approach and the Enterprise Application Integration ideas. There are restrictions on the software manipulation of electronic maps, even for Class 2, which may limit the benefit of Class 2 applications to increase pilot situational awareness.

Compliance to AC 120-76 implies compliance to 103 sections of five parts of the US Code of Federal Regulation (CFR) relating to airworthiness plus 45 additional sections of four parts of the operating regulations. In case electronic maps are used, DO-257[57] applies. Even within AC

[56] *Guidelines for the certification, airworthiness and operational approval of electronic flight bag computing devices*, FAA AC120-76, September 2002.

[57] *Minimum operational performance standards for the depiction of navigational information on electronic maps*, RTCA DO-257.

120-76 some parts relate to activities performed only once for the approval of software, other parts mention an operational approval valid for a specific operator for a specific period of time. This profusion of standards, regulations, etc., is typical for integrated aviation applications that aim to increase integration. Consequently, there is a need for a different approach to certification, which ensures the safety, but omits the many, not necessarily, harmonized standards and national sovereignty issues.

5.6 DRD 920

For the EGNOS program, ESA has produced a specific standard, DRD 920,[58] which combines the standard space-quality assurance practices PSS05 with DO-178B. Due to the many subcontractors and re-use of existing software these issues are addressed. ESA acts as customer as well as certifying authority. Consequently, for EGNOS additional certification activities need to be performed once EGNOS becomes operational in the member states.

As an annex to DRD 920 the PSAC (Plan for Software Aspects of Certification, a DO-178B required document) is provided. Its states that 40 DO-178B requirements are satisfied by the standard, 3 partial, 26 not satisfied and for 4 requirements it is unclear. For the unsatisfied requirements the contractor has to comply by other means. Human Machine Interface testing and specialized tools are specifically discussed in DRD 920.

Procured software is approved by ESA. For COTS software “certification material” is needed to satisfy the DO-178B objectives. When modifications of re-used software exceed 20% of the code, it is considered new code. A virus check for COTS is required, merging safety and security issues. DRD 920 standard describes its own software life cycle, which ignores iterations, deployment, maintenance and decommissioning. Despite a few useful innovations, it introduces is yet another standard without a kind of recognition of COTS certification according to different standards.

5.7 IEC 60880-2

In the nuclear industry IEC 60880-2 is applicable. IEC 60880[59] is based on the software classification provided in IEC 61226 and applies up to the highest “level A.” The basic single-failure criterion, the assembly of safety systems remains functional despite a random failure, is not applicable for software, as a software failure can cause multiple systems to fail. As a consequence, IEC 60880 devotes an appendix to the pros and cons of multiple diverse software

[58] *GNSS-1 Programme implementation phase, EGNOS software engineering standard*, to be obtained from the EGNOS programme office, August 1999.

[59] *Software for computers important to safety for nuclear power plants, Part 2, software aspects of defence against common cause failures, use of software tools and of pre-developed software*, http://ww.iec.ch, December 2000.

implementations. Multiple versions can only cover some fault classes, but an incorrect or ambiguous specification remains as single failure points.

The standard distinguishes between software tools that can introduce errors and tools that fail to detect them. The requirements for the former category are strict. Compilers (called translators) are acknowledged to be too large to demonstrate their correctness. They are trusted under certain restrictions, unlike DO-178B where binary code needs to be verified for the highest level. The compiler may not introduce dead code, or code which is not traceable to requirements (e.g. error handling). Operating experience may compensate for some lack of design documentation.

COTS, called pre-developed software, are allowed. There are strict requirements on the evaluation of functions, design documentation etc. In case operating experience is used, there are requirements on the operating history data. Also after acceptance of the COTS, all error and failure information has to be assessed for its potential impact on the approved system. The evidence provided by formal methods is recognized, unlike DO-178B.

5.8 FDA 1252

In the USA, FDA-1252[60] applies to software in medical devices. The software is classified into tree "levels of concern," see Table 6. As the probability of software failure can not be measured, only the severity of the software failure consequences is used to determine the level. A table listing 12 documents describes for each level of concern what type of information is needed, if any. No specific software life-cycle model is prescribed, but a general V-model for verification is provided. Verification needs to be performed at module, integration and system level.

Level of concern	Severity description
Major	Software failures that could cause, directly or indirectly, to death or serious injury of the patient and / or the operator
Moderate	Software failures that could cause, directly or indirectly, to non-serious injury of the patient and / or the operator
Minor	Software failures are not expected to cause injury to patient and / or operator

Table 6 FDA 1252 Level of concern

Artificial intelligence is allowed, as are neural networks. Even though it is stated that neural networks are impossible to verify, they are allowed for all levels of concern. Consequently the assumptions and the training of the neural network need to be verified, but no guidance is

[60] *Guidance for FDA reviewers and industry guidance for the content of pre-market submissions for software contained in medical devices*, http://www.fda.gov/cdrh, May 1998.

provided. Embedded and real-time systems pose unique concerns, but only the use of techniques, simulators, and emulators to analyze timing of critical events is mentioned. The importance of human factors is acknowledged without enforcing verification and validation requirements. In the same spirit security is raised, but no requirements ensue.

5.9 Software safety assurance in ATS UK SW01

The approach for safety assurance of ATM systems typically differs from the approach for aircraft. For the latter, a single certification is done, after which all aircraft of the same type are certified airworthy (in the country issuing the certificate). For services provided by ground systems, typically the national regulator issues a license to operate for a fixed period of time. After expiry of the license a periodic review is held and this license is extended for the same time. Unfortunately for ATM systems, as discussed above, no standard is (yet) internationally recognized. Even on a European level, harmonization is not completed. As the UK practice is advanced and informative, it will be discussed. UK-CAP-670[61] contains all regulations. The excerpt “Regulatory objective for software safety assurance in air traffic service equipment,” SW01[62] focuses on the software part. It defines Assurance Evidence Levels (AEL), to identify the type, depth and strength of evidence to be provided by the software development process to the assessor. In this way the developer may use any standard to provide the evidence. Based on the ESARR 4 safety classification, one AEL is defined for each class. Three types of evidence are acknowledged: test, field service and analytic. Every type of evidence can be either direct or backing, with requirements provided for each type.

This goal-oriented approach is more modern. It means that the supplier has to provide the classification, the type of evidence that is needed, the evidence itself, and reasoning that the evidence is adequate. This way of working would be more suitable for integrated NATO services than the profusion of the myriad standards currently available. This approach should be complemented by a mutual recognition scheme, which means that other countries would recognize approval in one country, preferably worldwide, but at least within NATO.

5.10 Security

Verification and validation are not only needed in case of safety concerns. It also applies for security. The Common Criteria[63] (CC) from the military domain is an international standard, which includes requirements on verification and validation. As such it is discussed in this section on standards.

[61] CAP 670 *ATS Safety Requirements*, UK CAA http://www.caa.co.uk/docs/33/CAP670.pdf, April 1998.

[62] http://www.caa.co.uk/srg/ats/default.asp?page=1371.

[63] Common criteria (August 1999), *Common criteria for security evaluation, Version 21*. CCIMB-99-03, 3 parts http://www.commoncriteria.org/cc/cc.html, also known as ISO/IEC 15408.

The Common Criteria aims to provide objective evidence about the product security level. Qualified and officially recognized assessors perform the objective and repeatable evaluation, much like for safety certification. The evaluation can lead to a certificate, which is currently recognized by 14 countries, including the USA, Canada, and several European NATO countries.

Security aims to prevent:

- Damaging disclosure of the service to unauthorized recipients (loss of confidentiality);
- Damage through unauthorized modification (loss of integrity);
- Damage through unauthorized deprivation of access to the asset (loss of availability).

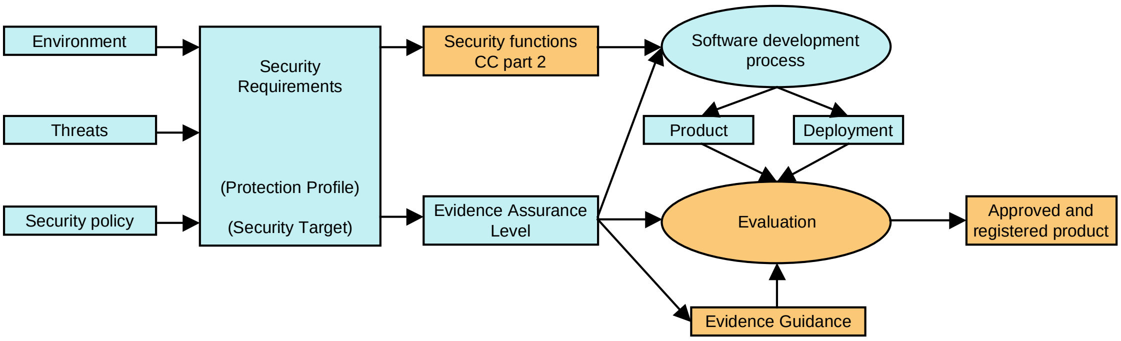

Figure 1 Overview common criteria

EAL	Description	# of COTS products	
		Certified	In evaluation
1	Functionally tested, security threats not serious	7	0
2	Structurally tested, low to moderate assurance	18	11
3	Methodically tested and checked, maximum assurance without infringing sound development practise	14	2
4	Methodically designed, tested and reviewed, maximum assurance compatible with good commercial practise	28	23
5	Semi-formally designed and tested, maximum assurance with moderate security engineering	1	0
6	Semi-formally verified design and tested, protect high value assets against significant risk	0	0
7	Formally verified design and tested, extremely high risk situations and/or high assets values	0	0
	Total # of COTS products	68	36

Table 7 Common Criteria Evaluation Assurance Level

The security environment provides the context of the asset. Combined with the perceived threats and the security policy the security requirements can be derived. These requirements consist of a re-usable Protection Profile (PP) and an asset specific Security Target (ST). Based on these requirements and the extensive listing of possible security functions in the common criteria part 2, the security functions of the system are determined. Separately the protection level is determined, which determines the amount of implementation and evaluation effort. Table 7 provides an overview of the Evaluation Assurance Levels (EAL), plus the amount of COTS products in the register at the time of writing.

The Common Criteria does add yet another level of requirements on the application development processes; so harmonization with the safety requirements is advantageous.

6 Certification

In this section, we examine the certification of embedded systems. First we review the meaning and different types of certification and discuss some of the barriers to certification. Next we address the relevance to NATO and consider a strategy for gaining maximum benefit from certification.

6.1 Different types of certification (who? what? why?)

Software certification is simply the process of generating a certificate that supports the claim that the software was either: (1) developed in a certain manner, (2) will exhibit some set of desirable run-time characteristics, or (3) has some other static characteristic embedded into it. For example, the certificate could simply state that a particular type of testing was applied and to what degree or thoroughness; or the certificate could state that the software can be composed successfully with any other software component with a certain set of predefined characteristics. Certification may be an attempt to transfer the risk to the certifier, but in many safety situations the user of the system still remains responsible for the safety.

Two key questions are “who does the certification?” and “what is being certified?”

Turning first to the question of who performs the certification. The process of creating a certificate is typically performed by one of three parties: by the vendor of the software, by a client of the software, or by an independent third party that is performing the service independent of the vendor or users.

Numerous examples of third party certification occur in industries such as electronics (e.g., Underwriters Laboratory), aviation (through designated engineering representatives), and consumer products (e.g., Consumer Reports). In industrial applications, the TUVs (see Page 79) dominate the market for the assessment of programmable controllers used in the process industry and are active in railways and nuclear assessments.

There are also wide ranges of certification activities that support interoperability. For example, The Open Group certifies that Linux implementations are indeed Linux and that X-windows conform to their standards. Other examples are protocol testing by telecom laboratories and compiler testing to demonstrate conformance with a language definition. The rather misnamed "Ada validation suite" tests for functional aspects of the compiler not all the required attributes such as reliability**.**

It is important to be clear by what is meant by the different kinds of certificates. To begin, these certificates are typically done as a follow-up check to insure that certain processes were followed during development; these "process certificates" are the first type of certification that will be discussed here. The second type of software certification often mentioned refers to the licensing of software engineering professionals; this is still referred to as software certification but should more appropriately be referred to as "professional licensing." The third and most important, but most difficult, claim that a certificate can make is to discuss how the software will behave in use. This is referred to as "product certification" and that will be our focus here. Note that there are three key messages that a certificate can convey that are not necessarily the same:

1. Compliance with standards versus
2. Fitness for purpose versus
3. Compliance with requirements

Compliance with the standards simply means that the standards that were required during development were indeed followed. However, that does not mean that the product itself is fit for the purpose that the user needed and it does not necessarily mean that the software complies with the requirements. The key difference between (2) and (3) is that those two are only equivalent if the requirements accurately and completely defined what the user needed the software to do, and thus it is possible that the software meets the requirements but does not perform as the user needs. Note that (1) deals with process certification, and (2) and (3) deal with product certification.

The beauty of having an accurate certificate is that it allows for a common language (mutual recognition agreements) to be employed to discuss relatively abstract notions. As we all know, software is somewhat amorphous in that, unlike a hardware entity, it is hard to get a handle on entities that are so abstract. So from that standpoint, certification standards can be beneficial. One classic example is the recent adoption by many nations of the Common Criteria, which is simply a process certificate that defines various security levels and the processes that must be employed to demonstrate those levels. And further, there are accreditation agencies (e.g., NIST) that now certify third party companies that actually perform the work and generate the certificates.

The term Independent Verification and Validation (IV&V) has long been used in the NASA and U.S. DoD communities, and so we now explain how that fits into the certification framework. To begin, Barry Boehm defines validation as making sure that you are building the right product and verification as making sure that you build the product right. And while these terms may seem confusing, they are both closely related. Little “i” V&V is simply performing first person V&V on a system, and big “I” V&V requires independence, from either a second party or third party (third party is typically preferred). How IV&V relates to certification is as follows: they are the same, provided that the type of certification being performed is either a process certification or product certification. However there is one caveat: IV&V does not necessarily result in a certificate.

There is a form of pre-certification or generic certification that gives “type approval” for a product. This might certify part of the product such as the operating system core along with the application development tools. This would allow each new application of the product (e.g., a programmable logic controller in a process plant) to focus on the fitness for purpose of the added application-specific code. This amortizes the costs of certification across the many potential applications.

6.2 Assessment of certification and relevance to NATO requirements

The development, deployment and integration of systems developed and perhaps certified to a wide range of standards is an inescapable part of the NATO landscape. There are benefits and threats from this:

Potential benefits:

- Certification can influence the market – minimum requirements that all software should satisfy might raise the general level of dependability or reduce costs to the use.
- Certification as a basis for gaining assurance.

- Risk transfer from the user or vendor to the certification authority.
- Guarantees of product behaviour (functional, non-functional).

Potential threats:

- Adding unnecessary costs and delays to projects.
- Giving unwarranted confidence in system behaviour.
- Preventing flexibility, innovation and inter-working, as certificates can be quite narrowly defined.
- Reducing ability of user to undertake their own examination of a product (why bother if someone else has already done it?).

One approach is to encourage the certification of civil products so that they can be more readily used in defence applications. This might be pre-certification or done on a particular project but of wider benefit (e.g., the level 7* profile for the memory management of some real time operating systems to be used in the F22 program).

While it unclear as to what the current world market is in software certification, we can look at the recent NIST report that said in the United States alone, the U.S. lost around $60B as a result of inadequate software testing in 2001. Certification, perhaps, even self-certification, might provide one way of making it more obvious to the market the extent to which software products have been tested and analyzed.

Note that there are two differing camps on certification: there are those that believe any “bar” that people are forced to cross is better than no bar at all. And there are those that believe that any bar lulls those that produce products into a false sense of security: as long as they do just enough to cross the bar, then they have done enough to satisfy minimum industry standards. In practice any certification approach will have an impact on the market and the behaviour of suppliers and so the issues for NATO are not solely technical and any strategy musty be cognizant of the, perhaps subtle, interplay of technical, social and market forces.

Future NATO systems are likely to be heterogeneous, dynamic coalitions of systems of systems and as such will have been built and assessed to a wide variety of differing standards and guidelines. Here, our main recommendation is that the certification of embedded systems should be based around the concept of dependability case, generalizing the current requirement for safety and reliability cases. “Reliability case” found in U.K. “Def Stan 00-42 Part 3,”[64] which is on the Reliability and Maintainability Case and “Part 2” deals specifically with the software

[64] The U.K Defence Standards can be obtained from http://www.dstan.mod.uk/.

reliability case. Similar requirements appear in equivalent NATO standards. Safety cases are required in U.K. "Def Stan 00-55" and in the more recently the U.K. CAA Safety Regulation of air traffic management systems and its proposals.[65]

The dependability case should be based on a claims-arguments-evidence[66] framework with the following components:

1. A goal-based view of that expresses certification requirements in terms of a set of claims about the system and its attributes
2. Evidence that support the claims
3. An explicit set of argument that provides a link from the evidence to the claims
4. For critical systems the underlying assumptions and concepts used to support and formulate the goals and claims should be described in terms of a series of models (e.g. at system, architecture, design, implementation levels)

Such a framework should provide the technical basis that allows for inter-working of standards especially IEC 61508 and DO178B.
While this goal based approach is moving from safety to other dependability areas (e.g. to interoperability), it needs supporting technical work and the development of a body of practice. While there is considerable experience with this approach for safety applications, it will be novel to many NATO partners and the deployment of the approach would be facilitated by:

1. Guidance on strategies and arguments for demonstrating claims and on how claims might be derived including guidance on what are useful certifiable and measurable properties
2. Guidance on how evidence is generated by VV&C techniques. This is not found in any existing standards and we have begun to elaborate recommendation on how evidence is generated throughout the lifecycle. Details are in Appendix B. It should be noted that highly critical systems are extremely difficult. To obtain substantially improved MTBF (beyond commercial software) one requires orders of magnitude in reduction of fault density.
3. Guidance on pragmatics such as to feasibility, scalability and tool support.
4. Guidance on the relationship to and interface with frequently used standards.

[65] CAP670 SW01, *"Requirements for Software Safety Assurance in Safety Related ATS Equipment."*
[66] An introduction to safety cases on which these ideas are built can be found at www.adelard.com, which hosts the guidance for the IEE Functional Safety portal on this topic.

However, perhaps the most compelling study would be one that looked at the return on investment (ROI) and risk and benefits of the approach. Some work is being performed in the European nuclear Cemsis project. In this project the cost-benefit claims are:

- Reduced costs particularly assurance costs for critical systems.
- Reduced uncertainty in costs. The extra costs of a safety related system has varied between 6-100% of the cost of a system of normal industrial quality. Uncertainty arises from direct costs and time
- Reduced time to develop and assure systems important to safety.
- Increased commonality between different countries/utilities and with other industrial sectors.

6.3 Certification Conclusions

There are disparate forms of certification and they vary widely in their objectives, the level of detailed information they provide, and their ability to effectively reduce project and system risk.

NATO should encourage certification as a means of supporting interoperability and confidence in the systems (products) behaving as required. Process certification should support confidence in the evidence (and normally be a pre-requisite to baseline standards such as ISO9001) but the overall emphasis of certification should be on the product. We propose that a claim-argument-evidence-based approach should be adopted as best practice.

We are cautious about the actual arrangements for certification and whether self-certification or independent certification is required. For critical systems this is likely to be already best practice and this should be continued. For less critical systems, the benefits of independent oversight should be assessed and activities that add value to projects identified (rather than just a complex time consuming overhead).

Critical systems (whether security, safety, operational) pose particular challenges to current development and certification approaches.

7 Conclusions and Recommendations

In this final section of the report we summarize our conclusions and recommendations. These points were formed during the writing of the above sections, but were specifically formed during two days of meetings in Prague, The Czech Republic, in October 2003.

In general terms, we find that much of our discussion is not specific to embedded systems; the problems of verification, validation and certification encompass many kinds of systems. We observe that trust, in itself, is a sociological-technical matter; it is not attained wholly through technical means.

In the Terms of Reference for the Task Group, it is stated that the

> "… Task Group is to review the techniques currently used in the software industry to product high quality products; an appropriate number of methods and software life-cycle metrics for systems of relevant complexity should also be examined, particularly those which are supported by fully operational environments. At the conclusion of this activity, the Task Group should deliver a report addressing the following topics:
>
> Assessment of current technical capabilities and relevance of these techniques and methods to embedded military systems
> Assessment of relative strengths and limitations of these methods
> Assessment of current research trends in testing, formal methods, and requirements traceability
> Specific recommendations for military application of these techniques
> Recommendation for future NATO IST efforts, if relevant, such as Symposium or Workshop."

For the rest of this section, we provide a summary discussion of each of the aforementioned topics.

Assessment of current technical capabilities and relevance of these techniques and methods to embedded military systems

It is the Task Group's conclusion that certification can be used successfully to increase our confidence in systems. However, that confidence could be expensive. We have observed that certification is working with bounded problems and has been particularly successful in the Avionics arena. However, we also note a couple of cautionary aspects to the use of certification in the military realm. Many military systems are not bounded (e.g., consider coalitions), nor, unlike Telecom for example, are there operational profiles in the battle space.

Further, we note that certification is only as good as the standard one is certifying against and the individuals who are performing the certification. In fact, there is substantial reliance on “good teams” to achieve the requisite goals. There is a threat that standards may become “gameable.”

Without doubt, the complexity of a certification grows with system size and complexity. We can generally certify “in the small;” the problems of attaining certification of “systems-of-systems” are extremely complex and, perhaps, with current technology, not truly achievable for highly critical systems. We further observe that to obtain substantially improved mean-time-between-failures, one requires orders of magnitude reduction of fault density.

A bottom line perspective on certification is that it is unclear as to what is the precise message accruing from certification. There is certainly an element of transferring risk from one organization to another. However, in the broad scheme, it is an open question what certification actually achieves. Of concern to the embedded systems charter of the task group, is that certification generally occurs at the system level, not at that of embedded system components. So for example, there are concerns pertaining to Integrated Modular Avionics, since it is the system that is certified, not the component.

Assessment of relative strengths and limitations of these methods

All of the aforementioned methods are extremely difficult to use with highly critical or complex systems. To obtain substantially improved “mean time to failure,” one requires orders of magnitude reduction in the number of faults. In general, there is a complex interplay of certification, legal issues, licensing, market and technology.
From a technical perspective, we found that current technical capabilities are not well developed for:

- Verification, validation and certification for component reuse. Obviously a serious problem, for amortization of the costs of VV&C over component reuse is necessary for broader adoption of these capabilities.
- Rigorous predictability of the behavior of networks and systems. This inability to attain predictability will constrain the adoption of systems in highly critical contexts.
- Non-interference of multi-level critical processes.

Assessment of current research trends of these methods

In the body of this report, we have discussed a number of research trends for embedded systems and the VV&C of such systems. We observe the following current research trends:

- Increasing use of Commercial-Off-The-Shelf (COTS) systems.
- Increasing occurrence of systems of systems.
- An increasing number (already too numerous) of unharmonised standards. We posit that the “goal-based approach” described above could be a potential solution. There are some preliminary indications of this trend arising from the use of goal-based certification and regulation as espoused in 00-55 and SW01.
- The ongoing reality of striving for cheaper and faster systems, while attempting to maintain quality objectives.
- Increasing adoption of formal modeling and analysis techniques in industrial tools.
- Research and development into the complementary aspects of various verification and validation techniques. For example, the use of automated deduction to extend the reach of model checking. A further example is the TTA[67] Group’s efforts to verify functionality and safety of their Time Triggered Protocol. Formal verification was used to check the protocol’s key algorithms for consistency, stability and safety. Fault tolerance behavior and error detection properties were investigated through the injection of millions of faults. On the certification side, a number of efforts have been completed. The design and development process of their key chip (TTP-C2) is documented against aerospace standards. An “operating system” and “configuration checker” are being developed according to DO-178B (Level A).
- There are numerous challenges in determining “fitness for purpose” of embedded systems. An isolated system may be fine, but changing the context to a networked device may open unsavory issues (e.g. networked SCADA devices). There are further challenges with the modeling and analysis of embedded systems hardware and software interfaces (sensors).
- A major research area requiring investigation is the “interoperability of standards.” For example, if system A is assessed against a standard SA and B is assessed against a standard SB, what does SA and SB tell us about how A and B will interact?
- Ongoing and heightened (post 9/11) concerns with the security of networked embedded systems. Supervisory Control and Data Acquisition (SCADA) systems are of particular concern.
- Increased interest at correlating the threat space (whether security or safety) with the verification, validation and certification techniques to be used.

[67] See www.tttech.com.

- A trend towards scenario based software development and towards systems being certified against scenarios.
- A trend towards the use of goal-based certification and regulation. However, further R&D supporting goal-based approaches is required. Especially a need to build up a body of practice and supporting technical work. Exploration is required for the techniques used to generate evidence, combine evidence, for expressing claims and for the presentation of evidence.

Specific recommendations for military application of these techniques

The research task group has the following specific recommendations for military applications of these techniques:

1. That NATO gives serious consideration to using a goal-based approach to certification of systems.
2. That when performing the verification, validation and certification of "critical components," regardless as to whether the criticality is safety- or security-critical:
   - That VV&C be performed and/or assessed by an independent third party.
   - Whether such independent third parties should be accredited.
   - That objective guidance be provided for required verification and validation artifacts.
   - That maximum freedom of Information Technology processes and techniques be provided within the scope of the VV&C framework.
   - That the certification effort reflects the criticality of the system.
3. That for highly critical systems, NATO
   - Insert formal modeling and reasoning into their development processes.
   - Insert advanced static analysis techniques.
   - Apply extensive testing.
4. That for systems of lesser criticality, NATO can consider as a <u>minimum</u> baseline the following:
   - Use standard "good" quality assurance practices.
   - Test, test and test again.
   - Perform ongoing error monitoring of Commercial-Off-The-Shelf systems.
   - Apply ISO 9000, which is not specific to software.
   - Trend to the application of dependability cases, which is already underway under one of the NATO quality assurance groups.

As noted, this is a minimum baseline. Emerging techniques (such as new static analysis techniques, innovative approaches to formal analysis and modeling, new testing techniques, deployment of dependability) should be continually evaluated and prudently incorporated.

Recommendation for future NATO IST efforts, if relevant, such as Symposium or Workshop

1. Dissemination

The research task group has three recommendations regarding the dissemination of VV&C results to both the NATO community and the larger R&D community. From a NATO perspective, the use of a lecture series and/or workshop would help to disseminate results and help to transfer knowledge of verification, validation and certification of embedded systems. Furthermore, the RTG observed during its deliberations that there were difficulties in interacting with the operational side of NATO, yet the interaction between operations and R&D is important for both communities. The R&D community will find out what are the specific pain points being faced by operational staff; the operational staff will learn what new capabilities may make some portions of their tasks simpler. Consequently, a workshop format through which R&D and operational representatives could interact could be an important feature in ameliorating the above difficulties. Finally, it is the intention of the authors of this report to write a synopsis of this report and submit the synopsis for publication in a widely disseminated journal.

2. Dual Use of High Assurance Technologies

The primary recommendation arising from the research task group's deliberations was the recommendation for a new task group entitled "Dual Use of High Assurance Technology." This new Task Group has been approved and is identified as IST-048/RTG-020. The intent of the new RTG is as follows:

> *High Assurance Technologies (such as formal methods) have normally been used to develop systems requiring high degrees of assurance as to functionality, safety and security. One of the key benefits of such technologies is their ability to ferret out subtle problems with system requirements, design and implementation.*
>
> *However, experience has also been obtained in which such technologies can be used to identify exploitable weaknesses in systems potentially being used by adversaries. From a formal methods perspective, the phrase "formal methods-based tiger teaming" is meant to capture the idea of using formal methods to help drive the identification of flaws in such systems. The interaction of mathematical modeling with experimental testing of such systems has been shown to be very effective, but not widely known, in identifying exploitable flaws.*

> *In effect, there is a duality between all assurance technologies and offensive measures. Knowledge from assurance can be used in (1) deciding on the type of attack, and (2) where to attack. For example, Byzantine proofs assume synchrony so attacks could be based on falsifying the assumption. Nuclear protection static analysis does not deal properly with concurrency, so one should attack the concurrency basis. Safety cases reveal preoccupations and guide one to areas not normally considered.*

The specific objectives of the RTG are:

The Task Group will review and evaluate techniques currently used for high assurance, general V&V activities, and information warfare. Based on the review and evaluation, the Task Group will develop (possibly meta-level) processes for effectively identifying exploitable weakness in critical infrastructure. These processes will be based on a fusion of the most effective techniques found in high assurance, V&V and information warfare activities. This activity will be a joint research effort of the participating nations.

Appendices

A Terms of Reference

I. ORIGIN

Background

In September 1996, the Flight Vehicle Integration Panel made a proposal to form a Working Group that was approved by RTB in March 1997 under the same title. This group never started due to some difficulties to meet the experts from the appropriate organisms. In addition, RTO was restructured within the same period. The new SCI Panel received the legacy of this activity and decided at its 1998 Fall Business Meeting to ask IST Panel if this activity could be transferred under the IST panel auspices.

The idea for a Task Group has been discussed both in IST Panel meetings and via e-mail. In a discussion at its Fall 1999 meeting, a sufficient number of Panel Members supported the formation of a Task Group on the subject for more detailed and substantial work. Embedded systems are ubiquitous in military systems as well as in commercial products used in military applications. From tactical radios to range finders to avionics control units, the Battlespace is filled with digital devices that rely on embedded software for proper operation and functionality. For life-critical and mission-critical systems, extensive testing is required; yet the size and scope of embedded software in today’s products precludes exhaustive testing.

Justification (Relevance to NATO)

Some testing techniques, such as modified condition/decision coverage (MC/DC), have been adopted by government agencies as required methods. Yet there is no body of knowledge that characterizes the types of faults typically found by the various methods and which types of faults might remain undetected. Furthermore, performance testing and stress/load testing are oftentimes overlooked until the product is fielded. Thus, verification and validation are critical efforts for embedded systems. Furthermore, the recent Y2K exercises demonstrated the difficulties many vendors experienced in certifying software. Those Y2K issues related strictly to date and time and yet were quite costly. Expanding the scope to include proper functionality under planned and unplanned usage is important, especially for safety-critical systems; cost-effective and timely methods are needed for certification.

II. OBJECTIVES

Aim

The aim of the Task Group on “Validation, Verification, and Certification of Embedded Systems” is to consider better techniques to validate and verify embedded systems. In classic terminology, these two issues are:
“Are we solving the correct problem?” and
“Are we solving it correctly?”

Beyond the issues of verification and validation, there is the question of the basis on which these two aspects are judged. Is there a means to certify that systems meet their intended requirement specifications? This is of critical importance since many embedded systems are mission-critical and life-critical applications.

Specific Goals

The Task Group will evaluate new testing techniques, partial or complete verification techniques, and certification techniques for correctness as well as better means of requirements specification and traceability in order to validate, verify, and certify the reliability of embedded systems. The following topics will be considered:

- Verification, including
- Formal methods of correctness
- Applications of model checking and theorem proving
- Validation, including requirements traceability
- Certification, including probabilistic measures of effectiveness (MoE) of survivability in the presence of system faults as well as intrusions or attacks
- Survivability and/or graceful degradation and/or anomaly management
- Cost and performance estimation of embedded systems, where performance includes both speed of execution and reliability
- Automated techniques for requirements traceability
- Techniques for testing stress and overload conditions
- Design techniques for test and recovery
- Interface testing techniques for embedded systems which could include both specialized and commercial-off-the-shelf (COTS) components

Expected Deliverables and/or End products

The proposed Task Group is to review the techniques currently used in the software industry to product high quality products; an appropriate number of methods and software life-cycle metrics for systems of relevant complexity should also be examined, particularly those which are supported by fully operational environments. At the conclusion of this activity, the Task Group should deliver a report addressing the following topics:

Assessment of current technical capabilities and relevance of these techniques and methods to embedded military systems
Assessment of relative strengths and limitations of these methods
Assessment of current research trends in testing, formal methods, and requirements traceability
Specific recommendations for military application of these techniques
Recommendation for future NATO IST efforts, if relevant, such as Symposium or Workshop.

Duration
Three years from approval by RTB

III. RESOURCES

Membership
Invited experts nominated by participating nations should include government, industry, and academic participants working in relevant areas.

Recommended Technical Team Leader: Initially, Dr. Ann MILLER (US)
Recommended Lead Nation: United States

NATIONS WILLING TO PARTICIPATE : BE, CA, CZ, FR, NL, PO, **US**.

National and/or NATO resources needed
Some consulting support, as determined by first Task Group meeting,
Symposiums' support if that is the proposed recommendation.
No special needs are foreseen except for Internet access.

SECURITY CLASSIFICATION LEVEL

Both Activity and publications will be NATO Unclassified

PARTICIPATION BY PARTNERS NATIONS

This Task Group will welcome Partners' participation

LIAISON

Other NATO Panels will be invited to send representatives if desired.

B TAP

<table>
<tr><td colspan="2"><b><i>ACTIVITY</i></b></td><td>Task Group (RTG)</td><td colspan="13" rowspan="2"><b>Validation, Verification and Certification
Of Embedded Systems</b></td><td colspan="4">03 / 2000</td></tr>
<tr><td colspan="2"><b><i>Activity REF. Number</i></b></td><td>IST-027 / RTG-009</td><td colspan="4">01 / 2001</td></tr>
<tr><td colspan="3"><b><i>PRINCIPAL MILITARY REQUIREMENTS</i></b></td><td>1</td><td>2</td><td>3</td><td>4</td><td></td><td colspan="2"></td><td></td><td></td><td colspan="4">NU</td><td colspan="4">12 / 2003</td></tr>
<tr><td colspan="3"><b><i>MILITARY FUNCTIONS</i></b></td><td>1</td><td>2</td><td>3</td><td></td><td></td><td colspan="2">6</td><td>7</td><td>8</td><td>9</td><td>10</td><td colspan="2">11</td><td>12</td><td colspan="2">13</td><td>14</td></tr>
<tr><td colspan="3"><b><i>PANEL AND COORDINATION</i></b></td><td colspan="9">IST (Information Systems Technology)</td><td colspan="5"></td><td colspan="3"></td></tr>
<tr><td colspan="3"><b><i>LOCATION AND DATES</i></b></td><td colspan="14">Various participating NATO countries</td><td colspan="3">P-I</td></tr>
<tr><td colspan="3"><b><i>PUBLICATION DATA</i></b></td><td colspan="6">Technical Report (TR)</td><td colspan="6">Fall 2003</td><td colspan="3">150 / 180</td><td colspan="2">NU</td></tr>
<tr><td><b><i>KEYWORDS</i></b></td><td colspan="2">Validation</td><td colspan="5">Verification</td><td colspan="7">Certification</td><td colspan="5">Performance</td></tr>
<tr><td colspan="3">Embedded Systems and Software</td><td colspan="5">Trustworthiness</td><td colspan="7">Reliability</td><td colspan="5">Real-Time</td></tr>
</table>

I. BACKGROUND AND JUSTIFICATION (Relevance to NATO)

Embedded systems are ubiquitous in military systems as well as in Commercial Off The Shelf (COTS) products used in military applications. From tactical radios to range finders to avionics control units, the battlespace is filled with digital devices that rely on embedded software for proper operation and functionality. For life-critical and mission-critical systems, extensive testing is required; yet the size and scope of embedded software in today’s products precludes exhaustive testing. There is no body of knowledge that characterizes the types of faults typically found by the various methods and which types of faults might remain undetected. Furthermore, performance testing and stress/load testing are oftentimes overlooked until the product is fielded. Two fundamental issues are (i) “Are we solving the correct problem?” and (ii) “Are we solving the problem correctly?” Beyond the issues of verification and validation, there is the issue of certification, especially for mission-critical and life-critical embedded systems. Thus, verification and validation (V&V) are crucial development tasks for embedded systems. Typically, military systems devote significantly less development time for V&V compared to commercial systems. Effective strategic and tactical methods for Validation, Verification, and Certification (VV&C) of embedded military systems are needed.

II. OBJECTIVES

The aims of the Task Group on “Validation, Verification, and Certification of Embedded Systems” are: (i) to compile and critique existing and emerging techniques to validate and verify embedded systems, (ii) to provide recommendations for potential NATO adoption, and (iii) to review the applicability of civil and military certification standards and their effects on validation and verification of NATO systems.

III. TOPICS TO BE COVERED

The following topics will be considered as a part of the task group's VV&C of embedded systems:

AdoptionAdoption models/impact on emerging technologies
CertificationAnalysis of existing and emerging standards (FAA 178B/US and GAMT 17/France)
Classification of Embedded Systems Time triggered architectures (for instance)
Design for non-functional requirements Validation, verification, traceability, etc.
Estimation Cost and performance estimation of systems, performance including both speed of execution and reliability
Measures Measures of effectiveness (MoE) that can be used to evaluate the operational impact and efficacy of techniques
Process Techniques for requirements elicitation, capture and specification
Effect of specific techniques to the certification process
Validation Techniques for testing stress and overload conditions
Interface testing techniques for systems which could include both specialized and COTS components
Role of testing techniques in certification
Verification Specification, analysis, proof
Role of verification techniques in certification

IV DELIVERABLE AND/OR END PRODUCT

At the conclusion of this activity, the Task Group should deliver a Final Report addressing the following topics:

(i) to compile and critique existing and emerging techniques to validate and verify embedded systems,
(ii) to provide recommendations for potential NATO adoption., and
(iii) to review the applicability of civil and military certification standards and their effects on validation and verification of NATO systems.

In addition a website containing relevant materials pertaining to the VV&C of embedded systems will be developed.

V. **TECHNICAL TEAM LEADER AND LEAD NATION**

Recommended Team Leader: Prof. Ann MILLER (US)
Recommended Lead Nation : **United States**

VI **NATIONS WILLING TO PARTICIPATE**: BE, CA, CZ, FR, NL, PL, PO, **US**.

VII **NATIONAL AND/OR NATO RESOURCES NEEDED**

The RTG is expected to deliberate on the specific topics given above, over the Internet, in 1-2 day meetings to be held 2-3 times a year and, possibly, in a workshop.

VIII **RTA RESOURCES NEEDED**

Consultants will be asked upon request.

C Programme of Work

This Programme of Work describes the manner in which the Task Group RTG-009 shall carry out its activities as identified in the Terms of Reference (TOR).
The aim of the Task Group on “Validation, Verification, and Certification (VV&C) of Embedded Systems” is to evaluate techniques to validate and verify embedded systems software. Beyond the issues of verification and validation, are methods to certify software relative to the environment within which the software reside. These activities are of critical importance since many embedded systems are mission-critical and/or life-critical applications.
The following topics will be considered as a part of the task group’s evaluation of techniques for the VV&C of embedded systems software:

Adoption	Adoption models/impact on emerging technologies.
Certification	Analysis of existing and emerging standards (FAA 178B/US and GAMT 17/France).
Classification of Embedded Systems	Time triggered architectures (for instance).
Design for non-functional requirements	Validation, verification, traceability, etc.
Estimation	Cost and performance estimation of systems, performance including both speed of execution and reliability.
Measures	Measures of effectiveness (MoE) that can be used to evaluate the operational impact and efficacy of techniques.
Process	Techniques for requirements elicitation, capture and specification. Effect of specific techniques to the certification process.
Validation	Techniques for testing stress and overload conditions. Interface testing techniques for systems that could include both specialized and COTS components. Role of testing techniques in certification.
Verification	Specification, analysis, proof. Role of verification techniques in certification.

At the conclusion of this activity, the Task Group shall deliver a final report addressing the following topics:

1. Assessment of current technical capabilities and relevance of these techniques and methods to embedded military systems.

2. Assessment of relative strengths and limitations of these methods.
3. Assessment of current research trends of these methods.
4. Specific recommendations for military application of these techniques
5. Recommendation for future NATO IST efforts, if relevant, such as Symposium or Workshop.

The nations that have agreed to participate are Belgium, Canada, The Czech Republic, France, Georgia, The Netherlands, Poland, Portugal, Slovakia and the U.S. This Task Group welcomes the participation of all NATO Nations and Partners for Peace nations. The list of names and contact information for RTG-009 participants is attached.

RTG-009 is expected to deliberate on the specific topics given above, over the Internet, in 1-2 day meetings to be held 2-3 times a year and, possibly, in a workshop. Some support from RTA for consulting shall be requested. Workshop or symposium support shall be requested, if that is the proposed recommendation. Otherwise, no special needs are foreseen except for Internet access.

RTG-009 shall use appropriate NATO/RTA standards for documentation.

RTG-009 milestones and deliverables are summarized in the following table. Further milestones and deliverables will be added as required.

Date	Milestone	Deliverables
March 2001	RTG-009 formed	Minutes Revised TAP Pertinent DCI list
May 2001	2nd RTG-009 Meeting	Minutes Revised ToR PoW
May 2001	Partners for Peace Meeting	Presentation on RTG-009 status
June 2001	IST Panel Meeting	Presentation to IST Panel on RTG-009 status
October 2001	3rd RTG-009 Meeting (including Preliminary Workshop)	Minutes Workshop presentations Final Report Outline Web site structure Identification of FY2002 activities
December 2001		Annual report
Spring 2002	4th RTG-009 Meeting	T.B.A.
December 2002		Annual report
December 2003		Annual report
March 2004	Workshop, if proposed	
March 2004		Final Report

The October, 2001 RTG-009 meeting will consist of a Preliminary Workshop (with 3-5 presentations including Cazin on the state of the practice of safety/security partitioning and Voas on the state of the research on intrusion tolerance. The Preliminary Workshop will be followed by an RTG-009 business meeting.

D Participants

The following table identifies the members of the research task group. The following members were particularly active and have played a material role in the development of this report: Robin Bloomfield, Jacques Cazin, Dan Craigen, Natalia Juristo, Ernst Kesseler and Jeff Voas.

'*' means principal member.

Task Group Chairman

CANADA

*Mr. Dan CRAIGEN

ORA Canada
27-2000 Thurston Drive.
OTTAWA, Ontario K1G 4K7
CANADA

e-mail : dan@ora.on.ca
Tel : +1 (613) 241 8692
Fax : +1 (613) 241 8704

Task Group Members

CZECH REPUBLIC

Dip. Eng. Radek ORSAK

Air Force Research Institute Prague
VTUL a PVO Praha
tr. kpt. Jarose, 27
602 00 BRNO
CZECH REPUBLIC

e-mail : radek.orsak@vtul.cz
Tel : +420 (5) 41.18.24.50.
Fax : +420 (5) 45.24.61.80.

FRANCE

Mr. Jacques CAZIN

ONERA/DTIM
2, Av Edouard Belin B.P 4025
TOULOUSE, CEDEX 3
FRANCE

e-mail : jacques.cazin@cert.fr
Tel : +33 (0) 5.62.25.25.90 31055
Fax : +33 (0) 5.62.25.25.93

GEORGIA

Vice-Col Shota GELENIDZE

Deputy Head, Central Board of Strategy
Planning and Scientific-Technical
Researches
Moscow avenue, 7/1
380020 TBILISSI
GEORGIA

e-mail : ist_georto@yahoo.co.uk
Tel : +995 (32) 92.37.79.
Fax : +995 (32) 95.69.49.

POLAND

Maj. Andrzej STASIAK

Faculty of Cybernetics
Military University of Technology
Kaliskiego 2
00-908 WARSAW
POLAND

e-mail : stasiak@iar.wat.waw.pl
Tel : +48 (22) 685.70.25.
Fax : +48 (22) 685.71.44.

PORTUGAL

Prof. Fernando BRITO E ABREU

INESC (Instituto de Engenharia e
Sistemas de Computadores)
Rua Alves Redol, 9
1000 LISBON
PORTUGAL

e-mail : fba@di.fct.unl.pt
Tel : +351 (21) 310.02.63.
Fax : +351 (21) 314.58.43.

SLOVAK REPUBLIC

Lt. Col. Dr. Josef TKAC

Military Academy
Kt-302, PO Box 45
03101 LIPTOVSKY MIKULAS
SLOVAK REPUBLIC

e-mail : tkac@valm.sk
Tel : +421 (849) 552.22.34.
Fax : +421 (849) 552.22.37.

SPAIN
Dr. Natalia JURISTO

Facultad de Informática
Universidad Politecnica de Madrid
Campus de Montegancedo
28660 - Boadilla del Monte
MADRID
SPAIN

e-mail : natalia@fi.upm.es
Tel : (+34) 91336 6922
Fax : (+34) 91336 6917

THE NETHERLANDS
Dr. Ernst KESSELER

National Aerospace Laboratory (NLR)
P.O. Box 90502
1006 BM AMSTERDAM
THE NETHERLANDS

e-mail : kesseler@nlr.nl
Tel : +31 (20) 511.34.62.
Fax : +31 (20) 511.32.10.

UNITED KINGDOM
Mr Robin BLOOMFIELD

Adelard and CSR
Drysdale Building,
Northampton Square
LONDON EC1V 0HB
UNITED KINGDOM

e-mail : reb@adelard.co.uk
Tel : +44 (20) 7490.9453
Fax: +44 (20) 7490.9451

U.S.A.
Dr. Jeffrey VOAS

Cigital
21351 Ridgetop Circle, Suite 400
DULLES VA 20166
U.S.A.

e-mail : jmvoas@cigital.com
Tel : +1 703 404 92 93
Fax : +1 703 404 92 95

Task Group Members (pending confirmation)

BELGIUM

Prof. Dirk VAN HEULE

Ecole Royale Militaire (MMWW)	
Department of Mathematics	e-mail :
avenue de la Renaissance	Dirk.Van.Heule@mmww.rma.ac.be
B 1000 BRUSSELS	Tel : +32 (2) 737.65.23. 30
BELGIUM	Fax : +32 (2) 737.65.12.

CZECH REPUBLIC

Dr. Roman CARDA

Air Force Research Institute	
Vojensky technicky ustav letectva a PVO	
Praha	
Mladoboleslavska ul	e-mail : roman.carda@vtul.cz
PRAGUE 9	Tel : +420 (2) 20.20.72.01.
CZECH REPUBLIC	Fax : +420 (2) 82.72.87. 197 06

Task Group Liaison Member

CANADA

*Dr. Malcolm R. VANT

Deputy Director General, Defence R&D	
Canada-Ottawa	
3701 Carling Avenue	e-mail : Malcolm.Vant@drdc-rddc.gc.ca
OTTAWA, ONT K1A 0Z4	Tel : +1 (613) 998.25.69.
CANADA	Fax : +1 (613) 998.45.60.

E Combating Terrorism Recommendations

During the period of the RTG's deliberations, the group was asked to recommend courses of action in support of combating terrorism. The group identified the following four recommendations based on their experience and technical backgrounds. The first recommendation (Dual Use of High Assurance Technologies) was approved as a follow on activity for the current RTG.

Dual Use of High Assurance Technologies

High Assurance Technologies (such as formal methods) have normally been used to develop systems requiring high degrees of assurance as to functionality, safety and security. One of the key benefits of such technologies is their ability to ferret out subtle problems with system requirements, design and implementation.

However, experience has also been obtained in which such technologies can be used to identify exploitable weaknesses in systems potentially being used by adversaries. From a formal methods perspective, the phrase "formal methods-based tiger teaming" is meant to capture the idea of using formal methods to help drive the identification of flaws in such systems. The interaction of mathematical modeling with experimental testing of such systems has been shown to be very effective, but not widely known, in identifying exploitable flaws.

In effect, there is a duality between all assurance technologies and offensive measures. Knowledge from assurance can be used in (1) deciding on the type of attack, and (2) where to attack. For example, Byzantine proofs assume synchrony so attacks could be based on falsifying the assumption. Nuclear protection static analysis does not deal properly with concurrency, so one should attack the concurrency basis. Safety cases reveal preoccupations and guide one to areas not normally considered.

Economic and Technical Trade-offs of the non-functional Attributes of Systems

The grand challenge facing the software quality research community is the ability to accurately define, in the very earliest stages of development, the techniques that will be need to achieve the needed levels of the various non-functional attributes (a.k.a. the "ilities"): reliability, availability, fault tolerance, testability, maintainability, performance, software safety, software security, etc. Note, however, that there are associated technical and economic trade-offs that must be made in order to achieve quality, and also to certify software quality. To satisfy a particular level of each attribute requires specific cost expenditures, and some of these attributes conflict with each other. Therefore to gain confidence that the likelihood that cyberterrorism will be unsuccessful requires some way to balance the economic and technical tradeoffs among the software "ilities". This is an open research question that to date has not been adequately addressed by the software engineering research community.

Mitigating Attacks on Embedded Systems

Embedded systems are evolving towards becoming ubiquitous and network (I.P.) enabled. The resulting open nature of such systems is vastly different from the current preponderance of embedded systems where access is tightly controlled and, often, physically isolated from non-relevant systems.

The evolving open nature of embedded systems significantly changes the threat landscape for such systems. Embedded systems designers must be aware of the new landscape and apply/develop technologies to mitigate attacks arising from the new opportunities available to our adversaries. One possible attack, for example, would be a denial of service attack on a hard real time embedded system.

Software security classification (and enforcement)

Observation: for embedded software safety and security are characteristics which share similarities. Consequently an approach for security could benefit from the substantial work on safety.

The various software safety standards (e.g. DO-178B, DO-278, IEC61508 and its derivatives for railways etc) all consider the various (software) functions of a system. A safety (or risk) assessment then provides the basis for a qualification of the various functions to the defined safety levels. Subsequently for each safety level, development standards of various rigors are prescribed. The most safety critical levels mandate very rigorous software development and verification methods. Measures have to be taken to prevent unintended interference in case software classified at several levels has to execute on one shared platform.

The suggestion is that similarly for security issues the functions or features of an embedded system are also to be classified. Using a partitioning scheme, it can be ensured that the most important functions have a high probability of remaining fully functional whereas nice to have options might succumb to attacks. Such partitioning software of course has to be of the highest or most reliable level itself. For these "core" functions rigorous methods like formal methods, extensive verification etc can be applied to assure they will be functioning as required. Usually the most important functions are small compared to the total set of requirements so it becomes more affordable to apply the rigorous development methods (or conversely the available effort can be allocated to the most critical functions). Standardized levels provide a uniform (internationally recognized) measure of how much the function can be trusted to function correctly.

F Software Certification Agencies and Services Offered

There are a number of organizations that are active in a variety of different types of software certification[68]

- TUVs
- National Laboratories
- Open Group
- Factory Mutual
- Microsoft
- CASS

In the UK the National Physical Laboratory provides services in assessing several kinds of software. These include compilers, OSI communication stacks, and numerical software. Other National Laboratories such as TNO in the Netherlands have similar roles.

Factory Mutual is a major US based insurance organization It manages a not-for-profit organization Factory Mutual Research (FMR) which has a wide-ranging programme of safety assessment. Although originally based in the USA, FMR have recently become active in Europe, and have formed alliances with TÜV Product Service and TÜV Rheinland. They have a programme of functional and product safety assessment.
Microsoft identified that a major source of Windows problems was due to third party software, especially driver software, causing problems with memory corruption. They initiated a programme on driver verification with an associated driver certification scheme. Note that this does not verify that the driver is fit-for-purpose only that it does not threaten the system software that it works with.

The CASS scheme is an initiative to support the certification of safety-related systems against IEC 61508 [RD 1]. Although a UK scheme, the first certification organization to be accredited under CASS is claiming significant international interest.

[68] This section is summarized from an Adelard ESA/Astrium study.

G Mapping techniques to properties and attributes

Definition of properties

There are a variety of ways in which the desired properties of a system can be classified. Indeed there has been considerable work in trying o define the dependability terminology. Here we take a practical set of attributes from the draft air traffic regulation in the UK CAP670 SW01.

- **Functional properties**: The primary functional behaviour of the software
- **Timing properties**: The time allowed for the software to respond to given inputs or to periodic events, and/or the performance of the software in terms of transactions or messages handled per unit time.
- **Robustness**: The behaviour of the software in the event of spurious (unexpected) inputs, hardware faults and power supply interruptions, either in the computer system itself or in connected devices.
- **Reliability**: The probability that the software will perform to a specified requirement for a specified period of time under specified conditions.
- **Accuracy**: The required precision of the computed results.
- **Resource usage**: The amount of resources within the computer system that can be used by the application software.
- **Overload tolerance**: The behaviour of the system in the event of, and in particular its tolerance to, inputs occurring at a greater rate than expected during normal operation of the system.

Bibliography techniques

2. SAFETY ANALYSIS
   2.1 Preliminary Hazard Analysis (SYS)
   2.2 HAZOP

Fault propagation analysis

   2.3 Cause Consequence Diagrams/Event-Tree Analysis

2.4 FMEA/FMECA/Fault Tree Analysis
2.5 Hardware/Software Interaction Analysis

3. FAULT AVOIDANCE

Organizational methods

3.1. ISVV
3.2. Independent Safety Assessment
3.3. Metrics for process
3.4. Metrics for product
3.5. Maturity assessment
3.6. Traceability analysis
3.7 Procurement policy

Design techniques

3.8. Application-oriented Languages
3.9. Formal Analysis
3.10. Formal Specification
3.11. Formal Development
3.12. Simulation of human operations
3.13. Modeling (SDL, Petri…)
3.14. Safe Language Subsets /Strongly Typed Programming Language

4. FAULT DETECTION

Testing

4.1. Structural /Functional /Requirement based /Stress testing
4.2. Regression testing
4.3. Fault Injection

Other techniques

4.4. Inspections and Walkthroughs
4.5. Static analysis – control and data flow
4.6. Schedulability analysis / Worst case analysis

5. FAILURE DETECTION

Control Flow Monitoring

5.1. Watchdog
5.2. Dynamic logic
5.3. Signature analysis / Memorized executed cases
5.4. Exception handling
5.5. Run time anomalies detection

Data Monitoring

5.6. Assertions / Plausibility checks
5.7. Integrity checks / Detection codes
5.8. Defensive programming

6. FAILURE CONTAINMENT

Fail Safe

6.1. Wrapping
6.2. Safety supervision
6.3. Manual Override

Fault Tolerance

6.4. N-version Programming
6.5. Recovery blocks

7. MONITORING AND FEEDBACK

7.1. Event recorders
7.2. Internal error and status recording
7.3. Fault reporting
7.4. Incident reporting
7.5. Accident analysis

Attribute technique mapping (partial)[69]

Techniques/object (IWA)		contractual boundary		
	Requirements	**Specification**	**Design**	**Code**
Functional properties	Review Traceability Animation, prototyping 7*[1];	Review Traceability Animation 3.8; 3.10; 3.11;	Review Traceability 3.8; 3.11; 4.4; 5.1; 5.2; 5.3**; 5.4**; 5.6-5.7**	Review Acceptance tests 3.11; 4.1; 4.4; 5.2; 5.4**;
Interoperability (close to non-interference, fault propagation - but of interfaces)	Composability via formal notations? Use of standards, orbs (object request brokers) 2.2;	Composability via formal notations? 2.3;		Interface testing
non-interference (of features and intended functions; of other "unused" features)	2.2;	Partitioning Proof Fault/failures propagation Covert channel analysis 6.1; 2.3;	Partitioning Covert channel analysis Fault/failures propagation - flow based tools; guideword methods 2.3; 4.4;	Static analysis - information flow, pointer analysis Tests - fault injection to look at fault propagation Covert channel analysis 2.3; 4.4; 4.5;
Timing properties	Review Simulation	Schedulability analysis 3.13; Timed Petrie nets	WCET Design review looking at mismatch between architecture and requirements Schedulability analysis 4.6; 3.13;	WCET Timing tests Schedulability analysis 4.1; 4.6

[69] This table is taken from IPFI10.4, CAA regulation.

Correctness	(could check correctness wrt to system requirements)	Review wrt requirements	Review wrt specification Refinement proof	Review Refinement proof Testing 3.14; 4.1;
Absence of faults (specific classes)	Inconsistency, incomplete - Animation, prototyping	Inconsistency -	Language subsets Code generation Design review looking at mismatch between architecture and requirements 3.14;	Code analysis (for certain fault classes) Buffer overflow - code scanning Symbolic execution (array bounds, divide by zero) 3.14; 4.5;
Reliability, availability	Field experience Process Process profiling	Field experience Process Process profiling 6.4-6.5	Project specific metrics Process profiling 5.1; 5.2; 6.4-6.5;	Statistical testing Project specific metrics (N/T) Process profiling 4.1; 5.2;
Accuracy	Review Simulation	Numerical analysis Simulation	Numerical analysis	Acceptance testing 4.1;
Resource usage	Review Simulation	Performance analysis (architecture)	Performance analysis (architecture)	Performance analysis Integrity checks (locking, stack, pointer access..) 4.1;
Robustness (all interfaces; internal, external, human)	Review Simulation 2.2;	Analysis for independence; diversity redundancy Single failure criterion 2.3;	Analysis for independence; diversity redundancy Single failure criterion 2.3; 5.8;	Fault injection Stress testing 2.3; 4.3; 5.8;

Overload tolerance (linked to timeliness)	Review Simulation	Review h/w; s/w issues Identify worst cases Simulate design Identify coping strategies; degradation Schedulability analysis	Identify worst cases Schedulability analysis	Stress testing Schedulability analysis 4.1; 4.3;
Maintainability (see non-interference)	Review	Review	Review	Review 4.2;
Modifiability	Review; definition of environment; types of change	Modularity	Tests for coupling; cohesion	Trial 4.2;
Usability	Task Analysis Prototype trials Good practice? 3.12;	Task Analysis Prototype trials Thea 3.12;	Thea, model checking procedures	Trials
mis-use testing (could be robustness to user)	Experience, ethnography, trials to identify "Hazard" identification techniques Attack trees Tiger teaming	Formal methods based attacks Tiger teaming	Formal methods based attacks Tiger teaming	Formal methods based attacks Tiger teaming 4.1; 4.3;
Safety	2.1 ; 2.2; 6.2; 6.3;	6.2;		4.3;
Security	Implement patches (req on process; product) Define security policies; vertical domain standards(constraints on dev. processes and constrains on			Testing for patches updates 4.1;

	components that are selected); *look at intrusion detection and tolerance schemes; patch management*			
Confidentiality	Requirements often via standards or legislation (hippa - medical US)			
or users	Passwords, bio-metrics PKI			
Integrity (data, information)		Review design for techniques. Specification via security policy	encryption,	
Availability				
Insider attacks, malicious code (see authentication to restrict access)				Scanning binaries