estec

European Space Researchand Technology CentreKepl
T +31 (0)71 565 6565F +31 (0)71 565 6040w w w

DOCUMENT

Guidelines for the Automatic Code Generation for AOCS/GNC Flight SW Handbook: Volume 1 - General concepts

APPROVAL

Title Guidelines for the Automatic Code Generation for AOCS/GNC Flight SW	
Issue 1	**Revision** 0
Author [Author]	**Date** 15-Jun-21
Approved by	**Date**

CHANGE LOG

Reason for change	Issue	Revision	Date
First issue	1		

CHANGE RECORD

Issue 1	**Revision** 0		
Reason for change	**Date**	**Pages**	**Paragraph(s)**

Table of contents:

1 INTRODUCTION

Automated Code Generation (ACG) is more and more used on ESA missions/projects and in particular it is almost becoming baseline process for the AOCS & GNC flight SW application. The ACG has been already discussed in several forums and tackled from different viewpoints, concerning the modelling, the code quality, the validity of the tools, the process applied to the simulation models, etc..

The purpose of this document is to provide modelling rules and guidelines to develop AOCS & GNC models using MATLAB/SIMULINK in order to ensure the generated code being functionally correct, compliant with the existing standards as well as readable, reusable and maintainable. The document provides guidance on how to comply with ECSS standards [AD-01] and [AD-02] tailored for software criticality B. Compliance to the additional requirements specific to software criticality A is not addressed in this handbook and would require further analysis and considerations.

This document will also provide rules and guidelines to drive the user through the development, verification and validation process for AOCS automatic generated FSW being fully compliant with applicable standards.

Note that the nomenclature AOCS and GNC in the context of this handbook shall be understood as interchangeable.

This document is divided in two parts, one "process oriented" that is independent of the specific toolchain used and a part which gives guidelines that apply to a specific toolchain. In the current issue this second part is based on the use of MATLAB 2017a and the following releases for the supporting toolboxes:

Toolbox Used	Version	
MATLAB	Version 9.2	(R2017a)
SIMULINK	Version 8.9	(R2017a)
Embedded Coder	Version 6.12	(R2017a)
MATLAB Coder	Version 3.3	(R2017a)
MATLAB Report Generator	Version 5.2	(R2017a)
SIMULINK Coder	Version 8.12	(R2017a)
SIMULINK Report Generator	Version 5.2	(R2017a)
SIMULINK Verification and Validation[1]	Version 3.13	(R2017a)
Stateflow	Version 8.9	(R2017a)

[1] As of R2017b, the SIMULINK Verification and Validation product transitioned into SIMULINK Requirements, SIMULINK Check and SIMULINK Coverage. For more details: https://de.mathworks.com/products/transitioned/simverification.html.

This does not mean that the intention of the HB is to impose the use of the specific version listed above, but is a fact that the rules could only defined with a specific version as reference. The validity of reported guidelines for newer or different version of the listed toolboxes or even MATLAB/SIMULINK shall be assessed, notified and updated as necessary considering that the rational that drives the rule is version independent.

2 APPLICABLE AND REFERENCE DOCUMENTS

2.1 Applicable

REF	CODE	TITLE
AD-01	ECSS-E-ST-40C	ECSS Software
AD-02	ECSS-Q-ST-80C Rev. 1	ECSS Software Product Assurance
AD-03	ECSS-E-ST-60-30C	ECSS Satellite AOCS requirements

2.2 Reference

REF	CODE	TITLE
RD-01	GMV-AUTOCOGEQ-FR_V1.1	AUTOCOGEQ ((Preparation for the Qualification of Auto-Code Generated from SIMULINK Models) Final Report
RD-02	ESA-TECSAA-TN-007001	AOCS Flight SW Automatic Code Generation process
RD-03	ESA-TECSAA-TN-011350	AOCS FSW AUTOCODING Verification Process Final Report

3 TERMS, DEFINITION AND ABBREVIATED TERMS

3.1 Terms and Definitions

The following terms are defined into the scope of this document.

- GNC models/algorithm: MATLAB/SIMULINK model representing the GNC/AOCS part
- GNC SW: code generated from the model
- OBSW: SW considering also modules that are not part of GNC/AOCS
- Embedded MATLAB refers to MATLAB function used as source code for SIMULINK block

3.2 Abbreviated tems

Acronym	***Description***
ADD	*Architectural Design Document*
ASIC	*Application Specific Integrated Circuit*
ATB	*Avionics Test Bench*
CAN	*Controller Area Network*
CDR	*Critical Design Review*
COTS	*Commercial Off-The-Shelf*
DDD	*Detail Design Document*
DDR	*Detailed Design Review*
DDVV	*Design, development and Verification and Validation*
DKE	*Dynamic and Kinematic Environment*
EML	*Embedded MATLAB*
FES	*Functional Engineering Simulator*
FPS	*Flight Program Software*
FR	*Final Review*
FSW	*Flight Software*
GNC	*Guidance Navigation and Control*
HIL, HWIL	*Hardware In the Loop*
HW	*Hardware*
ICD	*Interface Control Document*
LV	*Launch Vehicle*
MIL	*Model In the Loop*
M/S	*MATLAB/SIMULINK*
N.A.	*Not Applicable*
NCR	*Non Conformity Report*
OBC	*On Board Computer*

OS	*Operating System*
PDR	*Preliminary Design Review*
PIL	*Processor in the Loop*
PM	*Progress Meeting*
RTOS	*Real Time Operating System*
SCMP	*Software Configuration Management Plan*
SDD	*Software Design Document*
SDP	*Software Development Plan*
SoC	*Statement Of Compliance*
SPR	*Software Problem Report*
SRD	*SW Requirement Document*
STB	*Software Test Bench*
SUM	*Software User Manual*
SVF	*Software Validation Facility*
SIL, SWIL	*Software In the Loop*
SW	*Software*
TBC	*To Be Confirmed*
TBD	*To Be Defined*
TBI	*To Be Issued*
TBW	*To Be Written*
TRR	*Test Readiness Review*

4 INTRODUCTION TO AOCS FSW DEVELOPMENT PROCESS

The process to design and implement AOCS/GNC algorithms it is nominally performed through the following main steps:

- Derivation of AOCS/GNC requirements from mission/system requirements
- Prototyping AOCS/GNC algorithms in MATLAB/SIMULINK
- Verification of AOCS/GNC requirements on Functional Engineering Simulator (FES) Model in the Loop (MIL)
- AOCS/GNC Flight SW requirements specification (which may also affect the modelling) – sometimes included in the Satellite URD
- Generation of the AOCS/GNC FSW (Coding)
- Verification of AOCS/GNC requirements on Functional Engineering Simulator (FES) Software in the Loop (SIL)
- Completion of verification of functional requirements and SW performance on Processor in the Loop (PIL), HW in the Loop (HWIL), Avionics Test Bench, Satellite

The design and implementation of the AOCS/GNC models is performed in MATLAB/SIMULINK where a Functional Engineering Simulator (FES) is built to verify the AOCS/GNC algorithms and main functionalities. Then coding is performed to generate the AOCS/GNC SW. The auto generated code is embedded and tested in the FES environment via Software In the Loop (SIL) to verify the successful conversion of models to code from functional point of view. The AOCS/GNC SW is then embedded in a Processor In the Loop (PIL) test bench that runs the SW in a space representative environment of the on-board computer to validate the algorithms and SW performance in flight realistic conditions.

The DDVV involved tasks are running in parallel (design and development of the OBSW) and also interleaved (verification and validation steps at different levels) all along the OBSW development activity. The development of the space flight critical software it is nominally performed following the SW lifecycle that accounts for different phases according to the ECSS-E40C ([AD-01]) standard such as:

- SW Specification
- SW Design
- SW Implementation
- SW Verification and Validation (V&V)

The definition of the test facilities used all along the DVV Process by AOCS/GNC team and/or by SW team depending on development phase are reported in the Table below.
It has to be noted that in order to increase efficiency of the process, some test facilities share the models used for the external environment, dynamics and units as well as the test definition and in some cases the test harness.

Configuration	Test facility	Purpose/Comment
MIL	FES Functional Engineering Simulator	Model of the GNC algorithms implemented in a simulation framework (e.g. MATLAB/SIMULINK). The FES (Functional Engineering Simulator) is a simulator of the spacecraft from the AOCS standpoint. MIL tests are used for tasks as: • Support the system requirement consolidation • Validate the key algorithms needed in the system (e.g. GNC, …) • Trade-off different design alternatives • Verify the system performance through a set of analyses (e.g. perturbation analysis, covariance analysis, Monte-Carlo analysis, worst-case analysis).
SIL	FES Functional Engineering Simulator	A software is produced from a model/architecture (either manually or with autocoding). It can be only the GNC or the full OBSW. The term "Software In the Loop" refers to Connecting the GNC software with the simulator of the spacecraft With the GNC software, SIL tests are used mainly for verification of GNC performances connecting the GNC SW to the FES. This step (a further step after MIL tests) demonstrate that the "real" GNC SW implementation still complies to the GNC requirements).
PIL	STB SW Test bench	It is a specific case of HWIL when the Avionics equipment present is the OBC (only real). The SW produced is executed on real target OBC, while connected to the simulator of the spacecraft (normally called RTS, Real Time Simulator) The purpose of this facility is the verification computing budget usage (e.g. timing, memory) of the GNC SW in realtime, with the real OBC
SVF	SW Validation Facility	The SW produced is Executed with the full software into a model (real or simulated) of the On Board Computer (usually called SVF – Software Validation Facility) The purpose of this facility is the validation of OBSW against TS/RB requirements
HWIL	ATB Avionics Test Bench	A software is produced from a model/architecture (either manually or with autocoding). The term "Hardware In the Loop" refers to the fact of executing this software into the real Avionics (OBC plus some equipment). A real time spacecraft simulator is usually present, "closing the loop".

Table 1: Test facilities definition

4.1 The classical development process: Manual Coding

The development process for the AOCS SW foresees different development branches with clear split between AOCS Engineering tasks and SW Engineering tasks.

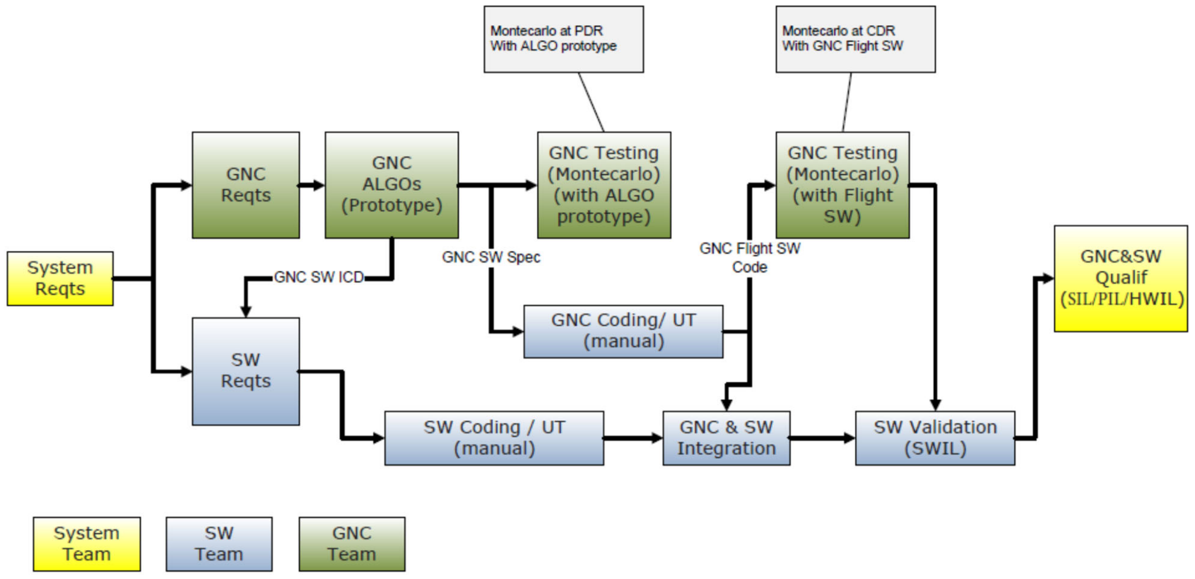


Figure 4.1-1: GNC and SW Development process with Manual coding

The process is as follows:

- The GNC requirements and the SW requirements are derived from the System requirements by each team

Then, in parallel:

- The GNC team:
  - Writes the GNC algorithm prototype (in MATLAB/SIMULINK, Fortran, any other), which generally is considered design simulator and not deliverable
  - Writes a GNC SW ICD for – and together with – the SW team (GNC task processing needs, high level interface of the algorithms)
  - The GNC algorithms prototype models are submitted to a first GNC testing campaign (Montecarlo in SWIL – *actual SW in the loop on emulated target processor*). The objective is to check that the algorithms itself is valid and to verify the performance
  - Writes a GNC Specification for the SW team (algorithm specification, GNC user requirements document)

- The SW team:
  - Starts the development of other parts of the SW
  - When the GNC SW Spec is received starts the manual translation or re-coding of the GNC algorithms.
  - The GNC code is unit tested (by the SW team with the GNC support)

When the GNC Unit testing is finished:

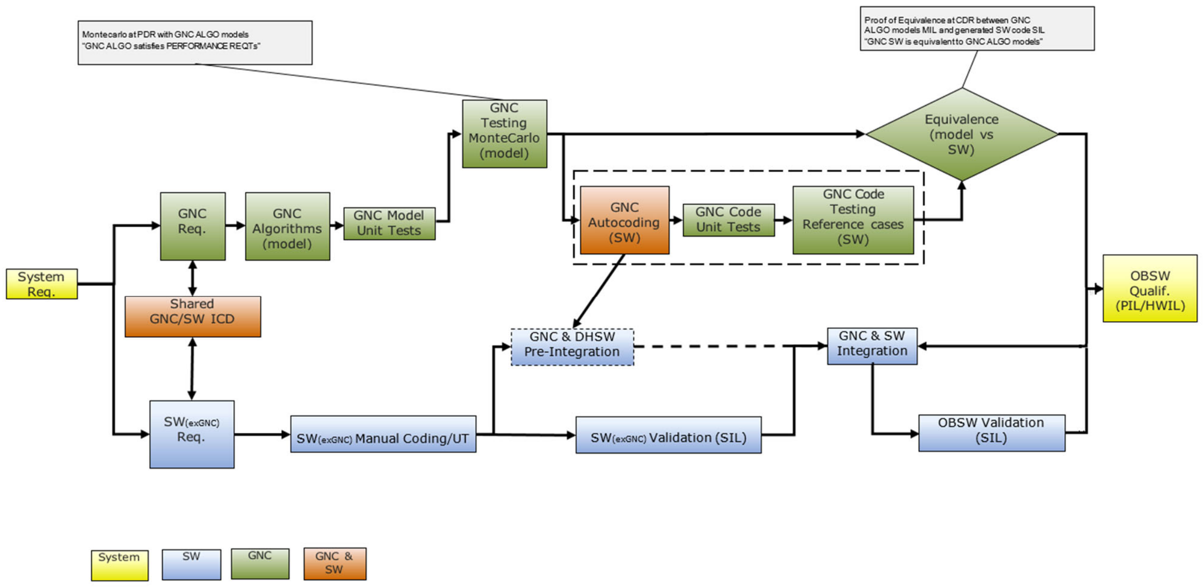
- The SW team delivers the GNC code to the GNC team *[The SW team can also deliver a wrapper of the SW part so the GNC team can run it on host]*
- The GNC team performs a Final GNC Testing (Subset of representative test cases or where possible Montecarlo in SWIL *[The SWIL can be run on host or on simulated target. Running on real target a Montecarlo test campaign is in general not possible]*) with this GNC SW
- When the GNC testing is finished the GNC team gives the go-ahead to the SW team to integrate the GNC code with the rest of the OBSW and validate it against the technical specification (typically in a representative target, but open loop)

Finally, the OBSW is qualified in a representative Avionics environment in PIL (might be SIL SVF for some cases) and HWIL tests. Both GNC and SW team collaborate in writing the Qualification Test Specification and checking the results of the test campaign. (It shall be checked not only that the GNC requirements are fulfilled, but also that the behaviour is compatible with the one of the GNC SWIL testing).

4.2 The development process with Autocoding

The use of Autocoding implies some changes during the development, driven by the earlier availability of the GNC SW for the GNC team and by the different assignment of tasks between GNC and SW teams. In the following two different approaches are presented. The first one is considered the preferable one and is presented in more detail. The alternative one is presented in a more synthetic way focusing on the differences with the first one.

The following drawing summarises an example of the GNC SW development process with Autocoding:



Montecarlo at PDR with GNC ALGO models
"GNC ALGO satisfies PERFORMANCE REQTs"
Proof of Equivalence at CDR between GNC ALGO models MIL and generated SW code SIL
"GNC SW is equivalent to GNC ALGO models"
GNC Testing MonteCarlo (model)
Equivalence (model vs SW)
GNC Req.
GNC Algorithms (model)
GNC Model Unit Tests
GNC Autocoding (SW)
GNC Code Unit Tests
GNC Code Testing Reference cases (SW)
System Req.
Shared GNC/SW ICD
OBSW Qualif. (PIL/HWIL)
GNC & DHSW Pre-Integration
GNC & SW Integration
SW(exGNC) Req.
SW(exGNC) Manual Coding/UT
SW(exGNC) Validation (SIL)
OBSW Validation (SIL)
System
SW
GNC
GNC & SW

Figure 4.2-1: GNC and SW Development process with Autocoding

At the beginning of the process the following documents will be prepared (it is also possible to merge more elements in single document):

- ICD: this is a joint work between GNC and SW team with the architectural choices driven by the SW expert that takes into account the data flow, frequency of calling required by the GNC specification. The ICD needs to define a set of automatic code generator settings compatible with the particular operating system used
- Model Requirements Specification (MRS): this document is used to design and implement GNC Model of the subsystem Algorithms (based on the GNC Requirements Specification).

The MRS together with the ICD represent the AOCS Requirements Baseline to be delivered and reviewed at SW SRR

The default approach supported by this HB is that auto-generated GNC code shall not be manually modified at any level, i.e. in case of any bug during the verification the process shall go back to Model development and FES MIL verification. This point implies that the GNC Model represents the AOCS SW Technical Specification and it shall be delivered (in model source code) as part of the GNC code release note at SW PDR.

It has to be noted that the above figure use as reference two separate teams. This is done to cover the scenario in which some of the activities are done for instance by different companies of in companies with a rigid role allocation. In the case activities are all executed within the same integrated AOCS/SW team, the concept of teams is not fully appropriate, but the current team allocation/description still identify tasks/activities that need to be carried out in order to ensure a correct flow of information and process documentation.

4.2.1 The development process with Autocoding and with proof of equivalence

In the following schema it is mapped the development process of the GNC code wrt the SW development cycle and main reviews.

N	Name	Description	Facilities
(1)	AOCS Requirements Definition	Derivation/apportionment of AOCS requirements from System requirements. • Model Requirements Specification (MRS) replacing the Requirement Baseline defining AOCS modelling requirements • AOCS/SW ICD shall be developed together by AOCS and SW team	Analyses tools (e.g. SIMULINK Requirements)
(2)	AOCS Algorithms Development (AOCS Models)	Development and modelling of the AOCS algorithms (following the modelling guidelines) Note: in parallel the models for external world (DKE, Sensors, Actuators, Environment) shall be developed and made available for the performance tests	Modelling tools (e.g. MATLAB, SIMULINK)
(3)	AOCS Models Unit Tests (AOCS Models)	The developed algorithms are subjected to Unit Testing where they are checked against modelling standard guidelines (generation of C-code) and open loop tests performed to be used as reference cases. Coverage tests are also performed to ensure sufficient model coverage with selected test harness. It needs to be reminded that model coverage does not imply coverage of the generated code. Note: the AOCS Models are used as Technical Specification (TS) for the automatic SW code	Modelling tools (e.g. MATLAB, SIMULINK)

		generation, in the sense that SW behaviour will be validated against the Model behaviour	
(4)	AOCS Performance verification (AOCS Models)	AOCS team runs performance verification campaign (normally Monte Carlo) to verify the compliance with AOCS performance requirements	Modelling tools (e.g. MATLAB, SIMULINK)
(5)	AOCS Code Generation (AOCS Models to AOCS SW)	Automatic Generation of the AOC SW Code	Coder tools (e.g. Embedded Coder)
(6)	AOCS Code Test (AOCS SW)	The (auto)generated code is subjected to several tests to verify the correctness of code generation process. This consists of code generation guidelines compliance testings and coverage testing to ensure sufficient code coverage with selected test harness.	Modelling tools (e.g. MATLAB, SIMULINK)
(7)	AOCS Code preliminary integration (AOCS SW not verified)	The generated AOCS Code is delivered to SW team for pre-integration tests with the other On board SW modules. The feedback from pre-integration test is sent back to AOCS team to be implemented in the Model before final delivery (derisk of integration issues after final delivery)	SW Testing Environment
(8)	Proof of Equivalence test (AOCS Model & AOCS SW)	The Proof of Equivalence test is aimed to verify that the generated code behaves as the Model at numerical precision level. The test compares reults from representative (sufficient coverage – metrics to be agreed within project – shall be granted together with sufficient excitation of functionalities) set of reference open loop tests cases run on the same test environment (e.g. MATLAB MIL and SIL) using AOCS Models and AOCS SW. This corresponds to the validation of generated code wrt Technical Specification (TS) represented by AOCS Models Note: in case the test is not succesful (i.e. some differences cannot be explained) or as an alternative the AOCS Generated Code shall be submitted to full AOCS Performance campaign (e.g. Monte Carlo) on the SIL to verify the compliance with AOCS performance requirements.	Test Environment MIL and SIL (e.g. MATLAB, SIMULINK)
(9)	AOCS Code Verification and Integration	The generated and verified Code is delivered to the SW team to undergo the SW Unit Tests and SW performance tests as per standard SW	SW Testing Environment (SVF, PIL, HWIL)

	(AOCS SW)	development plan and the integrated in the On Board SW (OBSW) for further qualification and acceptance testing before final delivery to system for functional verification (as per standard process). It is noted that unit test a SW level have a different scope than unit test at model level (3). The UT at model level are intended to address modelling aspect while the one at SW level are intended to address code coverage and robustness approach as per E40C 5.5.3.2. Between the two set of UT there is a certain level of overlap and the scope of the SW UT can be reduced if it can be shown that certain aspects are addressed in the model UT. It is also noted that current process allocate the unit testing at SW level to the SW team, but depending on team expertise and tool chain it could also be conceivable to allocate this activity to the AOCS team.	

Table 2: GNC development code map to SW main reviews

The Figure below maps also the development process into the main reviews and shows the evolution of the process in time and per phase for AOCS development and for SW development. The reviews include both GNC and OBSW reviews and they might not coincide in time. All the indicated steps indicated are considered mandatory.

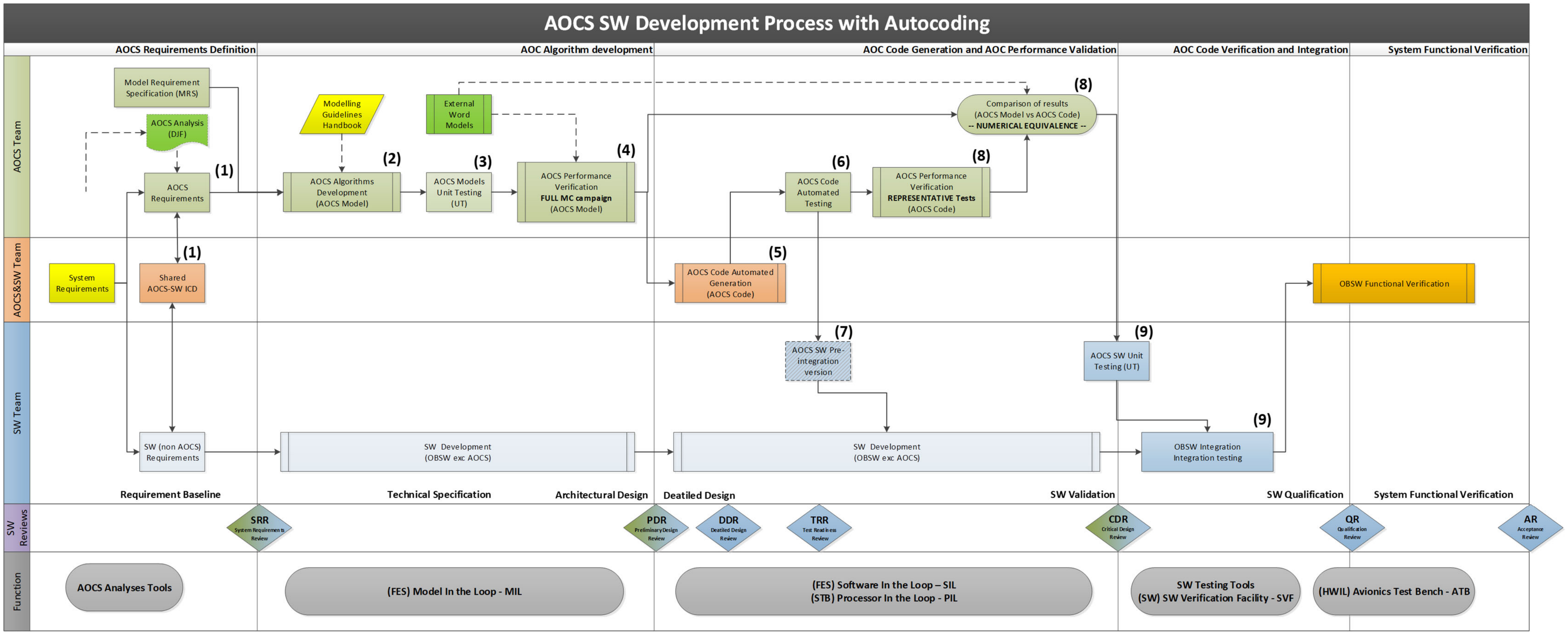

AOCS SW Development Process with Autocoding
AOCS Requirements Definition
AOC Algorithm development
AOC Code Generation and AOC Performance Validation
AOC Code Verification and Integration
System Functional Verification
AOCS Team
AOCS&SW Team
SW Team
SW Reviews
Function
Model Requirement Specification (MRS)
AOCS Analysis (DJF)
AOCS Requirements
(1)
Modelling Guidelines Handbook
AOCS Algorithms Development (AOCS Model)
(2)
External Word Models
AOCS Models Unit Testing (UT)
(3)
AOCS Performance Verification FULL MC campaign (AOCS Model)
(4)
Comparison of results (AOCS Model vs AOCS Code) -- NUMERICAL EQUIVALENCE --
(8)
AOCS Code Automated Testing
(6)
AOCS Performance Verification REPRESENTATIVE Tests (AOCS Code)
(8)
System Requirements
Shared AOCS-SW ICD
(1)
AOCS Code Automated Generation (AOCS Code)
(5)
OBSW Functional Verification
AOCS SW Pre-integration version
(7)
AOCS SW Unit Testing (UT)
(9)
SW (non AOCS) Requirements
SW Development (OBSW exc AOCS)
SW Development (OBSW exc AOCS)
OBSW Integration Integration testing
(9)
Requirement Baseline
Technical Specification
Architectural Design
Deatiled Design
SW Validation
SW Qualification
System Functional Verification
SRR System Requirements Review
PDR Preliminary Design Review
DDR Deatiled Design Review
TRR Test Readiness Review
CDR Critical Design Review
QR Qualification Review
AR Acceptance Review
AOCS Analyses Tools
(FES) Model In the Loop - MIL
(FES) Software In the Loop – SIL
(STB) Processor In the Loop - PIL
SW Testing Tools
(SW) SW Verification Facility - SVF
(HWIL) Avionics Test Bench - ATB

Figure 4.2-2: Full GNC and SW Development process with Autocoding
***Note** that AOCS and GNC in this context shall be understood as fully equivalent*

4.2.2 The development process with Autocoding and without proof of equivalence

A variation of the process described before focus on the possibility to not perform the proof of equivalence. Removing that step is possible, but has drawbacks and is up to the project to do the appropriate trade-off.

The process without proof of equivalence is depicted in Figure 4.2-3, and if one compare this with Figure 4.2-2 two aspects should be highlighted

On one side proof of equivalence (8) has been removed, but on the other side the AOCS performance campaign (4) is now split in two parts. A preliminary performance verification (4a) done on the models, and a full Montecarlo campaign (8a) done on the SIL. The scope and the level of the preliminary performance verification will have to be agreed and defined in each project, but it must be understood that the scope of this preliminary campaign is to reduce the risk for the SW development team and allow early integration of the auto generated code. In theory it could be possible to remove this preliminary performance campaign, but that would require either a not acceptable risk for the SW development team, or postponing the SW activities till the completion of the full Montecarlo campaign (8a) and this would most probably deny the advantages in terms of schedule that comes from the use of autocoding.

The preliminary performance campaign (4a) is also needed to provide inputs to the PDR, which means that removing this campaign would also create problem in defining the scope of the GNC part of the PDR.

All the rest of the activities, remains essentially the same, and is hence not discussed again in this section.

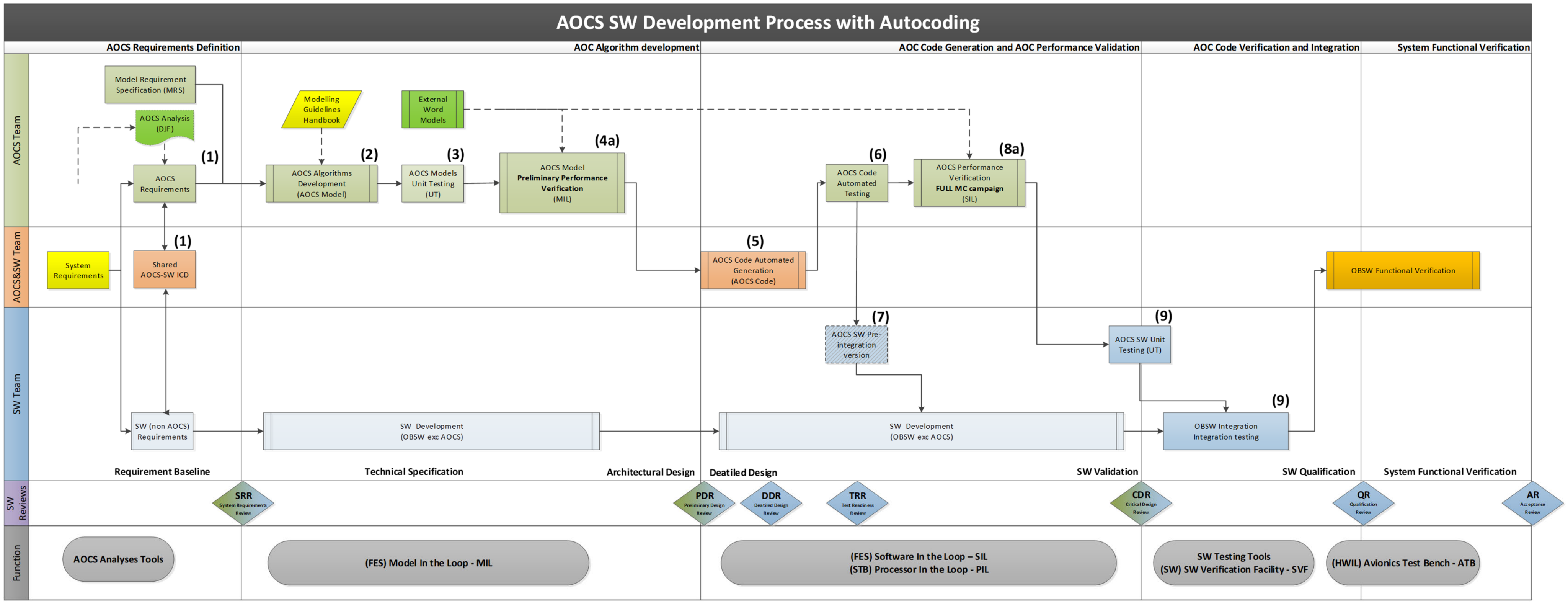

AOCS SW Development Process with Autocoding
AOCS Requirements Definition
AOC Algorithm development
AOC Code Generation and AOC Performance Validation
AOC Code Verification and Integration
System Functional Verification
AOCS Team
AOCS&SW Team
SW Team
SW Reviews
Function
Model Requirement Specification (MRS)
AOCS Analysis (DJF)
AOCS Requirements
(1)
Modelling Guidelines Handbook
External Word Models
AOCS Algorithms Development (AOCS Model)
(2)
AOCS Models Unit Testing (UT)
(3)
AOCS Model Preliminary Performance Verification (MIL)
(4a)
AOCS Code Automated Testing
(6)
AOCS Performance Verification FULL MC campaign (SIL)
(8a)
System Requirements
Shared AOCS-SW ICD
(1)
AOCS Code Automated Generation (AOCS Code)
(5)
OBSW Functional Verification
AOCS SW Pre-integration version
(7)
AOCS SW Unit Testing (UT)
(9)
SW (non AOCS) Requirements
SW Development (OBSW exc AOCS)
SW Development (OBSW exc AOCS)
OBSW Integration Integration testing
(9)
Requirement Baseline
Technical Specification
Architectural Design
Deatiled Design
SW Validation
SW Qualification
System Functional Verification
SRR System Requirements Review
PDR Preliminary Design Review
DDR Deatiled Design Review
TRR Test Readiness Review
CDR Critical Design Review
QR Qualification Review
AR Acceptance Review
AOCS Analyses Tools
(FES) Model In the Loop - MIL
(FES) Software In the Loop – SIL
(STB) Processor In the Loop - PIL
SW Testing Tools
(SW) SW Verification Facility - SVF
(HWIL) Avionics Test Bench - ATB

Figure 4.2-3: Full GNC and SW Development process with Autocoding without proof of equivalence
***Note* that AOCS and GNC in this context shall be understood as fully equivalent**

4.2.3 Reviews

It should be noted that both Figure 4.2-2 and Figure 4.2-3 present a line with the reviews and their phasing with the activities. Some of those reviews (SRR/PDR/CDR) are to be considered joint GNC and SW reviews, while other are pure SW reviews (DDR/TRR/QR/AR). Since AD-01 only covers the typical review objectives for the SW activities, the overall development plan should define the objectives of those joint reviews taking into account both the SW and the GNC needs.

If it is found more convenient not to held the review as joint reviews, both GNC and SW should still held them, but this creates dependencies that is hard to solve, so the two reviews should be as near in time as possible, with the GNC review occurring before the SW review.

4.3 Comparison and key differences

Disadvantages of the classical process:

1. Bugs introduced during manual coding: different interpretation of requirements. The process foresees exchange of user requirements in descriptive form, which is subject to human interpretation and prone to different interpretations and implementations leading to different results (The SW expert is not required to be aware about all the details of the GNC algorithms). The GNC algorithms and the GNC codes are different and identification of the discrepancy can be tricky and time consuming. This can be avoided having the code generated automatically as the Models are directly translated into Code without intermediate steps.

2. Late detection of incompatibility of the GNC algorithms with the flight HW. The GNC expert is not aware of the stringent constraints of the flight HW. The SW expert sees late the GNC specification when the GNC algorithms are almost complete. Some algorithms could need to be split, changed. This can be avoided by forcing the introduction of a formal GNC/SW ICD including timings early into the Project and using appropriate tools during development to verify compatibility with such requirements.

3. Costly introduction of changes. If the GNC team introduces a change late in the development once the Code has been already written, the impacted algorithms in the code shall be manually implemented and the code re-tested. The iterations “change of model / generate code / testing” can be done much faster with Autocoding, reducing the effort and cost.

4. The planning and schedule for manual coding implies longer time from validated GNC models to the C code generation, ready for verification (up to 6 months). However it has to be considered that less time/effort is required for model development (wrt Autocoding)

Disadvantages of the Autocoding process:

1. Unless clear and robust modelling guidelines adopted, there is a risk to generate code which higher CPU load and CPU occupation. Note that this should be not a problem if enough HW resources are available. Some guidelines have been defined to improve this aspect (e.g. structure, number of blocks, etc…)

2. Unless clear and robust modelling guidelines adopted, the generated code is more difficult to read that classical code (longer, mangled names for generated variables, functions…). Note that some guidelines are introduced later for managing this point.

3. High level model constructs can be translated to unexpected statements in the generated code. Minimization by implementing the testing process as per this HB; the use of automatic inspection tools can mitigate this problem.

4. Dependence from the development environment (e.g. MATLAB version to be maintained possibly for long projects). Possible mitigations are the implementation of portability across different environment versions.

5. Difficulty in tracing the link between SW functions and model blocks. Less control on generated code in case of bugs detected during SW verification involving group of functionalities or specific aspects of running code (e.g. memory usage or allocation). Mitigations come from the usage of specific tools to trace code lines to model blocks and from implementation of specific guidelines from this HB.

6. Difficulty of SW Unit Testing. The structure of the code (functions, names, etc…) is determined by the code generator, and it requires some analysis to determine, for example, the purpose of each function and execution branch or the valid input ranges. It is not possible, at UT level, to test each function against its specification, as no specification exists at this level. Moreover, changes in the model may lead to changes in the structure and names in the generated code, which complicate the usage of automated UT tools.

7. Limited control of the resulting code regarding coding rules and metrics. The control is indirect through the heuristics provided in this HB.

It is to be noted that some of the aspects listed above can be considered risks for the project and that some of the guidelines in this handbook aim at reducing those effects and the related risks.

The following figure reports the comparison of the GNC SW development cycle with manual coding and autocoding (also reported in figure Figure 4.2-2.

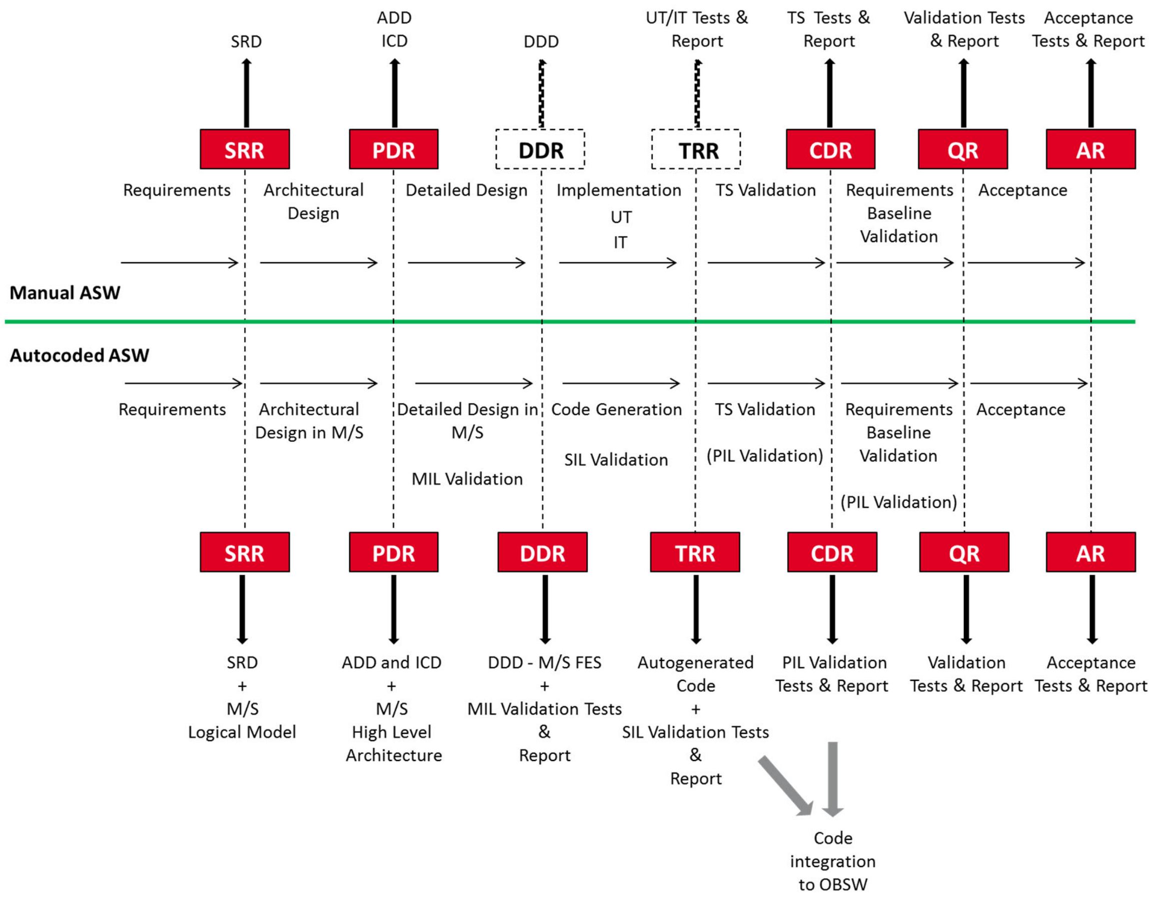


Figure 4.3-1: AOCS/GNC SW development lifecycle: Autocoding vs Manual

4.4 Maintenance

The change of process to introduce autocoding and the implicit decision to assume a certain level of independence between the AOCS team and the SW team has also implication on the SW maintenance approach. One can see the overall SW production as equivalent to a situation in which two different SW developers contribute to the development of the SW and each retains the responsibility of the SW modules it has developed. This means also seeing the AOCS autogenerated code as an independent SW product with its own configuration and development environment.

With this point of view it is clear that both the AOCS team and the SW team must define their maintenance procedures in case evolution or correction of the SW are needed. The SW team would in addition need to explicitly consider the case in which despite not having any changes in the parts it has developed it needs to integrate into the overall SW new sources coming from the AOCS team.

Taking this point of view means deciding that also during the maintenance phase the auto-generated code will not be manually edited but always modified via modification of the simulation model and re-execution of the code generation process.

The main advantage of this approach is ensuring that at any point in time model and code are kept aligned and each of the team maintains its own processes with no major difference between development and maintenance. This approach ensure also that unit test done with the support of the simulation environment (see Figure 4.2-2 box 6) can be repeated as needed for the modified items.
It has to be noted that from the AOCS team point of view this can be considered different from the classical development model as it implies ensuring that the overall environment used to autogenerate the code must be maintained through the all operational life of the SW in the same way as the SW team is used to maintain the SW development and validation environment and a type of activities that goes further than the classic engineering support.
It is acknowledged that in theory is possible to manually edit the autogenerated code but this is considered not advisable (regardless if done by the AOCS team or the SW team) and should not be taken as the baseline option.

5 TRACEABILITY TO ECSS-E-40C, ECSS-Q-80C REV. 1 FOR AUTOCODING

With reference to RD-02, TN1-A_V1.0 Section 8:

This Handbook defines a SW lifecycle that accounts for SW models-based design (in MATLAB/Simulink environment) and autocoding techniques for generating an AOCS/GNC SW production code of category B ready to be embedded into target or simulation facilities (e.g. SIL, PIL, etc.). A methodology has been defined for generation of production code that aims to standardize and guarantee the process and transformation of AOCS/GNC models prototype from MATLAB/Simulink into the final production code for embedded systems. In this context the ECSS standard for space applications shall be applied all along the SW development lifecycle for ensuring that the processes and standards for software of criticality category 'B' code are respected.
Even if there are some requirements in the ECSS Software standards (ECSS-E40 and ECSS-Q80) and a guideline for its use (ECSS-E-HB-40A), the use of model based SW design and autocoding impacts on the ECSS standards. This section analyses the ECSS requirements from ECSS-E40 and ECSS-Q80 to assess the compliance of the SW development proposed with the ECSS standard focusing on the aspects that the model based and autocoding approach impact on the ECSS.

Automatic code generation is addressed in the ECSS-E40 standard clause 5.3.2.4 about the SW Development plan as well as in 5.4.2.2 for the Technical Specification requirements related to autocoding. In the ECSS-Q80 standard, automatic code generation is addressed in clause 6.2.8 for the selection of automatic code generation tools, definition and verification of modelling standards and testing and code requirements.
Therefore, some aspects in the ECSS standard have been analysed and listed in following tables to assess the compliance of the approach of model-based SW development with autocoding to generate on-board SW category 'B'.

The compliance of the ECSS standards for the SW development involving autocoding comes from the analysis of the difference in the SW design process between the model-based approach vs traditional manual SW design, and between the code generated automatically via a tool vs manual SW code development.
In the context explained above, the following sections provide guidance on how to comply with [AD-1] and [AD-2] in a model-based OBSW development and automatic code generation project. The requirements not addressed in the following sections, still remain applicable (subject to each particular project tailoring).

5.1 Traceability to ECSS-E-40C

5.1.1 *SW Related System Requirement Process (5.2.2)*

Clause	Description	Compliance
5.2.2.1	**Specification of system requirements allocated to software** a. The customer shall derive system requirements allocated to software from an analysis of the specific intended use of the system, and from the results of the safety and dependability analysis. EXPECTED OUTPUT: The following outputs are expected: a. Functions and performance system requirements allocated to software [RB, SSS; SRR]; b. Verification and validation product requirements [RB, SSS; SRR]; c. Software operations requirements [RB, SSS; SRR]; d. Software maintenance requirements [RB, SSS; SRR] e. Requirements for in flight modification capabilities [RB, SSS; SRR]; f. Requirements for real-time [RB, SSS; SRR]; g. Requirements for security [RB, SSS, SRR]. h. Quality requirements [RB, SSS, SRR]	AOCS/GNC subsystem responsible a. Models can express functionality and interface requirements. Other outputs (for example performance, operational, maintenance, security, safety and verification requirements – outputs b to h) should follow a classical approach. SW requirements as performance requirements can be derived from the model by running tests over the generated code (SIL&PIL).
5.2.2.2	**Identification of observability requirements** a. The customer shall specify all software observability requirements to monitor the software behaviour and to facilitate the system integration and failure investigation. EXPECTED OUTPUT: System and software observability requirements [RB, SSS; SRR].	AOCS/GNC subsystem responsible The SW observability requirements must be implemented in the model since the code will be automatically generated from the model. A mechanism for monitoring internal model variables into the final software must be considered at modelling level, otherwise variables that need to be observed in the code will be hidden.

5.1.2 *SW Management Process (5.3.2)*

Clause	Description	Compliance
5.3.2.4	**Automatic code generation** a. The autocode input models shall be reviewed together with the rest of the software specification, architecture and design. NOTE The autocode input models are integral part of the software specification, architecture and design. EXPECTED OUTPUT: Autocode input model review [MGT, SDP; SRR, PDR]. b. In the case of coexisting autocoded and manually written parts, the software development plan shall include the definition of a clear interface definition and resource allocation (memory, CPU) at PDR. EXPECTED OUTPUT: Autocode interface definition and resource allocation [MGT, SDP; SRR, PDR]. c. The input model management, the code generation process and supporting tools shall be documented in the SDP. EXPECTED OUTPUT: Automatic code generation development process and tools [MGT, SDP; SRR, PDR]. d. The supplier shall define in the SDP the verification and validation strategy for automatic code generation as a result of the trade off between the qualification of the code generation toolchain and the end to end validation strategy of the software item, or any combination thereof, in relation with ECSS-Q-ST-80 clause 6.2.8. EXPECTED OUTPUT: Automatic code generation verification and validation strategy [MGT, SDP; SRR, PDR]. e. The configuration management of the automatic code generation related elements shall be defined in the SCMP. EXPECTED OUTPUT: Automatic code generation configuration management [MGT, SCMP; SRR, PDR].	a. Proposed in this HB: the model is part of the PDR, DDR reviewed by joint GNC/SW teams b. As proposed in this HB. In particular a SW/SW ICD between manual SW/GNC models and autocoded SW shall exist and be submitted to PDR c. This HB provided useful inputs for such Software Development Plan d. Qualification of the code generator is complex. Instead, this HB provide inputs for producing automated “qualifiable” code e. As discussed when speaking of maintenance the autogenerated code should be considered an independent SW product. This implies the AOCS team is responsible to document the approach to configuration management of model options, model tool chain and of the generated code. The SW team is responsible to clearly describe in its own SW Configuration Management Plan how it deals with the deliveries of the autogenerated code. While in theory it would be possible to include all the information in the SW Configuration Management Plan produced by the SW team this is not considered compatible with the point of view of this HB that considers SW team and

		AOCS team independent from each other. Having a common SW CMP would imply that a team would be in charge of documenting activities and procedure done by a different team, and this is not considered realistic especially in a scenario in which AOCS and SW team belong to different organizations.

5.1.3 SW Requirements and Architecture Engineering Process (5.4.2)

Clause	**Description**	**Compliance**
5.4.2.1	Establishment and documentation of software requirements a. The supplier shall establish and document software requirements, including the software quality requirements, as part of the technical specification. EXPECTED OUTPUT: The following outputs are expected: a. Functional and performance specifications, including hardware characteristics, and environmental conditions under which the software item executes, including budgets requirements [TS, SRS; PDR]; b. Operational, reliability, safety, maintainability, portability, configuration, delivery, adaptation and installation requirements, design constraints [TS, SRS; PDR]; c. Software product quality requirements (see ECSS-Q-ST-80 clause 7.2) [TS, SRS; PDR]; d. Security specifications, including those related to factors which can compromise sensitive information [TS, SRS; PDR]; e. Human factors engineering (ergonomics including HMI usability) specifications, following the human factor engineering process specified in ECSS-E-ST-10-11 [TS, SRS; PDR]; f. Data definition and database requirements [TS, SRS; PDR]; g. g. Validation requirements [TS, SRS, ICD; PDR] h. Interfaces external to the software item [TS, ICD; PDR]; i. Reuse requirements (see ECSS-Q-ST-80 clause 6.2.7) [TS, SRS; PDR].	a. Functional and performance requirements can be supported by the model. The SW designer will have an executable version of the AOCS/GNC algorithms early in advance. It means that the Software Specification is itself executable, as to allow the core behaviour and performance of the final software product, to be uniquely captured, described and verified already at specification level. This provides help for outputs (a) and (f). Other outputs have to be generated classically.

5.4.2.3	Construction of a software logical model a. The supplier shall construct a logical model of the functional requirements of the software product. EXPECTED OUTPUT: Software logical model [TS, SRS; PDR]. b. The supplier shall use a method to support the construction of the logical model. EXPECTED OUTPUT: Software logical model method [TS, SRS; PDR]. c. The logical model shall include a behavioural view. EXPECTED OUTPUT: Behavioural view in software logical model [TS, SRS; PDR].	a. The model itself. b. & c. The description of the behaviour and dynamic aspects of the SW is performed via MATLAB/Simulink models. Functional requirements are captured by MATLAB/Simulink CAS (Control Algorithm Specification) implementation, using a FES simulator that allows functional verification the AOCS/GNC algorithms implemented. SIL tests are then set-up to validate the correct generation and functional behaviour of the SW

5.1.4 SW Architectural Design (5.4.3)

Clause	Description	Compliance
5.4.3.1	Transformation of software requirements into a software architecture a. The supplier shall transform the requirements for the software item into an architecture that: 1. describes its top–level structure; 2. identifies the software components, ensuring that all the requirements for the software item are allocated to its software components and later refined to facilitate detailed design; 3. covers as a minimum hierarchy, dependency, interfaces and operational usage for the software components; 4. documents the process, data and control aspects of the product; 5. describes the architecture static decomposition into software elements such as packages, classes or units; 6. describes the dynamic architecture, which involves the identification of active objects such as threads, tasks and processes; 7. describes the software behaviour.	a. The model itself. (1) The model structure provides the top-level structure. (2) The model provides the high level software components. These shall be identified one to one into the generated code. (3) The model covers the hierarchy, dependencies and interfaces. Operational usage might need textual description (4) Part of the ADD documentation can be generated automatically from the model. (5) The model subsystems shall be mapped to software elements as packages, unit, functions.

	EXPECTED OUTPUT: Software architectural design [DDF, SDD; PDR].	(6) The model itself defines the dynamic architecture. UML (or other type) diagrams can be generated automatically from the model. (7) The software behaviour is described by the model There are real time aspects, interface aspects to other parts of the SW that will need a textual description.

5.1.5 SW Design and Implementation Engineering Process (5.5.2)

Clause	Description	Compliance
5.5.2.1	Detailed design of each software component a. The supplier shall develop a detailed design for each component of the software and document it. EXPECTED OUTPUT: Software components design documents [DDF, SDD; CDR]. b. Each software component shall be refined into lower levels containing software units that can be coded, compiled, and tested. EXPECTED OUTPUT: Software components design documents [DDF, SDD; CDR]. c. It shall be ensured that all the software requirements are allocated from the software components to software units. EXPECTED OUTPUT: Software components design documents [DDF, SDD; CDR]. d. The supplier shall produce the detailed design model of the software components defined during the software architectural design, including their static, dynamic and behavioural aspects. EXPECTED OUTPUT: The following outputs are expected: Software static design model [DDF, SDD; CDR]; b. Software dynamic design model [DDF, SDD; CDR]; c. Software behavioural design model [DDF, SDD; CDR];	Note: The model-based approach is characterized by the seamless use of executable graphical models for specification, design and implementation, using commercial modelling and simulation tools such as MATLAB/Simulink. This means that SW documentation such as Design and Detailed Design is generated with the support of the modelling tool. a. The detailed design is part of the model generation. The model has to contain the details to allow the code generation. b. The refinement consists in expressing the component in the modelling language. The model has to contain the details to allow the code generation. c. To be ensured by completeness of the model and traceability to requirements inside the

		generated code. Modelling rules and guidelines shall be taken into account for architectural and interface design since this will affect the way the autocoding techniques generate the code that must be consistent with the original MATLAB/Simulink design (e.g. architecture mapping). It shall to insert into the Architectural and Detailed Design Documentation information generated automatically from the model.
5.5.2.4	Software detail design method a. The supplier shall use a design method (e.g. object oriented or functional method) to produce the detailed design including: 1. software units, their interfaces, and; 2. software units relationships. EXPECTED OUTPUT: Software design method [DDF, SDD; CDR]	a. The design method is "Model based Development". It is specified in modelling rules and guidelines. These rules and guidelines will affect the way the autocoding techniques generate the code that must be consistent with the original Simulink design (e.g. architecture mapping) and generate code able to be "qualified" with success.

5.1.6 Coding and Testing (5.5.3)

Clause	**Description**	**Compliance**
5.5.2.1	Development and documentation of the software units a. The supplier shall develop and document the following: 1. the coding of each software unit; 2. the build procedures to compile and link software units; EXPECTED OUTPUT: The following outputs are expected: a. Software component design documents and code (update) [DDF, SDD, source code; CDR];	a. The code is generated automatically. The code generator tool is properly configured to generate an optimised production code. The selected tool generates automatically a browsable HTML report with the generation of the code.

	b. Software configuration file - build procedures [DDF, SCF; CDR].	
5.5.3.2	Software unit testing a. The supplier shall develop and document the test procedures and data for testing each software unit. EXPECTED OUTPUT: The following outputs are expected: (1) Software component design document and code (update) [DDF, SDD, source code; CDR]; (2) Software unit test plan (update) [DJF, SUITP; CDR]. b. The supplier shall test each software unit ensuring that it satisfies its requirements and document the test results. EXPECTED OUTPUT: The following outputs are expected: (1) Software component design document and code (update) [DDF, SDD, source code; CDR]; (2) Software unit test reports [DJF, - ; CDR]. c. The unit test shall exercise: 1. code using boundaries at n-1, n, n+1 including looping instructions, while, for and tests that use comparisons; 2. all the messages and error cases defined in the design document; 3. the access of all global variables as specified in the design document; 4. out of range values for input data, including values that can cause erroneous results in mathematical functions; 5. the software at the limits of its requirements (stress testing).	a. & b SW tests are performed first at model level (limited scope) and then they have to be executed also at code level. The test specification for the software units generated by autocoding is built using the modelling tool (e.g. MATLAB/Simulink). Thus a simulator in the modelling tool should implement the test and provide the expected results. c. The MIL tests probably do not exercise all the branches, boundaries, message errors, out of range. Complementary unit tests with code shall be foreseen.

5.1.7 Integration (5.5.4)

Clause	Description	Compliance
5.5.4.1	Software integration test plan development a. The supplier shall complement the software integration test plan to define the integration of the software units and software components into the software item, providing the following data: 1. test design;	a. There are different levels of SW/SW integration testing: • Between components of autogenerated code where the integration test is achieved by the process

	2. test case specification; 3. test procedures; 4. test data. EXPECTED OUTPUT: Software integration test plan (update) [DJF, SUITP; CDR].	(The SW/SW integration between modules inside the model is 'by free' and can be tested into the FES (embedding the generated code into a S-function, for example). • Between AOCS FSW autogenerated code considered as one block and other SW modules (i.e. DHSW). In this case the requirement is fully applicable and adequate SW integration testing must be conducted on the SVF (Software Validation Facility) . Moreover the automatic generation of code from MATLAB/Simulink models guarantees a correct integration of the code correspondent to the internal subsystems of the models. The internal calls of the functions are generated in a coherent and consistent way with respect to the models.

5.1.8 SW Validation Process (5.6.2)

Clause	Description	Compliance
5.6.2.1	Establishment of a software validation process a. The validation process shall be established to validate the software product. EXPECTED OUTPUT: Software validation plan - validation process identification [DJF, SValP; PDR]. b. Validation tasks defined in clauses 5.6.3 and 5.6.4 including associated methods, techniques, and tools for performing the tasks, shall be selected and the regression test strategy specified. EXPECTED OUTPUT: Software validation plan - methods and tools [DJF, SValP; PDR].	a. a. The approach to validate the OBSW (AOCS/GNC) will be based on black-box testing of the automatically generated code, comparing the outputs versus the reference results obtained during the validation campaign of the Simulator Template (FES) using MIL and SIL.

	c. The validation effort and the degree of organizational independence of that effort shall be determined, coherent with ECSS-Q-ST-80 clause 6.3.5.19. EXPECTED OUTPUT: Software validation plan - effort and independence [DJF, SValP; PDR].	MIL and SIL outputs will be compared at numerical precision level as specified in this HB. The tests are performed embedding the production code inside FES Simulator as a Simulink S-Function. These tests are complementary to the unitary, integration and validation tests performed previously in Simulink (MIL testing). b. As in classical approach. c. As in classical approach.

5.1.9 SW Verification Process (5.8)

Clause	Description	Compliance
5.8.2.2	Selection of the organization responsible for conducting the verification a. If the project warrants an independent verification effort, a qualified organization shall be selected for conducting the verification. EXPECTED OUTPUT: Independent software verification plan - organization selection [DJF, - ; PDR]. b. This organization shall have the independence and authority needed to perform the verification activities. NOTE ECSS-Q-ST-80 clause 6.2.6.13 (independent software verification) and ECSS-M-ST-10 (project planning and implementation) contain further requirements relevant for this clause. AIM: A coherent and consistent approach to project organization within each project. EXPECTED OUTPUT: Independent software verification plan - level of independence [DJF, - ; PDR]. Includes consistent and verifiable requirements. EXPECTED OUTPUT: Requirements baseline verification report [DJF, SVR; SRR].	a. & b, as in classical approach. Depending on the scope of ISVV activities it would be needed to provide to the ISVV contractor the access to the modelling environment description and AOCS/GNC model defined.
5.8.3.3	Verification of the software architectural design a. The supplier shall verify the architecture of the software item and the interface design ensuring that:	a. A Verification and Validation Tool at model level provides the tracking of the requirements from a database

	1. architecture and interface are externally consistent with the requirements of the software item; 2. there is internal consistency between the software components; 3. the traceability between the requirements and the software components is complete; 4. the software components that are not traced to the software requirements are justified; 5. producing a detailed design is feasible; 6. operations and maintenance are feasible; 7. the design is correct with respect to the requirements and the interfaces, including safety, security and other critical requirements; 8. the design implements proper sequence of events, inputs, outputs, interfaces, logic flow, allocation of timing and sizing budgets, and error handling; 9. the hierarchical breakdown from high level components to terminal ones is provided; 10. the dynamic features (tasks definition and priorities, synchronization mechanisms, shared resources management) are provided and the real-time choices are justified; 11. the synchronisation between external interface and internal timing is achieved. EXPECTED OUTPUT: The following outputs are expected: a. Software architectural design to requirements traceability matrices [DJF, SVR or SDD; PDR]; b. Software architectural design and interface verification report [DJF, SVR; PDR].	(e.g. Word, Excel, DOORs, etc.) to Simulink level (via a friendly GUI). The code generator will maintain the requirements traceability down to code level. Traceability and design reports can be generated directly from the tools. The points (10) and (11) require a classical approach and cannot guarantee by the generation tool.
5.8.3.4	Verification of the software detailed design a. The supplier shall verify the software detailed design ensuring that: 1. detailed design is externally consistent with the architecture; 2. there is internal consistency between software components and software units; 3. the traceability between the architecture and the detailed design is complete; 4. the software units that are not traced to the components are justified; 5. testing is feasible, by assessing that: (a) commandability and observability features are identified and included in the detailed design in order to prepare the effective testing of the performance requirements;	a. A Verification and Validation Tool at model level provides the tracking of the requirements from a database (e.g. Word, Excel, DOORs, etc.) to Simulink level (via a friendly GUI) and further to the generated source code. The points (7), (11) and (12) require a classical approach and cannot guarantee by the generation tool

	(b) computational invariant properties and temporal properties are added within the design; (c) fault injection is possible. 6. operation and maintenance are feasible; 7. the design is correct with respect to requirements and interfaces, including safety, security, and other critical requirements; 8. the design implements proper sequence of events, inputs, outputs, interfaces, logic flow, allocation of timing and sizing budgets, and error handling; 9. the design model has been checked; 10. 1the hierarchical breakdown from high level components to terminal ones is provided; 11. 1the dynamic features (tasks definition and priorities, synchronization mechanisms, shared resources management) are provided and the real-time choices are justified; 12. 1the synchronisation between external interface and internal timing is achieved; EXPECTED OUTPUT: The following outputs are expected: a. Detailed design traceability matrices [DJF, SVR or SDD; CDR]; b. Detailed design verification report [DJF, SVR; CDR].	
5.8.3.5	Verification of code a. The supplier shall verify the software code ensuring that: 1. the code is externally consistent with the requirements and design of the software item; 2. there is internal consistency between software units; 3. the code is traceable to design and requirements, testable, correct, and in conformity to software requirements and coding standards; 4. the code that is not traced to the units is justified; 5. the code implements proper events sequences, consistent interfaces, correct data and control flow, completeness, appropriate allocation of timing and sizing budgets, and error handling; 6. the code implements safety, security, and other critical requirements correctly as shown by appropriate methods; 7. the effects of run–time errors are controlled; 8. there are no memory leaks; 9. numerical protection mechanisms are implemented. EXPECTED OUTPUT: The following outputs are expected:	a. Verification of code is performed by a commercial tool selected for coding verification activities. This tool allows performing static and dynamic analysis on code and generates a coverage report that analyses the paths, statements and decisions not covered in an efficient and direct way (GUI and interactive views) to drive the definition of additional tests for reaching 100% of coverage. The coding verification tool performs coding rules and standards verification and It can cover verification activities in PIL like timing, execution time, worst execution time analysis, schedulability, CPU budget, etc.

	a. Software code traceability matrices [DJF, SVR; CDR]; b. Software code verification report [DJF, SVR; CDR]. b. The supplier shall verify that the following code coverage is achieved	Traceability from SW requirements through SW units is ensured through the code generator tool. 141 Points (7), (8), (9) may require additional verification activities of tests performed in SVF (i.e. that behaviour in host matches behaviour on target). b. No changes wrt classical code c. The code coverage should be demonstrated for CAT-B. d. No changes wrt classical code e. No changes wrt classical code

A	B	C	D
100%	100%	AM	AM
100%	100%	AM	AM
100%	AM	AM	AM

Code coverage versus criticality category

	NOTE: "AM" means that the value is agreed with the customer and measured as per ECSS-Q-ST-80 clause 6.3.5.2. EXPECTED OUTPUT: Code coverage verification report [DJF, SVR; CDR, QR, AR]. NOTE This requirement is met by running unit, integration and validation tests, measuring the code coverage, and achieving the code coverage by additional (requirement based) tests, inspection or analysis. c. Code coverage shall be measured by analysis of the results of the execution of tests. EXPECTED OUTPUT: Code coverage verification report [DJF, SVR; CDR, QR, AR]. d. If it can be justified that the required percentage cannot be achieved by test execution, then analysis, inspection or review of design shall be applied to the non covered code. AIM: The goal of the complementary analysis is to assess that the non covered code behaviour is as expected. EXPECTED OUTPUT: Code coverage verification report [DJF, SVR; CDR, QR, AR]. e. In case the traceability between source code and object code cannot be verified (e.g. use of compiler optimization), the supplier shall perform additional	

	code coverage analysis on object code level as follows:	

A	B	C	D
100%	N/A	N/A	N/A

	NOTE: N/A means not applicable. EXPECTED OUTPUT: Code coverage verification report [DJF, SVR; CDR, QR, AR]. f. The supplier shall verify source code robustness (e.g. resource sharing, division by zero, pointers, run-time errors). AIM: use static analysis for the errors that are difficult to detect at run-time. EXPECTED OUTPUT: Robustness verification report [DJF, SVR; CDR]	
5.8.3.10	Verification of software documentation a. The supplier shall verify the software documentation ensuring that: 1. the documentation is adequate, complete, and consistent; 2. documentation preparation is timely; 3. configuration management of documents follows specified procedures. EXPECTED OUTPUT: Software documentation verification report [DJF, SVR; PDR, CDR, QR].	a. No changes wrt classical code It is possible to generate documentation from the model, so that the information can be inserted into the Architectural and Detailed Design Documentation following the DRD specified in ECSS standard. Since requirements, design and interfaces are defined graphically in the model, a configuration control tool for MATLAB/Simulink is used to ensure that changes are performed in a controlled way and that previous configurations can be recovered.
5.8.3.11	Schedulability analysis for real-time software a. As part of the verification of the software requirements and architectural design , the supplier shall use an analytical model (or use modelling and simulation if it can be demonstrated that no analytical model exists) to perform a schedulability analysis and prove that the design is feasible. NOTE The schedulability analysis proves that the real–time behaviour is predictable, i.e. that all the tasks	a. Preliminary analysis of SW real-time performances can be assessed also at early phases during requirements definition and SW design performing a standard profile analysis (e.g. Ravenscar profile analysis). b. Once the SW has been implemented and during the

	complete before their deadline in the worst case condition. EXPECTED OUTPUT: Schedulability analysis [DJF, SVR; PDR]. b. As part of the verification of the software detailed design , the supplier shall refine the schedulability analysis performed during the software architectural design on the basis of the software detailed design documentation. EXPECTED OUTPUT: Schedulability analysis (update) [DJF, SVR; CDR]. c. As part of the verification of the software coding and testing , the supplier shall update the schedulability analysis performed during the software detailed design with the actual information extracted from the code. EXPECTED OUTPUT: Schedulability analysis (update) [DJF, SVR; CDR].	validation phase, the real time and profiling characteristics (e.g. schedulability, memory budget, worst execution time, etc.) of the AOCS/GNC code generated are verified by PIL tests. A specific target shall be selected to run the generated code in the PIL configuration. c. Reports can be generated for the code to assess real time and profiling characteristics.
5.8.3.12	Technical budgets management a. As part of the verification of the software requirements and architectural design, the supplier shall estimate the technical budgets including memory size, CPU utilization and the way the deadline are met. EXPECTED OUTPUT: Technical budgets - memory and CPU estimation [DJF, SVR; PDR]. b. As part of the verification of the software detailed design, the supplier shall update the estimation of the technical budgets. EXPECTED OUTPUT: Technical budgets (update) - memory and CPU estimation [DJF, SVR; CDR]. c. As part of the verification of the coding, testing and validation, the technical budgets shall be updated with the measured values and shall be compared to the margins. EXPECTED OUTPUT: Technical budgets (update) - memory and CPU calculation [DJF, SVR; CDR, QR, AR].	a. The design of the models allows the definition of input, output, logic flow and events of the auto generated code. A first estimation of the software budget can be obtained performing the autocoding process already at preliminary SW design. b. Incremental consolidation of this estimation can be obtained during the SW advanced implementation phase when models are more mature exercising the autocoding process and analysing the correspondent SW budget again at that stage. No definitive technical budgets can be assessed from just modelling design level but it can be obtained from the analysis of real time execution of the SW interacting with avionics and on-board computer representative environment (e.g. PIL).
5.8.13	Behaviour modelling verification	a. b. & c. The behavioural view is the model itself.

	a. As support to the verification of the software requirements, the supplier shall verify the software behaviour using the behavioural view of the logical model produced in 5.4.2.3c. EXPECTED OUTPUT: Software behaviour verification [DJF, SVR; PDR]. b. As support to the verification of the software architectural design, the supplier shall verify the software behaviour using the behavioural view of the architecture produced in clause 5.4.3.4 EXPECTED OUTPUT: Software behaviour verification [DJF, SVR; PDR]. c. As support to the verification of the software detailed design, the supplier shall verify the software behaviour using the software behavioural design model produced in 5.5.2.3a. eo c., by means of the techniques defined in 5.5.2.6. EXPECTED OUTPUT: Software behaviour verification [DJF, SVR;CDR].	The behaviour of AOCS/GNC code generated shall be maintained with respect to the Simulink models. The same SW tests performed first at model level are then executed at code level through SIL (embedding the code generated automatically into the FES simulator as S-Function) to verify the correct generation of the code (same tests result for models and code). Reports can be generated to produce evidence of the tests results and code verification. Moreover MATLAB/Simulink allows identifying some behavioural errors in the models prior to execution such as interfaces or data type inconsistencies, missing parameters, etc.

5.1.10 Software maintenance process (5.10)

Based on what discussed in section 4.4 the requirements in section 5.10 of E40 must be considered also by the AOCS team that needs to define how to answer/tailor them.
The attention is drawn on requirement 5.10.2.2 Long term maintenance for flight software as this implies that the AOCS team needs to document how it intended to ensure that the autocoding environment is kept functional till spacecraft end of life.

5.2 Traceability to ECSS-Q-80C

5.2.1 SW Life Cycle Definition

From SW lifecycle point of view, SW development vs AOCS/GNC development refers to section 4 in this Handbook.

5.2.2 SW Dependability and Safety

Clause	Description	Compliance
6.2.3.2	a. The supplier shall define, justify and apply measures to assure the dependability and safety of critical software. NOTE: These measures can include: • use of software design or methods that have performed successfully in a similar application; • insertion of features for failure isolation and handling (ref. ECSS-Q-HB-80-03, software failure modes, effects and criticality analysis); • defensive programming techniques, such as input verification and consistency checks; • use of a “safe subset” of programming language; • use of formal design language for formal proof; • 100 % code branch coverage at unit testing level; • full inspection of source code; • witnessed or independent testing; • gathering and analysis of failure statistics; • removing deactivated code or showing through a combination of analysis and testing that the means by which such code can be inadvertently executed are prevented, isolated, or eliminated. EXPECTED OUTPUT: Software product assurance plan [PAF, SPAP; PDR, CDR].	a. No big changes wrt the classical approach. Also at modelling level there are some standards and guidelines suggested by advisory groups or industrial associations (e.g. MAAB, MISRA, NASA…) which are used as reference for safety-critical software development and are focused on qualification goals. However, these standard/guidelines are used by the space industry as a nice-to-have reference and not as a mandatory set of guidelines. On the other hand and based on the GMV experience, it is necessary to define (most of them based on the well-extended standard as MISRA, DO-178C, MAAB…) a minimum set of rules and/or restrictions at modelling and implementation level to generate high-quality code, reducing the code integration time and simplifying the monitoring and debugging of this code in SIL and PIL test environments.

5.2.3 SW Configuration Management

Clause	Description	Compliance
6.2.4.1	a. ECSS-M-ST-40 shall be applied for software configuration management, complemented by the following requirements.	a. Configuration control of SW and the corresponding models may be especially important to ensure the consistency at all times of models and the SW derived from them. The SW Configuration Management Plan shall consider the special characteristics of autocode.

5.2.4 Verification

Clause	Description	Compliance
6.2.6.1	a. Activities for the verification of the quality requirements shall be specified in the definition of the verification plan. NOTE: Verification includes various techniques such as review, inspection, testing, walk¬through, cross-reading, desk-checking, model simulation, and many types of analysis such as traceability analysis, formal proof or fault tree analysis. EXPECTED OUTPUT: Software verification plan [DJF, SVerP; PDR].	a. A set of rules and/or restrictions at modelling level has been defined in the context of the Autocoding WG activity to ensure later generation of high-quality code and allowing code verification and test activities in SIL configuration.
6.2.6.5	a. Software containing deactivated code shall be verified specifically to ensure that the deactivated code cannot be activated or that its accidental activation cannot harm the operation of the system. EXPECTED OUTPUT: Software verification report [DJF, SVR; CDR, QR, AR].	a. Some code generators (in order to simplify their architecture and generation mechanisms) may systematically generate elements that are not really used or related to the model. In this case the generated code includes unused or dead code. The code generation process and the automatic tools shall be controllable to avoid this problem. If a method for removing unused code is established, it shall be automated and be made part of the official tool chain.

5.2.5 Automatic Code Generation

Clause	Description	Compliance
6.2.8.1	a. For the selection of tools for automatic code generation, the supplier shall evaluate the following aspects: 1. evolution of the tools in relation to the tools that use the generated code as an input; NOTE: For example: compilers or code management systems.	a. This note contains criteria to help the selection. Since the expert know-how in the hand code is lost by using the automatic code generation tool, this expertise need to be implemented in the model and in

	2. customization of the tools to comply with project standards; 3. portability requirements for the generated code; 4. collection of the required design and code metrics; 5. verification of software components containing generated code; 6. configuration control of the tools including the parameters for customisation; 7. compliance with open standards.	the automatic code generation tool likely in configuration variables or as model rules and auto-code generation tool features. In this way the gap between autocode and hand-code is reduced. Code generators often assume access to general functions, as standard libraries or mathematical libraries, depending on the target computer this could be an issue. Either the libraries are certified as SW category 'B', either the code of the library is tested as the rest of the SW code.
6.2.8.2	a. The requirements on testing applicable to the automatically generated code shall ensure the achievement of the same objectives as those for manually generated code. EXPECTED OUTPUT: Validation and testing documentation [DJF, SValP; PDR], [DJF, SVS; CDR, QR, AR], [DJF, SUITP; PDR, CDR].	a. Currently no code generation tool is qualified to guarantee that code generated from MATLAB/Simulink behaves (at least from functional point of view) exactly the same of the model (as required by ECSS-Q 80 req. 6.2.8.3) thus avoiding the execution of unit and integration tests on the generated code. Testing at code level (unit/integration and Validation) shall be also repeated at code level even if code is auto generated This note proposes methods to achieve the goal: SW test is performed at model level through SIL (embedding the code generated automatically into the simulator). Coherence with tests performed in PIL shall be checked
6.2.8.3	a. The required level of verification and validation of the automatic generation tool shall be at least the same as the one required for the generated code, if the tool is used to skip verification or testing activities on the target code.	a. SW test is performed at model level through SIL (embedding the code generated automatically into the simulator), The same tests defined for the models in MIL are then repeated at code level via SIL.

6.2.8.5	a. Adherence to modelling standards shall be verified. EXPECTED OUTPUT: Software product assurance reports [PAF, -; -].	a. This note provides a set of rules and describe how can be verified. Modelling verification tools shall verify the compliance of the AOCS/GNC models design and implementation with rules and standards. For example, Simulink Verification and Validation toolbox from Mathworks allows running checks on models to verify the rules and standards that have been selected.

5.2.6 SW Requirements Analysis

Clause	Description	Compliance
6.3.2.4	a. In addition to the functional requirements, the technical specification shall include all non-functional requirements necessary to satisfy the requirements baseline, including, as a minimum, the following: 1. performance, 2. safety, 3. reliability, 4. robustness, 5. quality, 6. maintainability, 7. configuration management, 8. security, 9. privacy, 10. metrication, and 11. verification and validation. NOTE: Performance requirements include requirements on numerical accuracy. EXPECTED OUTPUT: Software requirements specification [TS, SRS; PDR].	a. Functional and (partially) performance requirements can be incorporated into the model. Other requirements have to be derived manually

5.2.7 SW Architectural Design

Clause	Description	Compliance
6.3.3.2	a. Mandatory and advisory design standards shall be defined and applied. b. Design standards also cover MMI aspects including as a minimum: 1. Context sensitive help function. 2. Common data format implementation syntax for help facility. 3. Undo-capability of all MMI-actions, whenever possible. EXPECTED OUTPUT: Design standards [PAF, -; SRR, PDR].	a. No changes wrt classical code. In addition, the AOCS model-based development shall follow several rules and guidelines for allowing the compatibility of the Simulink models with the auto-coding process and the quality and compliance of the produced code with the required standards. Therefore, it is necessary to define a set of rules and/or restrictions at modelling level to ensure later generation of high-quality code. Thus modelling standards define the design of the system, their use must be part of the design standards. b. MMI probably not applicable.

5.2.8 Coding

Clause	Description	Compliance
6.3.4.1	a. Coding standards (including consistent naming conventions and adequate commentary rules) shall be specified and observed. EXPECTED OUTPUT: Coding standards [PAF, -; PDR].	a. The coding standard to be applied to the generated code is imposed first on Simulink models via modelling rules. In fact there are modelling rules targeted to generate the correspondent code compliant to a standard (e.g. MISRA-C). The code is then generated automatically from the models by using a commercial tool that should produce code compliant to the standard selected. Another commercial tool is used for the verification of the standard on the generated code, including coding rules from many

		standards like MISRA-C:1998, MISRA-C:2004, MISRA C:2012, MISRA C++:2008, etc.

5.2.9 Testing and Validation

Clause	Description	Compliance
6.3.5.1	a. Testing shall be performed in accordance with a strategy for each testing level (i.e. unit, integration, validation against the technical specification, validation against the requirements baseline, acceptance), which includes: 1. the types of tests to be performed; NOTE: For example: functional, boundary, performance, and usability tests. 2. the tests to be performed in accordance with the plans and procedures; 3. the means and organizations to perform assurance function for testing and validation. EXPECTED OUTPUT: Software product assurance plan [PAF, SPAP; PDR, CDR].	a. The verification and validation proposed is based on a step-wise validation and verification approach MIL/SIL to achieve a final AOCS/GNC flight application ready to be embedded in the target HW. The AOCS/GNC code is executed in close-loop embedded in a Simulink subsystem as an S-function (SIL configuration).This S-function can be automatically integrated into the MIL simulator (replacing the Algorithm Simulink model) to verify that the generated code provides the same results as the original Simulink model. The new simulator containing the S-function is executed using the complete MIL simulator (including Real-World) and the results are compared with respect to the reference results obtained during the validation campaign of the MIL. Following this approach the unitary/integration tests and static and dynamic analysis of the production code is performed by embedding the generated code of the ASW functions in the FES simulator as S-functions (SIL). Thus unit tests and integration tests are performed, based on the unit and integration tests executed at MIL level

5.2.10 Maintenance

Clause	Description	Compliance
6.3.8.1	a. The organization responsible for maintenance shall be identified to allow a smooth transition into the operations and maintenance. NOTE: An organization, with representatives from both supplier and customer, can be set up to support the maintenance activities. Attention is drawn to the importance of the flexibility of this organization to cope with the unexpected occurrence of problems and the identification of facilities and resources to be used for the maintenance activities. EXPECTED OUTPUT: Maintenance plan [MF, -; QR, AR, ORR].	a. There is the need of two set of plans and procedures for maintenance of the SW products. One dedicated to the autocoded parts and one for the overall SW. The one for the overall SW shall explicitly address how it integrates new version/delivery of the autocoded part.

5.2.11 Numerical Accuracy

Clause	Description	Compliance
7.1.7	a. Numerical accuracy shall be estimated and verified. EXPECTED OUTPUT: Numerical accuracy analysis [DJF, SVR; PDR, CDR, QR].	a. Modelling rules shall be taken into account since CAS (Control Algorithm Specification) as they imply architectural constraints, development strategies, etc. Moreover in case of AOCS/GNC these rules must guarantee that the proper level of numerical accuracy is obtained in the generated code. In order to do it the requirements and guidelines are checked and implemented through the use of a checking rule tool (e.g. Simulink Verification and Validation toolbox), which permits to define coding style and rules and performs automatic checks of the Simulink models

		(e.g. through the Model Advisor). Numerical accuracy may be also affected by the environment (e.g. target processor) where the code is running. Additional analysis may be performed on the data type representation that is used in the target processor and it is not dependent from autocoding techniques.

5.3 Summary and Conclusions

The previous sections analysed the impact of AOC/GNC SW model-based development and automatic code generation on the ECSS standards to assess the compliance of the proposed SW development with the ECSS. From this analysis it can be concluded that the ECSS standards (ECSS-E40 and ECSS-Q80) are applicable also to SW development that involves autocoding techniques to generate the final ASW (category B of criticality). Some tailoring of the activities defined by the ECSS is needed and it is supported by the autocoding tools selected (e.g. generation of traceability matrices, SW documentation, reports, tests, etc.).
The ECSS compliance analysis with autocoding methodology can be resumed by the following main points:

- The model-based approach is characterized by the use of executable graphical models for specification, design and implementation, using commercial modelling and simulation tools such as MATLAB/Simulink. This means that SW documentation such as Requirements Specification, and Design is generated with the support of the modelling tool. Documents requested in the ECSS shall be generated from the models directly or at least with the support of the models design and implementation.
- The definition of system requirements is supported by the models to be autocoded (e.g. Simulink models). The models can be considered as detailed design of the components identified at high level architecture. Functional and interface requirements can be supported by the models while other requirements as for example performance, operational, maintenance, security, safety and verification required by ECSS-E40 cannot be supported. These types of requirements will be defined as textual and complementary to the models.
- Analysis of the generated products is performed considering also the requirements related to real time constraints, profiling data, memory aspects, schedulability, etc. This analysis will potentially drive to re-iteration with the MATLAB/Simulink design. Then the entire process of design, implementation, autocoding, verification and validation shall be executed again. This way the activities of SW development that includes modelling and autocoding can be iteratively and easily executed from early till late development phases. While for classical SW development the phases are

executed in a more "static" way, next phase starts when the previous one is quite consolidated.

- Traceability matrices at different levels need to be created from the model where requirements and design of the SW is implemented. For example traceability between requirements and design, between design and code and between code and unit tests shall be produced. Thus traceability capabilities of the modelling tool or a tool for generating this documentation would be needed.
- The software observability requirements must be taken into account when generating the code from the models. Mechanism for monitoring internal model variables into the final software must be considered at modelling level.
- The safety, security and other critical requirements must be included into the model design in order to be reflected into the generated code, otherwise additional manual code adaptation may be needed but not recommended in autocoding generation. For example protection by zero division, logical errors or integer overflow shall be included at model design level. Anyway if manual code adaptation is required (not recommended) a well-defined and controlled approach shall be considered like using scripts or a process described in detail to avoid committing human errors.
  The use of defensive code has some drawbacks like increase of code lines and memory use, introduction of dead code not fully testable. The selection of the method (defensive code or functionally where possible) for avoiding division by zero is not forced by the guideline but it is left to the user.
- The models used for autocoding may contain parts that are not to be coded into the final SW. The system specification shall indicate which elements of the model constitute the software in order to indicate to the code generator tool to not generate the corresponding code.
- The design of the models allows the definition of input, output, logic flow and events of the autogenerated code. But no code timing and size budget can be assessed at modelling design level. This budget can be only assessed if sufficient information is known about the generation process used by the autocoding tool.
- The use of autocoding techniques in the SW development implies to define and adopt modelling rules and guidelines that shall be taken into account for architectural and interface design since this will affect the way the autocoding techniques generate the code that must be consistent with the original design.
- The test specification for the software units generated by autocoding is built using the modelling tool (e.g. MATLAB/Simulink). Thus a simulator in the modelling tool should implement the test and provide the expected results.
- SW tests (unit, integration and validation) are performed at model level and they have to be performed also at code level as well through SIL (embedding the code generated automatically into the simulator). No qualified code generation tool (as required by ECSS) allows avoiding the execution of unit tests on the generated code.
- The tests that cover 100% of the SW at model level will be run as well for checking the code coverage through SIL (embedding the code generated automatically into the simulator). If 100% of coverage is not reached at code level, additional tests shall be performed till 100% of the code is covered (tested or justified).
- From the configuration management point of view, since requirements, design and interfaces are defined graphically in the model, we shall ensure that

MATLAB/Simulink manages the modifications in a controlled way and that previous configurations can be recovered.

- Since the expert know-how in developing the manual code is replaced by using the automatic code generation tool, this expertise need to be implemented in the model and in the automatic code generation tool likely in configuration settings or as model rules and auto-code generation tool features. In this way the gap between the implemented code generated automatically and the code implemented manually is reduced.
- Documentation is requested such as unit and integration test specifications and reports that are performed at MIL and SIL level. The generation of the requested documentation from the model shall be produced in compliance with ECSS standard in terms of format and content.
- The ASW DDR (Detailed Design Review) and TRR (Test Readiness Review) reviews are proposed to be official formal ECSS SW reviews. This will allow the proper review of the models design and validation that drive the validation of autogenerated SW.
- Some code generators (in order to simplify their architecture and generation mechanisms) may systematically generate elements that are not really used or related to the model. In this case the generated code includes unused or dead code. The code generation process and the automatic tools shall be controllable to avoid this problem.
- Code generators often assume access to general functions, as standard libraries or mathematical libraries, depending on the target computer this could be an issue. Either the libraries are certified as SW category ‘B’, either the code of the library is tested as the rest of the SW code.

6 CONCLUSIONS

In order to simplify the use of the code generation tools and to enhance the quality of the code produce addressing some of the problems of autocoding listed in the introduction (code readability is an easy example, but not the only one) this handbooks advocate not only to follow the process it describes, but also the use of guidelines that limits and constraint the use of the tools.
Guidelines have to be seen/understood in the same spirit in which in the SW development (see ECSS-Q-ST-80C 6.3.4.1) introduce coding standard. Guidelines are hence to be considered project specific. In order to simply the discussion and accepted of guidelines, avoid the project to define the guidelines each time, the next volumes of this Handbook provide a set of baseline guidelines, that each project in theory could tailor to its own specific needs, but in doing so should consider the consequence on the generated code.
The guidelines are categorized per applicability:

- Mandatory
- Strongly recommended
- Recommended

Mandatory	Strongly Recommended	Recommended
Definition		
• Guidelines that are absolutely essential. • Guidelines where 100% compliance shall be required. • Mostly intended to achieve correctness in the sense they aim at ensure that the code generation process works properly, free of error and the generated code corresponds to the model.	• Guidelines that are agreed upon to be a good practice, but use of legacy models preclude from being compliant at 100% • Models should conform to these guidelines to the greatest extent possible; however 100% compliance is not required • Mostly intended to achieve reliability in the sense that aim to ensure that the generated code is fully equivalent to the model to ensure validation is valid.	• Guidelines that are recommended to improve the appearance of the model diagram, but are not critical to running the model • Guidelines where conformance is preferred, but not required • Mostly intended to achieve readability, shareability, reusability and maintainability at model level without manual changes of the generated code.
Consequences – If the guideline is violated		
• Essential items are missing • The model might not work properly	• The quality and the appearance deteriorates • There may be an adverse effect on maintainability, portability, and reusability	• The appearance will not conform with other projects
Waiver Policy – If the guideline is intentionally ignored		
• The reasons must be justified and documented	• The reasons must be documented	

The applicability describes the objective of the guideline, its importance and determines the consequences of violations.
The guideline are clearly tool chain dependent, at the current time the only tool chain that has been analysed is the one of Mathworks as is among the most used in the space domain.