Critical Software for Human Spaceflight

The equipment software for Orion Propulsion

Antonio Preden (1), Jens Kaschner (2), Felix Rettig (3), Michael Rodriggs (4)

(1) Orion ESM Avionics and Software Lead, European Space Agency, Noordwijk, The Netherlands.
(2) Orion ESM PDE Software Technical Authority, Airbus Defense and Space, Bremen, Germany.
(3) Orion ESM PDE Lead Engineer, Airbus Defense and Space, Bremen, Germany.
(4) Orion Service Module Software Technical Lead, NASA Johnson Space Center, Houston, US.

ABSTRACT

The NASA Orion vehicle that will fly to the moon in the next years is propelled along its mission by the European Service Module (ESM), developed by ESA and its prime contractor Airbus Defense and Space.

This paper describes the development of the Propulsion Drive Electronics (PDE) Software that provides the interface between the propulsion hardware of the European Service Module with the Orion flight computers, and highlights the challenges that have been faced during the development. Particularly, the specific aspects relevant to Human Spaceflight in an international cooperation are presented, as the compliance to both European and US standards and the software criticality classification to the highest category A.

An innovative aspect of the PDE SW is its Time-Triggered Ethernet interface with the Orion Flight Computers, which has never been flown so far on any European spacecraft.

Finally the verification aspects are presented, applying the most exigent quality requirements defined in the European Cooperation for Space Standardization (ECSS) standards such as the structural coverage analysis of the object code and the recourse to an independent software verification and validation activity carried on in parallel by a different team.

1 THE ORION ESM ARCHITECTURE

ESA and its prime contractor Airbus Defense and Space are developing the European Service Module (ESM) for the NASA Orion vehicle for human space exploration mission [2]. The ESM provides to Orion the generation of power through 4 deployable solar arrays, the power distribution to the ESM and Crew Module (CM) users, the passive and active thermal control system for ESM and CM using thermistors and fluid loops, the storage and delivery of water and gas to the CM, and the propulsion system. Fig. 1 shows the modules of the Orion Vehicle, in particular the service module and the crew module.

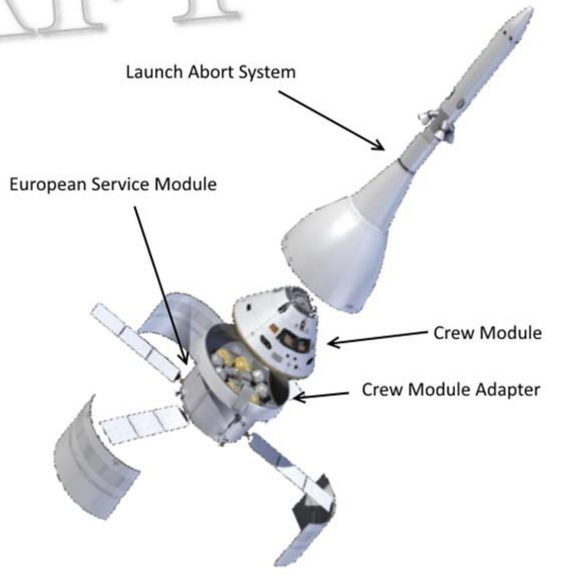

Fig. 1. Modules of Orion Vehicle

The hardware in the ESM is controlled by a set of electronic units, developed by Airbus Defense and Space and its subcontractors:

- Propulsion Drive Electronics (PDE), controlling the propulsion hardware
- Pressure Regulation Unit (PRU) controlling the electrical regulation of the pressure in the ESM propulsion tanks
- Power Control and Distribution Unit (PCDU), providing the users in the ESM and the CM with two different types of power at 28V and 120V, and controlling the power provided to the CM battery or supplied by the CM batteries depending if the ESM solar panels are on sun or in eclipse phase.

- Solar Array Driving Electronics (SADE), controlling the rotation of ESM solar arrays on two axes via the four driving mechanisms
- Thermal Control Unit (TCU), managing the active and passive thermal control systems and the storage and delivery of consumables such as water and gas to the CM
- Fluid Control Assembly (FCA), including the electronics for the control of the pumps for the active cooling system.

The overall mission and vehicle management is executed by the Orion flight computers, located in the CM and developed by NASA and its subcontractors. The ESM electronic units communicate with the Orion flight computers via a 1GBit Time-Triggered Ethernet network, called Orion Onboard Data Network (ODN).

Fig. 2 depicts the overall avionics subsystem architecture and interfaces. The interface to the SLS launcher is depicted on the left side of the drawing, and the CM and CMA (labeled as SM-CM I/F Adapter) are on the right side.

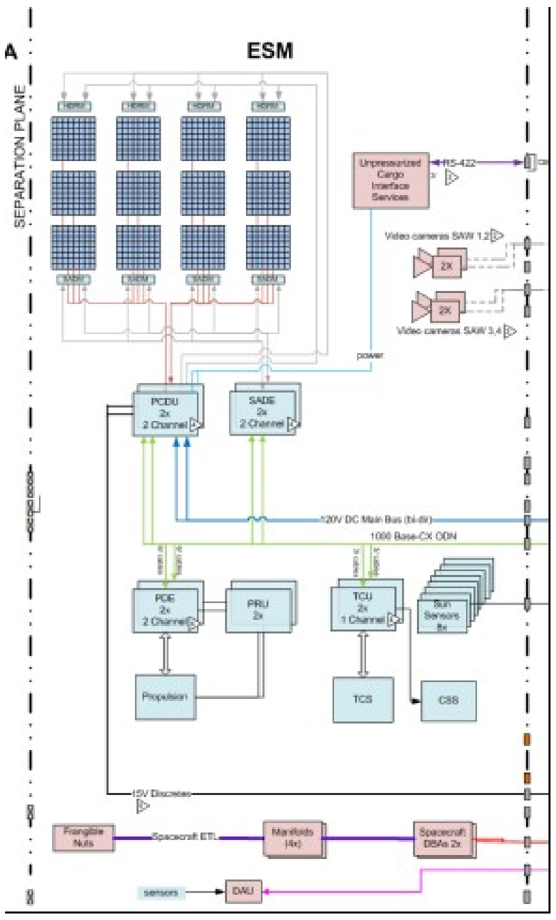

Fig. 2. ESM Avionics architecture

1.1 The ESM propulsion subsystem and the PDE mission

One of the main subsystems in the ESM is the Propulsion [3], that includes three different types of engines: the 27.7 kN Main Engine, reused from NASA's Space Shuttle program, gimbaled on two axes by a Thrust Vector Control system and providing the boost for the main orbital manoeuvers; the eight auxiliary engines, providing a thrust of 490 N each, used together with the Main Engine for the orbital manoeuvers; and 24 Reaction Control System (RCS) thrusters to ensure attitude control of the vehicle.

The ESM propulsion is equipped with an innovative electrical pressure regulation system, controlled autonomously in the ESM on the basis of setpoints fixed by the Flight Software depending on the phases of the mission.

The ESM Propulsion Drive Electronics (PDE) is the main interface between the Orion Flight Computers and the ESM Propulsion hardware. The PDE receives through the ODN interface the commands to operate the engines according to the results of the control algorithms in the Flight Software, and provides to the Flight Software the readings from the sensors allowing monitoring and failure detection of the propulsion hardware. In the fulfillment of this mission two categories of requirements play a key role: reliability and time reactivity.

1.2 The ATV experience

The design of the ESM is inherited from the experience of the Automated Transfer Vehicle (ATV), that serviced the ISS between 2008 and 2015 with 5 different missions.

The first heritage is represented by the PDE itself: the overall concept of the ESM PDE comes as an evolution of the ATV PDE, by expanding its functions to control a wider range of engines and a gimballing mechanism, modernizing the Mil-Bus interface to a Time-Triggered Ethernet one, increasing its reactivity from the 100ms ATV cycle to the 25ms of Orion, and adapting its reliability to meet the requirements for a crewed mission.

The second major heritage is the development of critical software. On ATV, the most critical functions were centralized in the Monitoring and Safing Unit (MSU), which was in charge of establishing and maintaining a safe position of ATV with respect to ISS by executing and controlling a retrograde manoeuvre, overriding the ATV Flight Application Software (FAS) and the ATV PDE by direct commands to the propulsion hardware. The presence of the MSU allowed for the reduction of the criticality of the ATV FAS and PDE software, and only the MSU software had been developed to the highest criticality A grade [1]. On ESM an independent safing system such as the MSU is not present, and the reliability requirements are not only associated with collisions but also spread over the entire flight; this forced the development of PDE SW to criticality category A.

As of today, the ATV MSU SW and the ESM PDE SW are the only two examples of Category A software developed on ESA projects.

2 PDE ARCHITECTURE

2.1 PDE description

The PDE architecture is designed for high reliability and fault-tolerance protecting from inadvertent activation of critical actuators.

The ESM has two identical PDE boxes (Fig. 3) each comprising two independent channels providing several actuator controllers and sensor interfaces. At the system level, a high degree of redundancy is achieved by connecting nominal and redundant actuators and sensors to different PDE channels. Each PDE channel communicates with the Flight computers in the CM via a dedicated Time-Triggered Ethernet interface.

Within each channel, a hardware/software co-design architecture combines high fault-tolerance and time precision while maintaining flexibility. Time-critical low-level actuator controllers and acquisition of sensor measurement data are implemented in radiation-hard anti-fuse FPGAs (Microsemi RTAX) while high-level control, processing and validation of commands from the Flight Computers, and telemetry data generation are implemented in software running on a fault-tolerant LEON2-FT processor.

All actuator power drivers are designed one-fault tolerant by implementing two independent safety barriers in series which are controlled by separate FPGAs. In addition, the health status of each barrier is monitored by the FPGA not controlling the barrier itself. The operation of the low-level FPGA functions is controlled and monitored by the software, rendering the software the only central single point of failure.

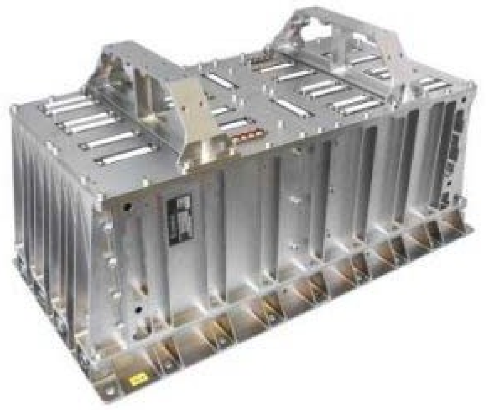

Fig. 3. ESM PDE box comprising two channels

2.2 Time-Triggered Ethernet interface

The exchange of command and telemetry data with the Orion Flight computers is realized with Time-Triggered Ethernet (TTE) with three independent lanes per PDE channel. TTE provides high reliability and robustness, and guarantees deterministic data timing according to a well-defined schedule [5]. Each PDE channel is equipped with a dedicated Standard Network Interface Card (SNIC) delivered by NASA subcontractors. The SNIC also provides a precise synchronization signal defining the system-wide 25ms operation cycles.

2.3 PDE On-board computer

Each PDE channel comprises a controller board (MPCC) comprising an Atmel AT697F implementation of ESA's fault-tolerant LEON2-FT processor based on a SPARC V8 32-bit RISC architecture running at 100 MHz. The Atmel ATC18RHA CMOS process is radiation-hardened and utilizes several single event effect mitigation techniques: the architecture provides full triple module redundancy, transient filtering, error detection and correction (EDAC) for registers and external memories, and parity protection of the caches. The AT697F also provides a 33 MHz PCI interface, a memory controller, and several peripherals including UARTs, general purpose I/Os, as well as an embedded Debug Support Unit (DSU) with trace buffers.

Via the processor bus, an external 2 MByte (+EDAC) SRAM, Flash memories containing four independent 4 MByte software images, as well as three high-reliability anti-fuse FPGAs are attached. Access to and content of the SRAM and the Flash memory are protected by LEON-internal error correction and detection (32 data + 7 EDAC bits). FPGA-internal registers and memory blocks are mapped into the processor's I/O space. Accesses to these resources are protected by application-level mechanisms comprising multiple bus transactions. In addition to using a highly reliable radiation-hard processor device, a processor watchdog is implemented in a dedicated Supervisor FPGA.

Permanent comprehensive self-monitoring of the system firmly embedded in the 50ms command/measurement period (bus accesses, integrity of FPGA registers and state machines, health status of driver barriers) and deterministic cache refreshing further improves reliability and protection from inadvertent activation ofcritical functions.

2.4 Interface with Pressure Regulation Unit

Since the Pressure Regulation Unit of the ESM does not provide a direct ODN interface, the PDE channels act as bridges for the communication between the CM flight computers and several Pressure Regulator Unit channels.

An Orion-specific implementation of the High-Level Data Link Control (HDLC) protocol is used for point-to-point serial data links between individual PDE and PRU channels. Data integrity across the entire data chain is ensured on several levels: the HDLC interfaces provide CRC protection at the link level; the PDE-internal data buffers implement error detection and correction; the actual data frames exchanged between CM flight computers and PRU are transported transparently by the PDE to provide dedicated end-to-end CRC protection.

3 PDE SOFTWARE DEVELOPMENT

3.1 Software standards

Development of software for space application in Europe is carried on according to ECSS standards, in particular:

- ECSS-E-ST-40C "Space engineering: Software"
- ECSS-Q-ST-80C "Software product assurance"
- ECSS-Q-ST-40C "Safety"

As the Orion Flight Software is developed by NASA and its subcontractors, a different standard is applied:

- NPR 7150.2A "NASA Software Engineering Requirements"

The NASA Procedural Requirements (NPR) impose requirements on software development, whether created by NASA or developed for NASA programs and in the case of the ESM, provides a common set of requirements, that has been used to map to European industry standards.

Due to the integrated nature of the ESM and the CM within the Orion spacecraft, the NASA requirements imposed on the ESM lead to the need to harmonize the software standards applied on the different components, so that the vehicle level analyses could rely on activities and documentation from PDE SW.

An option to harmonize the standards could have been to make NPR 7150.2A formally applicable to all the ESM SW, and develop it according to US standards. However, the learning curve of using a different standard that the one usually applied in Europe would have had unacceptable cost and schedule impacts, so this option had to be modified.

A joint ESA/NASA activity took place in order to map the requirements of NPR 7150.2A to the ECSS clauses, in order to show how the NPR Software Engineering Requirements would be met. Rather than a generic "meet or exceed" exercise between the two standards, the mapping has been oriented to identify the actual activities and artefacts developed for PDE SW, either because it was required by ECSS or because it was generated by other plans that are applicable at the project level. The results have been captured in the bilateral document MPCV 72547 "Agreement on applicability of NASA software engineering requirements to ESM", and it has been agreed to between ESA and NASA that by applying the ECSS standards as defined in the MPCV 72547, ESA will show compliance to the requirements in NPR 7150.

As a support, the – still partial – results of the working group on "Mutual recognition of S&MA standard NASA, ESA and JAXA" have been used as a reference [4].

The results of the mapping of the 132 requirements of NPR 7150.2A have been:

- 42 requirements are of programmatic nature, or otherwise organization related. These were determined to be not applicable to ESA and no mapping to ECSS has been done.
- 90 requirements are of technical nature, and are applicable to ESA. A mapping to ECSS has been identified for each of those.

The single requirement applicable to ESA for which no compliance could be shown is the Software Engineering requirement 32 (SWE-032), which requires that all human rated space software systems are certified to Capability Maturity Model Integration (CMMI) Maturity Level 3. The ECSS Q80 requests the supplier to monitor and control the effectiveness of the development processes (§5.7.2), and in particular §5.7.7.2 requests that assessments shall be in conformance with ISO/IEC 15504, Software Process Improvement and Capability Determination (SPICE) [7]: however, no specific model (such as CMMI) is imposed. This was incompatible with NASA requirement, which explicitly calls for CMMI level 3, while Airbus was at the time only certified to level 2. As this requirement is applicable at NASA's Agency level, the Orion project had no authority to waive it; a dedicated waiver on requirement SWE-032 was then requested to NASA's Chief Engineer and eventually obtained. The waiver was based on the use of other standards and project plans that showed how the CMMI ML3 technical process area requirements would be met, and was supported by the proven experience of Airbus on previous projects such as Columbus and ATV.

3.2 Software criticality

A main step to prepare the PDE SW development has been the determination of its criticality. The concept of software criticality category is introduced in ECSS-Q-ST-80C "Software Product Assurance", and is based on the consequences that the loss or degradation of a software function can have at system level, on a scale going from the lowest Category D (minor or negligible consequences) to Category A (catastrophic consequences, such the loss of life). An implication of this approach is that a system level safety analysis is necessary to determine the effects of the loss of a software function, and this can be only performed once a preliminary design of the system is defined.

TABLE I. ECSS SOFTWARE CRITICALITY CATEGORIES

Software Category	Definition of Software Category
A	Software that if not executed, or if not correctly executed, or whose anomalous behaviour can cause or contribute to a system failure resulting in: → **Catastrophic consequences**
B	Software that if not executed, or if not correctly executed, or whose anomalous behaviour can cause or contribute to a system failure resulting in: → **Critical (On Ground) / Mission (In Flight) consequences**
C	Software that if not executed, or if not correctly executed, or whose anomalous behaviour can cause or contribute to a system failure resulting in: → **Major consequences**
D	Software that if not executed, or if not correctly executed, or whose anomalous

Software Category	Definition of Software Category
	behaviour can cause or contribute to a system failure resulting in: → **Minor *or* Negligible consequences**

Alternatively, the approach defined by NASA in the NPR 7150.2A is a classification of the safety-criticality of the software based on 8 Classes, from A-Human Rated Space Software Systems down to E-Small Light Weight Design Concept and Research and Technology Software (Classes F through H cover business and information technology related software in decreasing order of applicability) . The designation of the software in a criticality class is based on criteria such as the domain of use of the software, the extent to which humans depend upon the system, and the criticality of its use. According to NPR 7150.2A criteria, all the software needed to perform primary function on a human rated system belongs to Class A, independently of any analysis on the consequences of failures.

TABLE II. NASA SOFTWARE CRITICALITY CATEGORIES

Software Class	Definition of Software Category
A	Human Rated Space Software Systems. → Any SW developed and/or operated by or for NASA that are needed to perform a primary mission objective of human space flight and directly interacts with human space flight systems.
B	Non-Human Space Rated Software Systems or Large Scale Aeronautics Vehicles → Flight and ground software that must perform reliably to accomplish primary mission objectives, or major function(s) in Non-Human Space Rated Systems.
C	Mission Support Software or Aeronautic Vehicles, or Major Engineering/Research Facility Software → Flight or ground software that is necessary for the science return from a single (non-primary) instrument, or that is used to analyze or process mission data, or other software for which a defect could adversely impact attainment of some secondary mission objectives or cause operational problems.
D	Basic Science/Engineering Design and Research and Technology Software → Ground software that performs secondary science data analysis, or supports engineering development, or is used in testing other Class D software systems.
E	Small Light Weight Design Concept and Research and Technology Software → Software developed to explore a design concept or hypothesis, but not used to make decisions for an operational Class A, B, or C system or to-be built Class A, B, or C system, or software used to perform minor desktop analysis of science or experimental data.
F - H	Business and IT software.

The profound difference in the two approaches makes it impossible to map the ECSS “categories” to the NPR “classes”. By NASA’s definition of Class rating, NASA rates the PDE SW is as Class A. In agreement with NASA, the ECSS approach has been applied to determine the criticality of PDE Software, with a Software Criticality Analysis that has been prepared at Subsystem level and assessed at System level, resulting in the determination of the Category A for the PDE SW.

The classification to Category A has a major impact on the development cycle of the software. In particular, two requirements that are specific to category A software are the verification with 100% modified decision condition coverage of source and object code (E40 §5.8.3.5), and the execution of Independent Software Verification and Validation (ISVV) by a 3rd party organization (Q80 §6.3.5.28). The impacts of these requirements on the PDE SW are detailed in the next sections.

3.3 Coding rules

ESA coding rules BSSC(2000)1 vs. industry standards like e.g. the Motor Industry Software Reliability Association (MISRA) coding rules represent different aims of the rule set with some consequences: the MISRA coding rules concentrate mainly on practical rules which seem to have the goal of enforcing the production of 'good' source code even by un-experienced programmers. The resulting rigid rule set may jeopardize the overall quality of the designed software, but on the other hand allows a high degree of automated verification by commercially available tools (e.g. PolySpace). The ESA coding rules instead are of more 'philosophical' nature and have a clear aim towards a higher overall quality of the produced source code with the consequence that the resulting rules are -in general- not verifiable by tools. With a certain amount of goodwill, some ESA coding rules can be mapped onto MISRA coding rules allowing automated verification, but for the majority of the ESA coding standard such a mapping is not feasible. As a consequence, the application of ESA coding rules requires skilled/experienced and self-disciplined programmers in conjunction with skilled/experienced reviewers necessary for the peer-reviews and code walk-throughs foreseen during the software development cycle. No tool can do this job for You!

3.4 Software architectural design

The software architectural design resulting from the given requirements baseline as well as from the system analysis, lead to the development of a 'bare-bone' software for the LEON-based PDE MPCC:

- no operating system, no synchronous interrupts
- no 3rd party libraries
- no board support packages or automatic start-up code
- reuse of a well-proven boot loader for basic processor set-up and PDE software loading during start-up
- a control main loop as simple as possible, synchronized by events derived from the Time-Triggered Ethernet, with a strict processing schedule and a clearly predictable timing behavior

The result of the software architectural design is a simple main loop with two 25 ms sub-cycles (one 'command', one 'measurement' sub-cycle) and an overall cycle time of 50 ms, visualized in Fig. 4.

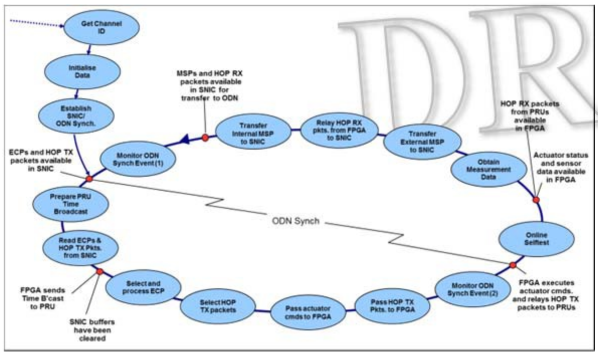
Get Channel ID
Initialise Data
Establish SNIC/ ODN Synch.
MSPs and HOP RX packets available in SNIC for transfer to ODN
Transfer Internal MSP to SNIC
Relay HOP RX pkts. from FPGA to SNIC
Transfer External MSP to SNIC
HOP RX packets from PRUs available in FPGA
Obtain Measurement Data
Actuator status and sensor data available in FPGA
ECPs and HOP TX packets available in SNIC
Monitor ODN Synch Event (1)
Prepare PRU Time Broadcast
ODN Synch
Online Selftest
Read ECPs & HOP TX Pkts. from SNIC
Select and process ECP
Select HOP TX packets
Pass actuator cmds to FPGA
Pass HOP TX Pkts. to FPGA
Monitor ODN Synch Event (2)
FPGA executes actuator cmds. and relays HOP TX packets to PRUs
FPGA sends Time B'cast to PRU
SNIC buffers have been cleared

Fig. 4. PDE software main loop

No real surprise was that the usage of the Time-Triggered Ethernet as the one and only communication channel with the PDE software had no major effects on the software design: it behaves in principle like the well-known MIL-Bus with a well-defined time framing while providing higher band width with triple-redundancy not visible to the user.

Except for the re-used bootloader written in assembler all other parts of the PDE software are designed for coding in the high-level programming language 'C' based on the gnu tool chain.

3.5 Development principles

Some of the used principles in developing the PDE software in order to minimize the risk for failures are listed hereafter:

- Usage of a well-known and often used tool chain, adopted already in various other projects
- Adherence to the goal of simplicity for the detailed software design as well as for coding in order to achieve simple, fully tested and thus highly reliable software
- Usage of object-oriented principles in the S/W design to achieve module separation
- Disciplined source code development using version-controlled template files reflecting the corresponding ESA Board for Software Standardization and Control (BSSC) rule set
- Exclusive usage of 'atomic' logical decisions to ease the target of modified condition and decision coverage (MCDC) thus allowing for 100% unit testing with 100% source code and object code coverage
- Source code development cycle with several peer reviews and internal (development group) and external (ISVV, ESA) audits resulting in 'early finding reports' in order to obtain readable, coding rule-conformant and understandable source code
- Usage of tools for static code analysis (e.g. PolySpace) to catch well-hidden bugs as well as un-necessary source code statements
- Development and unit testing on a dedicated H/W-platform as similar as possible to the PDE target H/W in order to detect early any possible problems in the object code interaction with the target H/W
- Close collaboration with product assurance already early in the development phase (that is the point where software quality and reliability comes into the developed code) and adherence to the implemented software production tool chain (compilers / linker / version control / target deployment / testing and reporting)

The following paragraphs will pin-point only some of the various concepts, techniques and tools used to develop the Category A software for PDE.

Using object-oriented principles in the S/W design allows early proto-typing while still providing flexibility to react to changes induced by the environment (H/W and/or system). The encapsulation of 'effector groups' (i.e. actuator groups) and 'basic services' (e.g. CRC32-calculations or FPGA-access or access to the Time-Triggered Ethernet) into dedicated modules allows for the creation of a set of 'library modules' with high-level functionality available for call by the main loop of the PDE software. The benefits of this design approach have been proved several times during the development cycle, in particular for what concerns the encapsulation of effects caused by required changes or additional functionality decided late in the project cycle.

The C-pre-processor is the developers friend - this often under-estimated tool helps to catch design and coding errors early by using assertions (i.e. specific conditional debug test code) and by allowing creation of (mostly) non-intrusive unit tests to achieve -in conjunction with the coverage tool gcov- 100% source code testing by a dedicated unit test suite allowing 100% regression testing after each source code modification. Conscious use of the pre-processor provides a mighty tool and makes the C-language the preferred programming language even for critical software tasks at a moderate level of effort.

The essential requirement of source code coverage analysis for Category A software was covered by the gcc-compiler built-in tool gcov. This tool not only provides source code coverage measuring during development by means of unit tests, but also provides vital information about S/W and H/W-interaction during integration testing too. It allows, for example, even over long periods of time to show whether the error-handling branches of the H/W-accessing routines are accessed or not.

3.6 The test platform

Usage of similar H/W (SPAICE processor board which served as basis for the PDE MPCC development) in conjunction with a hosted OS (VxWorks) which runs the PDE software modules/subroutines as one dedicated task, allows software development and unit testing independent of target system. It has fast turn-around times and greatly shortens development times without interfering with H/W development and sub-system testing.

There has been no S/W testing by means of simulators: the production software running on real target hardware provides early results with a high level of confidence in the source code and the resulting object code without the necessity to keep a simulator in sync with the 'as-built' configuration of the target hardware.

3.7 Object code coverage

"What you code is what you get - always?" Or in other words: does the executable object code really reflect only the intentions specified by the source code or is there more inside the executable object code? That's why ECSS requests for Category A software verification of source code with 100% Modified Condition and Decision Coverage (MCDC), and the coverage of 100% of object code. The tracing of object code coverage on the target or on a SW emulator of the PDE computer would have required a big investment on the platform, so an alternative approach has been used to verify the complete traceability between source code and object code, thus ensuring the object code coverage by the MCDC coverage of source.

As no commercial tool for this task was available, a script-driven LEON-specific analysis method based on the gnu-tool set has been developed in order to trace all executable object code statements to the corresponding source code files and to automatically identify those object code statements not directly traceable to a specific source code line in a dedicated source code file. This task is accomplished by the developed tool chain to an unexpected high level of successful matches (>95%), reducing the necessary 'manual' analysis for the object code coverage analysis to a manageable amount. A first analysis on the current object code stumbled across some generated object code not exactly 4 byte-aligned thus 'fooling' the quite simple analysing tool chain. Object code not traceable to source code has not been found so far.

4 ISVV

One of the implication of the Category A classification is the need to execute Independent Software Verification and Validation, ISVV [6]. The category A classification has been established for the PDE software after the PDR, when the design of the PDE equipment was consolidated enough to allow completion of the RAMS analyses. As a consequence, the ISVV process was not established early enough to cover the activities of Technical Specification Analysis, that have then been covered only by the nominal team, and integrated by a deep review activity carried on by Airbus, ESA, NASA and Lockheed Martin.

The main drivers for the selection of the organization that would have executed the ISVV have been the competences and background, the level of independence, the accessibility to the test platform, and the accessibility of the relevant project documentation without restrictions. Considering these drivers, and considering that the risks were relatively reduced due to the low complexity of the PDE SW, it was decided to assign the ISVV activity to an independent team in the same organization.

The PDE SW ISVV has been concentrated on three tasks: Design Analysis, Code Analysis and Validation.

Due to the nature of PDE SW and the simple design, the Design Analysis has been limited to the verification of timing & sizing budget, based on the delivered results by the software supplier. In the same phase the Code Analysis has been executed, both by code inspection and via static code analysis using the commercial tool *Polyspace Bug Finder*. The static code analysis identified 178 potential defects, and raised a number of recommendations for improvement to the PDE development team. After joint review between Airbus and ESA, a few recommendations have been accepted and implemented to improve the code or the comments. It is considered that the ISVV Code Analysis has been effective in allowing the improvement of the code quality; however, using the same approach on a much larger code would be more challenging, due to the relatively high number of false positive raised by the static code analysis tool.

The final ISVV task has been the independent Validation, executed on the final environment with the software integrated on a PDE EM. The objectives for the independent testing have been defined by the ISVV team in coordination with ESA. As the PDE SW has a relatively reduced number of operational scenarios, it has been decided to focus the independent testing on the non nominal functions, including stress tests, robustness tests, tests at the boundaries, tests with invalid inputs, test extending beyond the domain defined by the requirements. The test are executed independently by the ISVV team on the PDE Engineering Model, and are still ongoing at the time of writing.

5 THE NEAR FUTURE: QUALIFICATION AND FLIGHT

The current status of PDE Software development is the preparation of the PDE Integration Readiness Review. This is the milestone marking the completion of SW development and the delivery to its next step customer - the equivalent of a Software Qualification Review as per ECSS E40. The PDE SW will then be integrated in the

PDE Qualification Model and used for the PDE formal Qualification Campaign, to be completed in the summer 2017 with the PDE Qualification Review.

In parallel, the PDE SW is deployed on the PDE Engineering Models on the avionics test platforms, both in Europe to test the integrated functions at ESM level, and in the US to test the end-to-end chain with the ESM and the CM including the Orion Flight Software. The PDE with its PDE SW will be submitted to a thorough testing that will include verification of external interfaces, data formats, reactivity, functional behaviour in nominal and non nominal cases.

Upon completion of the qualification phase, the first launch of Orion will take place, with a mission around the moon with no crew on board; the first mission with crew is scheduled 2 years later.

6 CONCLUSIONS

This article presents some of the challenges encountered in the development of the PDE Software for the European Service Module, both in the technical field and in the organization and management of the development.

Some aspects presented contains important lesson learned, to be considered in future international cooperation and more generally for development of human spaceflight software.

A first aspect is the harmonization of software standards between the different cooperating organization. So far, different ESA Human Spaceflight projects such as Columbus, ATV and Orion ESM have each followed a different approach. The work for mutual recognition of software standards between ESA, NASA and Jaxa should be completed, and a framework should be made available to ESA projects.

Another lesson learned comes from the process to determine the SW criticality. First, as part of the standard harmonization and mutual recognition a compatibility between US standards and ECSS approaches is needed. Second, the ECSS approach relies strongly on the system level safety analysis, and this delays the categorization in cases where there is some parallelism between system, hardware and software development cycles. The aspect of software criticality should be addressed from the early phases of system development, and set-up both technically and contractually. Changes introduced in ongoing revisions of ECSS can allow to determine earlier the SW criticality category, and to design from a system perspective additional mitigating means such that the overall safety requirements are meet.

Finally, no reference toolchain, let alone a qualified one, is available for development of Category A software compliant to ECSS requirement. The solutions that have been found for the PDE SW are in some cases specific and cannot be applied generically, so the need remains for the definition of a set of tools that can be safely used for development of critical software.

The Category A software is a necessity for Human Spaceflight, as proven by ATV and Orion ESM, and Category A software will be again present on future ESA developments for human exploration systems, that will most probably be carried out in cooperation with other space agencies. The lesson learned from the ATV Project and from the ESM PDE SW development can be considered as a reference to prepare for this future.

ACKNOWLEDGMENTS

Development of critical software in such a complex system as the Orion European Service Module can only be accomplished by working as a team, open to cross-organization contribution, and in interface with all the relevant disciplines at system and subsystem levels. This includes the redaction of the present article, for which precious contribution came from our Airbus and ESA colleagues from different disciplines. The authors thank in particular Florian Bittner, [redacted] and Jens Hartmann (Airbus Defence and Space); David Johnston (CIMS); Lars Oliefka and Jordi Duatis (ESA).

Fig. 1 is courtesy of NASA, Fig. 2 Fig. 3 and Fig. 4 are courtesy of Airbus Defence and Space.

REFERENCES

[1] O. Boudillet, D. Berthelier and D. Dalemagne, “Category A software development for the ATV”, DASIA 2006.

[2] M. Bottacini, J. Grantier, M. Gronowski and Bill Johns, “The European Service Module Contribution to the Orion Program”, IAC 2015

[3] Jan-Hendrik Meiss, Markus Jaeger, Matthias Gronowski, Thierry Kachler, and Kevin Dickens. “Evolution and Status of the Orion-ESM Propulsion Subsystem“, AIAA SPACE 2016, AIAA SPACE Forum, (AIAA 2016-5622)

[4] Mutual Recognition of S&MA Standards Software Assurance Task Force Report - NASA/ESA/JAXA Trilateral Meeting September 2012

[5] Jens Hartmann, Bernd Wolff, “Deterministic High Speed Data Communication in Space - MPCV ESM Overview and related development“, ADCSS 2013

[6] ESA Guide for Independent Software Verification & Validation, Issue 2, 29.12.2008

[7] ECSS-Q-HB-80-02 SW Process Assessment Handbook