09/05/2017 @ESTEC

AUTOCOGEQ Final Presentation

INNOVATING SOLUTIONS

Index

AUTOCOGEQ Project Overview & Main Objectives

AUTOCOGEQ Project Data

- 

→ September 2015 - May 2017 (~20 months)

- 

→ Manrico Fedi Casas

- Section → TEC-QQS (contribution from TEC-SAG, TEC-ECN and TEC-SWE)

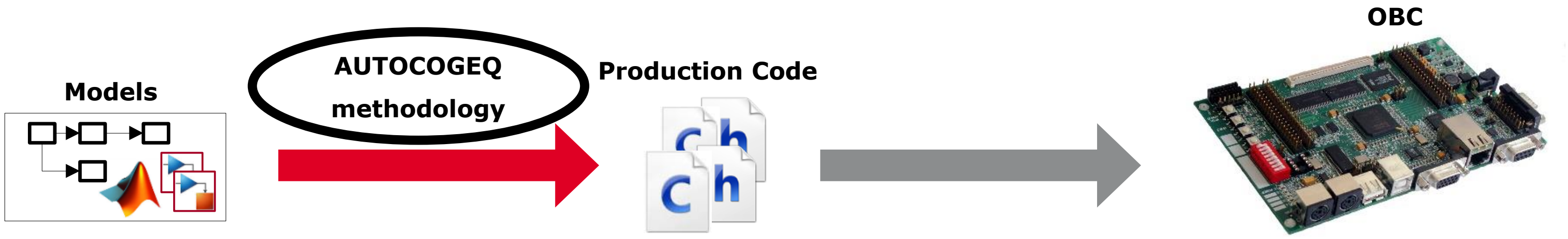
- Company → gmv INNOVATING SOLUTIONS

Introduction

- Consolidated approach in prototyping space **AOCS/GNC SW** is to follow a **model-based** component **approach**:
  - → Develop a simulator in the **Matlab/Simulink** environment for GNC algorithms
  - → The **models** are then **autocoded** (by means of automatic techniques)
  - → Production **code** generated is **embedded** in **OBSW/OBC** and validated in real time test benches.
- Complete process of the **SW development** and **verification** shall be clearly **defined** and **analyzed**
- **Quality** of the **code** produced from autocoding of Matlab/Simulink models shall be **analyzed** according to the final use

AUTOCOGEQ Main Objectives

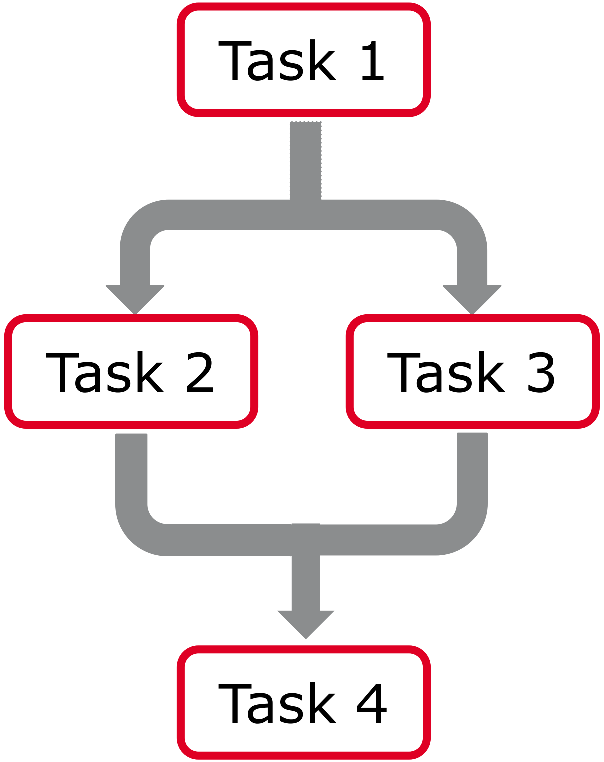
- The AUTOCOGEQ activity has the following main objectives:
  - Define a **methodology** that allows automatic code generation from Matlab/Simulink models for direct integration in on-board critical flight SW
    → ECSS critical software category B



  - Assess the **impact** of model-based design and autocoding **in ECSS**
  - Select a **set of tools** to support the SW development with autocoding
  - Develop a Matlab tool (**Wizard**) to help the developer applying the autocoding methodology defined

AUTOCOGEQ Activities Overview

- 4 main tasks have been defined:
  - **Task 1**
    - Define autocoding methodology (SW development and V&V)
    - Perform tools evaluation
    - Define modelling rules definition
    - Assess autocoding impact to ECSS
  - **Task 2**
    - Selection and update of GNC simulator use case
  - **Task 3**
    - Define and implement an autocoding tool (Wizard)
  - **Task 4**
    - Demonstrate the complete end to end autocoding methodology
    - GNC simulator used with support of Wizard



Autocoding Methodology

SW Development & Verification Overview (1/3)

- Impact of the model-based design and use of autocoding techniques in flight SW lifecycle

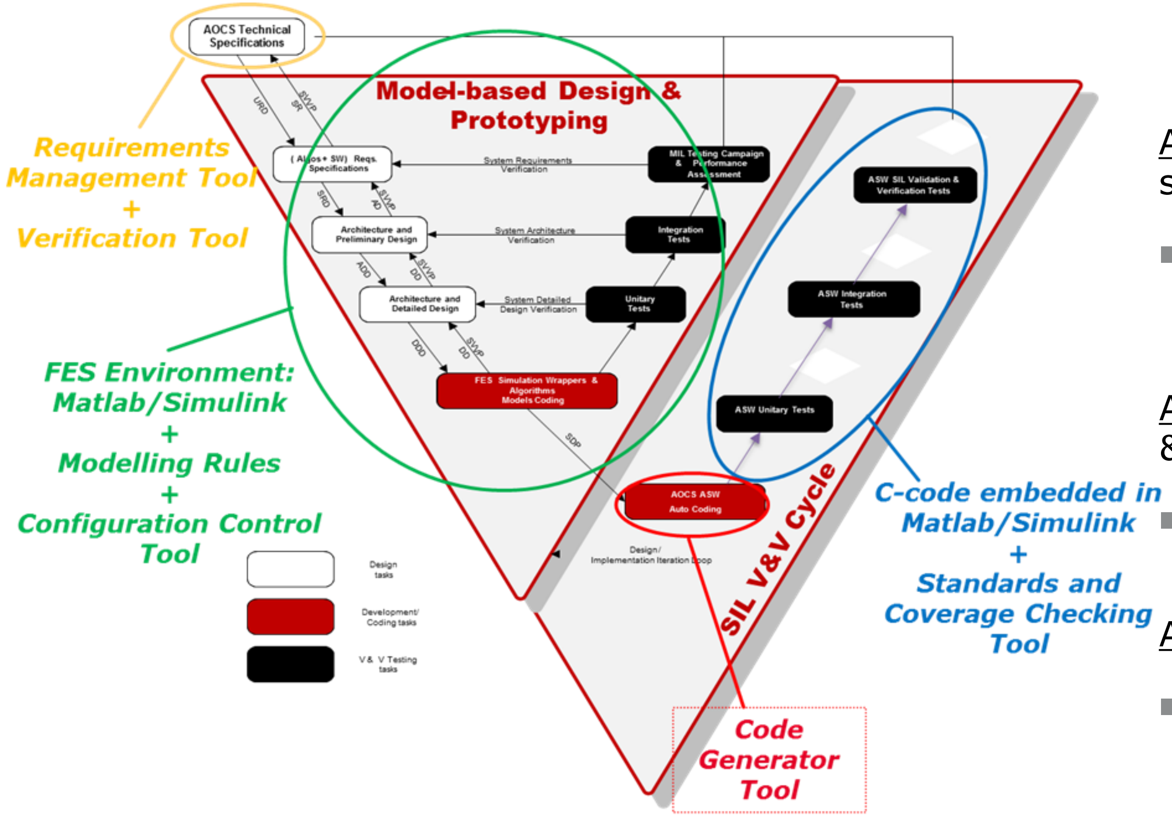
- Analyzed SW lifecycle phases according to the ECSS-E40 standard such as:
  - SW Specification
  - SW Design
  - SW Implementation
  - SW Verification and Validation (V&V)

- AUTOCOGEQ activity focuses on the AOCS/GNC SW development → model-based design in Matlab/Simulink (Functional Engineering Simulator - FES)

- AOCS/GNC SW development strategy is part of an integrated, coherent and incremental Design, Development, Verification and Validation (DDVV) approach based on the chain:

FES → Autocoding → SIL → PIL

SW Development & Verification Overview (2/3)



- **SW Specification**

  Software System Specification and AOCS/GNC Control Algorithm Specification

<u>AUTOCODING</u> → Models supports the requirements specification

- **SW Design (Preliminary & Detailed Design)**

  SW algorithms, architecture and interfaces definition

<u>AUTOCODING</u> → Model-based design - Modelling Rules & Standards

- **SW Implementation**

  FES implementation in Matlab/Simulink

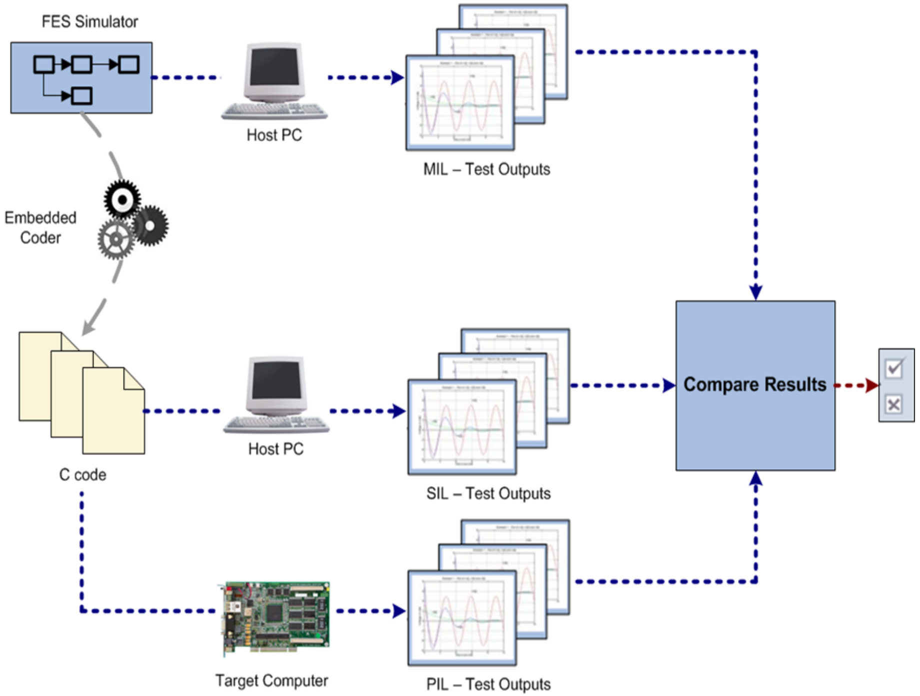
<u>AUTOCODING</u> → Autocoding methodology & Tools

- **SW V&V**

  SW Verification & Validation (unit, integration and validation tests)

<u>AUTOCODING</u> → SIL, Requirements, Modelling and Coding standards verification

SW Development & Verification Overview (3/3)



- **SIL Verification**

  Generated code embedded into an S-Function block in Simulink.

  → Verify correct portability of models algorithms to code

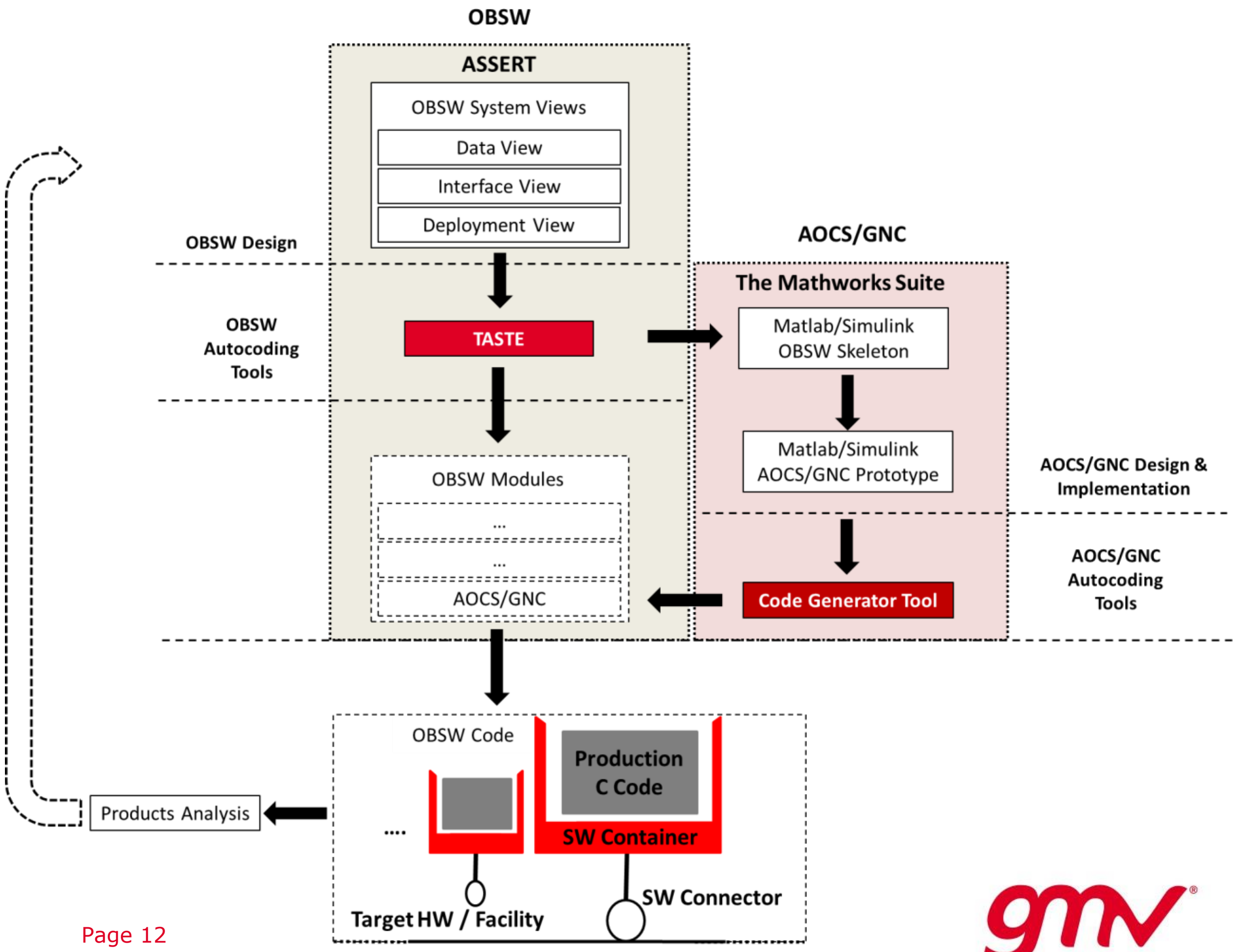
- **PIL Verification**

  Generated code embedded into a flight representative OBC.

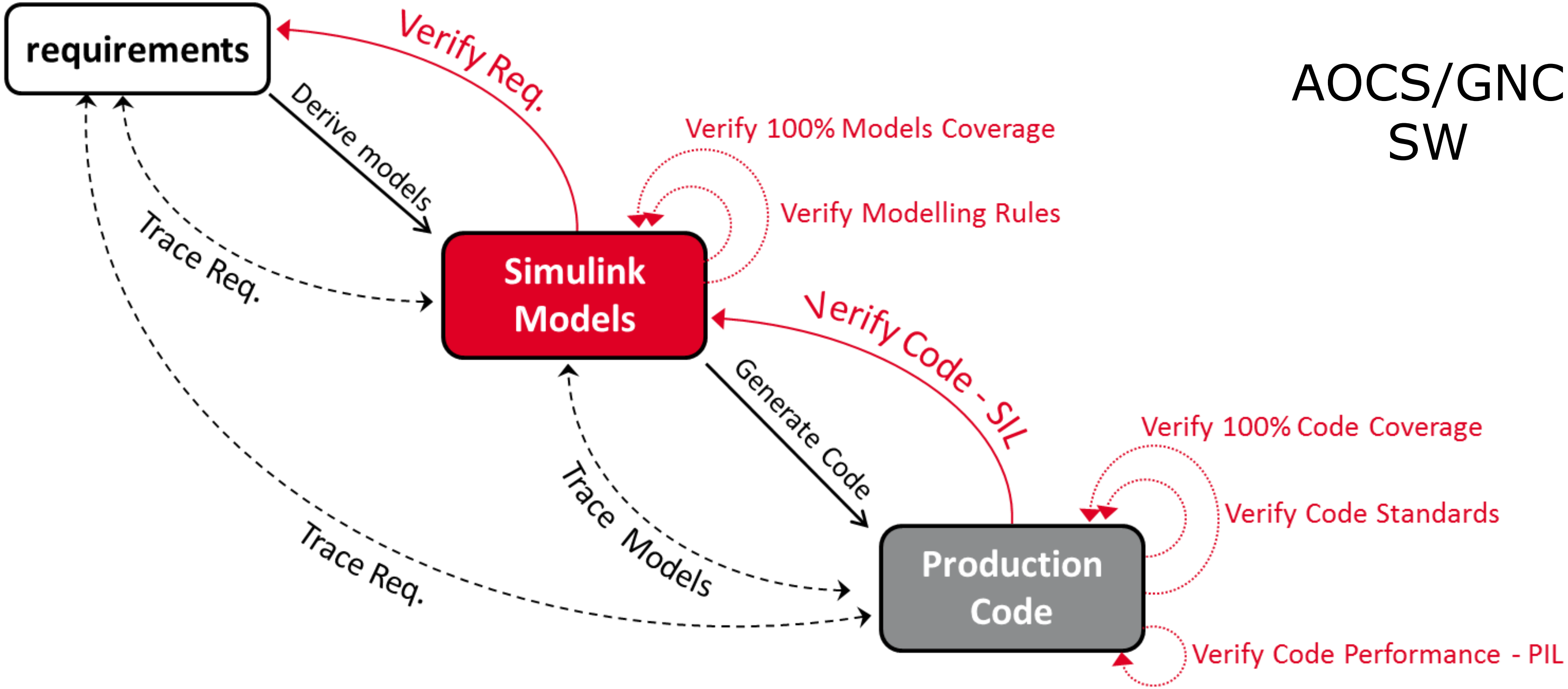
  → Verify code performances (e.g. schedulability, memory budget, worst execution time, etc.)

OBSW Autocoding Generation Approach

- OBSW generation based on ASSERT approach
- OBSW is designed according to system views (i.e. data view, interface view and deployment view)
- TASTE toolsuite is proposed to design the complete OBSW
- Autocoding approach adopts 2 parallel branches:
  - The **Mathworks branch** for AOCS/GNC generation
  - **ASSERT/TASTE branch** for other OBSW modules generation

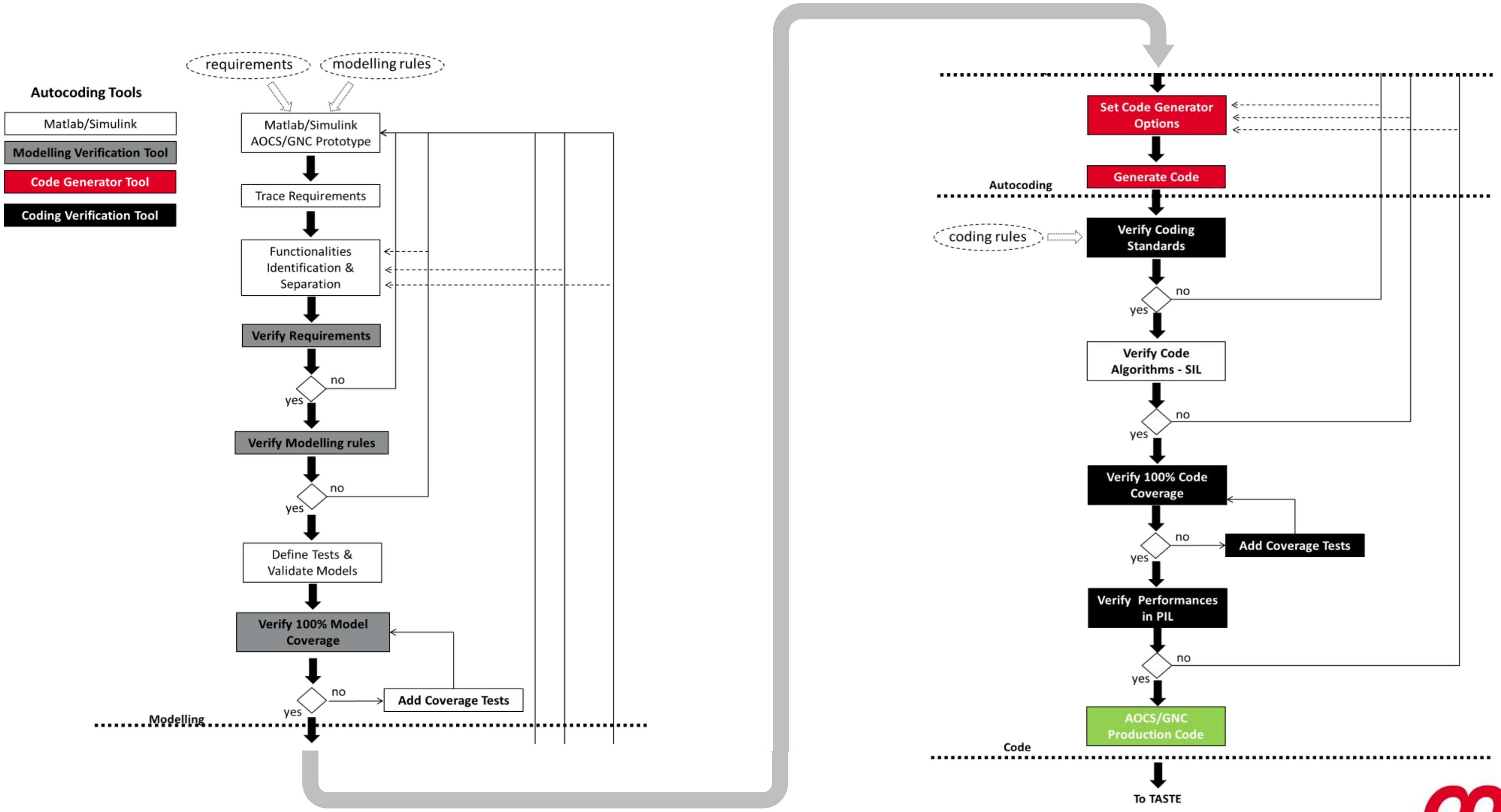


AOCS/GNC Autocoding Activities Defined by Methodology



AOCS/GNC
SW

AOCS/GNC Autocoding Workflow



Modelling Rules & Guidelines

- AOCS model-based development shall follow several **rules** and **guidelines** for allowing the compatibility of the Simulink models with the auto-coding process.

- The guidelines for the AOCS/GNC modelling in Matlab/Simulink may be grouped in two categories:

  – **Modelling Architectural and Design Rules**
    - Rules/guidelines that need to be followed at architectural and design level of the AOCS/GNC subsystem
    - RATIONALE: to port efficiently code to SVF and PIL verification and guarantees architectural mapping

  – **Modelling Implementation Rules (coding and style)**
    - Rules/guidelines that need to be followed by the Simulink implementation of the AOCS/GNC models
    - RATIONALE: to prevent errors, language-specific pitfalls, non-optimised statements, forbidden constructs, complexity restrictions and readability in the code generated

Autocoding Tools Evaluation (1/2)

- 3 groups of tools have been evaluated to support autocoding activities:
  - **Code Generation Tools**
    Tools used for the generation of production code from Matlab/Simulink models
  - **Modelling Verification Tools**
    Tools used for the modelling standards compliance verification (e.g. requirements, modelling rules and coverage) on the Matlab/Simulink models
  - **Coding Verification Tools**
    Tools used for the verification of the production code generated (e.g. metrics, standards, coverage, etc.)

Code Generation Tools	Modelling Verification Tools	Coding Verification Tools
Embedded Coder	Simulink Verification and Validation	BullseyeCoverage
Target Link	MES Model Examiner	LDRA toolsuite
QGen	BTC Embedded Specifier, Validator and Tester	Vector Cast
TASTE	MES M-XRAY	Polyspace
	QGen (Static Model Verifier)	Rapita Verification Suite (RVS Toolbox)

Autocoding Tools Evaluation (2/2)

- Autocoding tools evaluated according to the following criteria:
  - **Generic criteria**
    - Interfacing with Matlab/Simulink Environment
    - Installation Procedure
    - Learning Curve
    - Market Price
    - Documentation
    - Support
  - **Tool Specific criteria**
    - Performance
      - Code generation Tools: Code readability, Requirements traceability, Code architectural mapping level, Code optimization (modules and lines), Generator configurability level (for metrics and statements)
      - Modelling Verification Tools: Requirements verification, Modelling rules verification, Model Coverage, Verification tool configurability level, Reporting verification
      - Coding Verification Tools: Requirements verification, Coding rules verification, Metrics verification, Static analysis, Coverage features, Reporting verification

Autocoding Tools Selection

- The tools selected and then purchased for AUTOCOGEQ activities are:
  - Code generation Tool → **Embedded Coder**
  - Modelling Verification Tools → **Simulink Verification & Validation Toolbox**
  - Coding Verification Tools → **LDRA**

ECSS Compliance Analysis (1/2)

- ECSS-E40 standard has been reviewed and requirements that are relevant to autocoding methodology has been analyzed
- ECSS-Q80 has been analysed and impact of model-based design and autocoding has been assessed
- Main conclusions from analysis:
  - The definition of system **requirements** is **supported by the models** that can be considered as detailed design of the components identified at high level architecture
  - **SW documentation** such as Requirements Specification, and Design is **generated** with the support of the **modelling tool**.
  - SW development that includes modelling and autocoding can be iteratively and easily executed from early till late development phases (**dynamic development**)
  - **Traceability matrices** created **from the model** where requirements and design of the SW is implemented
  - The software **observability, safety**, **security** and other **critical requirements** must be included into the model design in order to be reflected into the generated code (e.g. protection for division by zero, logical errors
  - Some ECSS verification activities **cannot fully covered** by methodology (e.g. testability, atomicity, correctness, etc.)

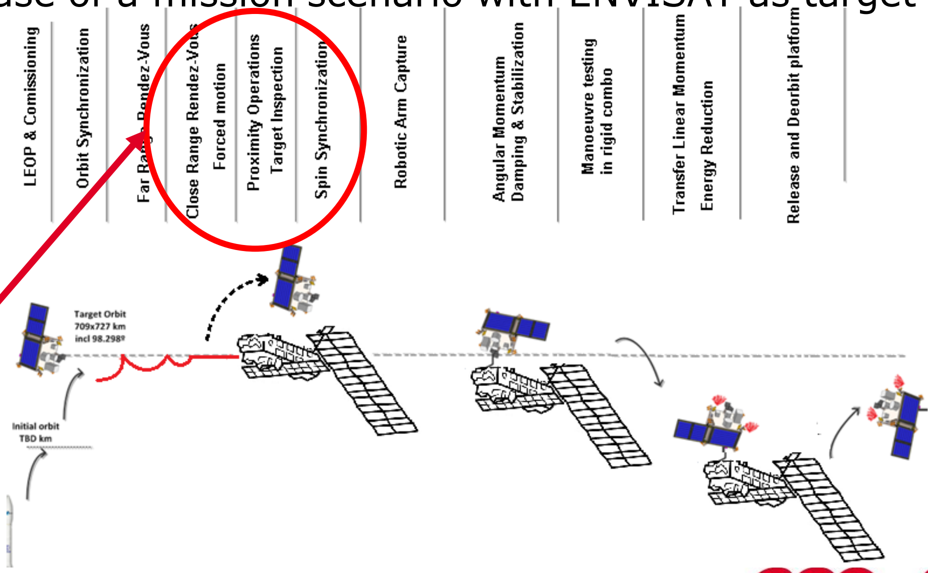
ECSS Compliance Analysis (2/2)

- The **models** used for autocoding **may contain parts** that are **not to be coded** into the final SW
- No **code timing** and **size budget** can be assessed at modelling design level
- The use of autocoding techniques in the SW development implies to define and adopt **modelling rules** and **guidelines**
- **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**
- Certain **aspects** of **Unit Tests** are **not covered** by methodology (e.g. robustness, boundary, etc.)
- Documentation is requested such as unit and integration **test specifications** and **reports** that are performed at **MIL** and **SIL** level
- The ASW **DDR** (Detailed Design Review) and **TRR** (Test Readiness Review) reviews are **proposed** to be official **formal ECSS SW reviews**
- Some **code generators** (in order to simplify their architecture and generation mechanisms) may systematically generate **additional elements**
- Code generators often assume access to **external libraries** that must also be **qualified** as SW category B

GNC Demonstrator Overview

GNC Demonstrator Overview (1/3)

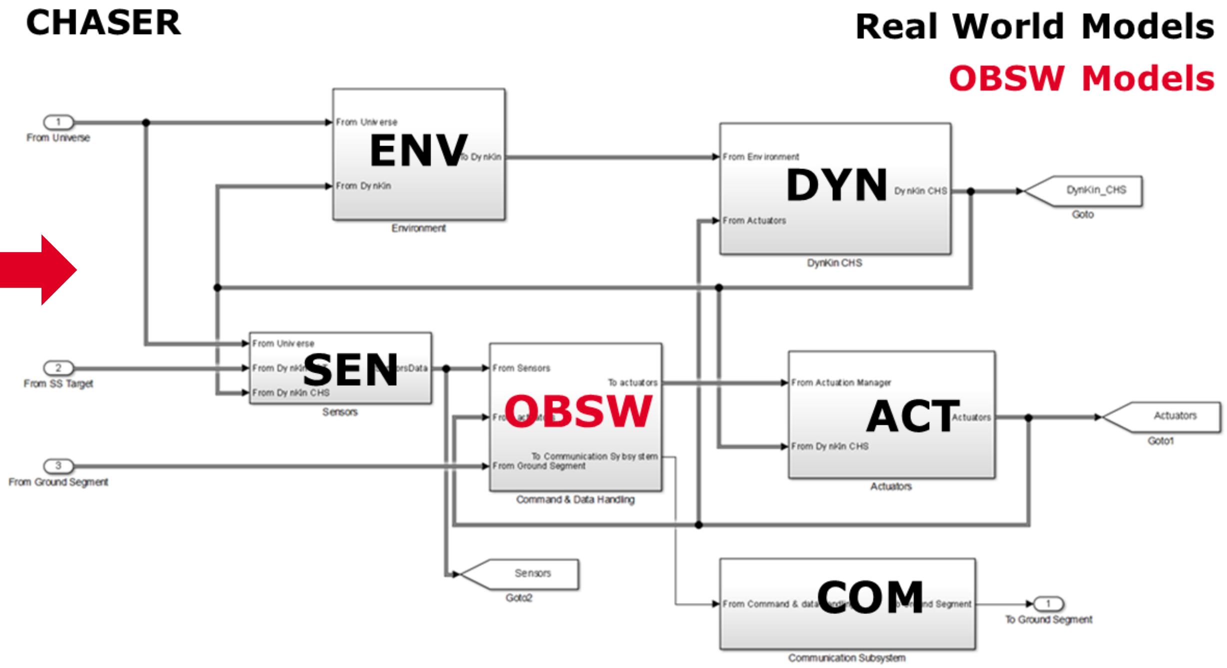
- A Simulink simulator is selected to be used as use case for autocoding methodology demonstration
- GNC demonstrator implements a real and complex GNC scenario
- No high performance simulator is required in the scope of AUTOCOGEQ → focus on methodology and processes
- GNC demonstrator covers the last synchronization phase of a mission scenario with ENVISAT as target (ADR):
  - Large ESA-owned dead satellite: Envisat
  - Satellite is tumbling
  - ~8 tones of mass
  - Polar sun-synchronous orbit (altitude ~772km)



Simulated part in GNC demonstrator

GNC Demonstrator Overview (2/3)

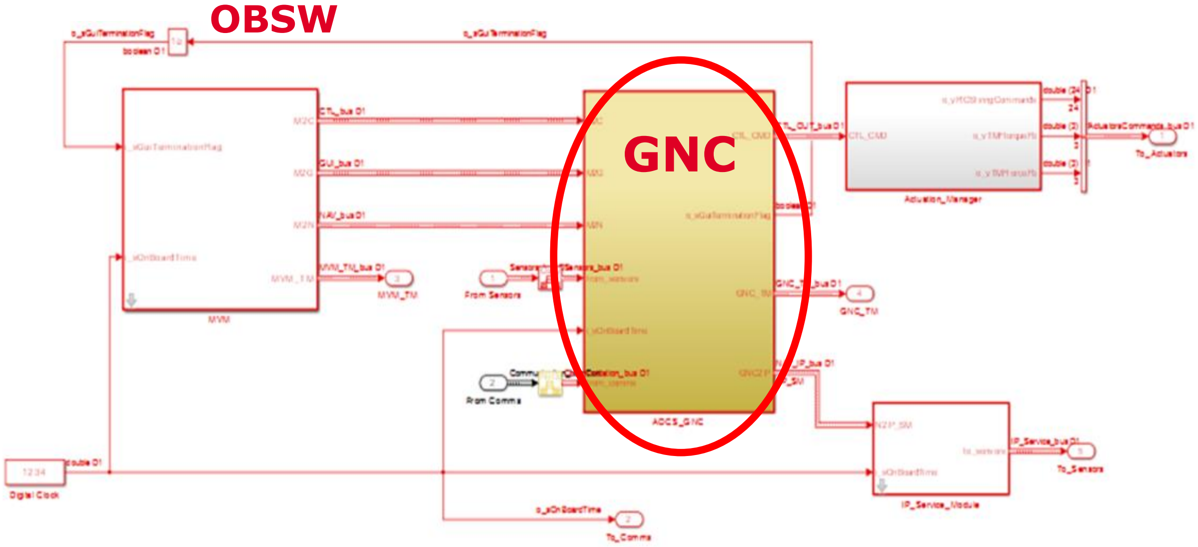
- A Matlab/Simulink simulator re-used from NGT-ATB activity (ADR simulator):
  - Universe
  - Ground Segment
  - Space Segment Target
    - ENV
    - DYN
  - Space Segment Chaser
    - ENV
    - DYN
    - SEN
    - ACT
    - COM
    - OBSW



GNC Demonstrator Overview (3/3)

- Only AOCS/GNC subsystem has been analysed in AUTOCOGEQ → part to be autocoded

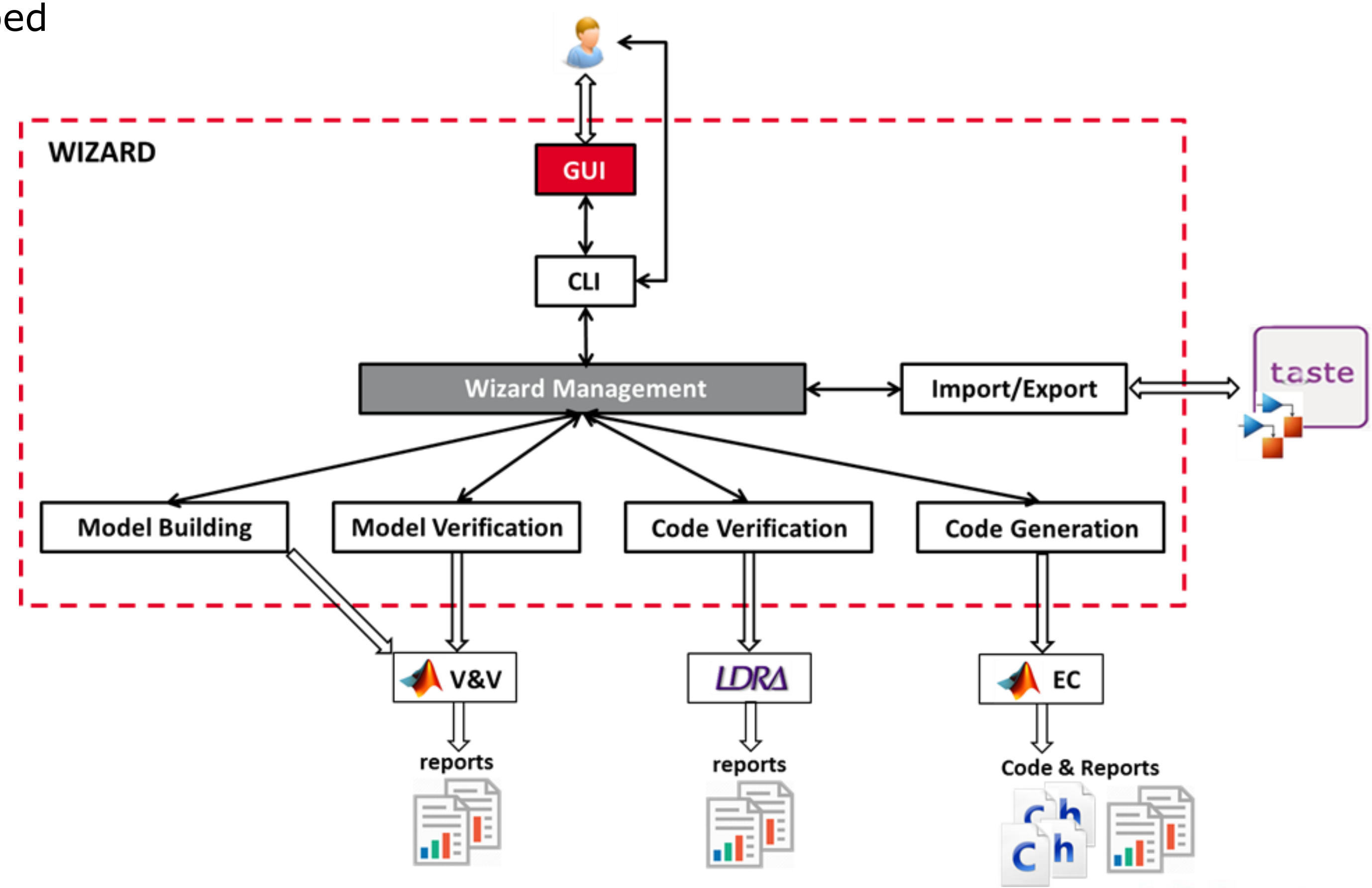
- GNC models have been reviewed and updated to be compliant with the autocoding methodology:
  - Track requirements to models
  - Compliance with modelling rules
  - Identification and separation of functionalities



AUTOCoding Wizard

AUTOCoding Wizard Overview

- A tool (**Wizard**) has been developed to support the **autocoding activities** defined by the methodology:
  - Support to models building/updating
  - Support to models verification activities
  - Support to code generation
  - Support to code verification activities
- The Wizard is **implemented** in **Matlab** and integrates the tools selected
- Wizard can support all SW development phases



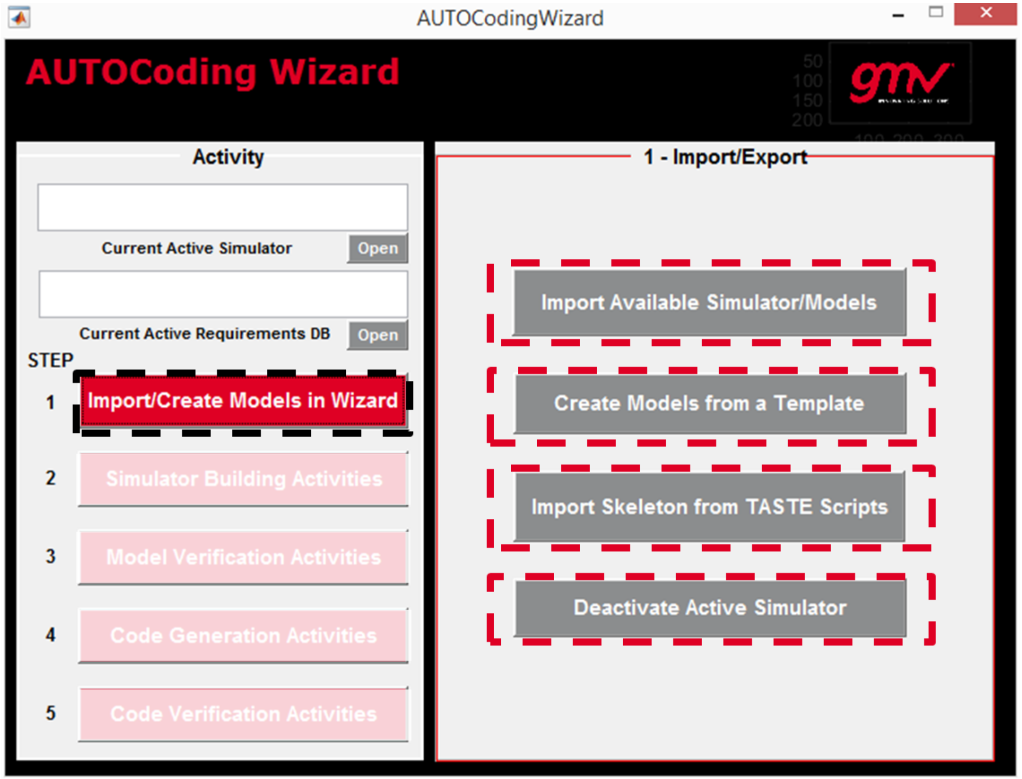
Wizard Support for Import/Update Models

- **Import Available Simulator/Model**
  Import an existing Simulink model into the Wizard

- **Create Model from Template**
  Create a new model based on Simulink templates provided with the Wizard

- **Import Skeleton from TASTE Scripts**
  Create and import a Simulink skeleton from TASTE generated scripts

- **Deactivate Active Simulator**
  Remove the active Simulink model from the Wizard



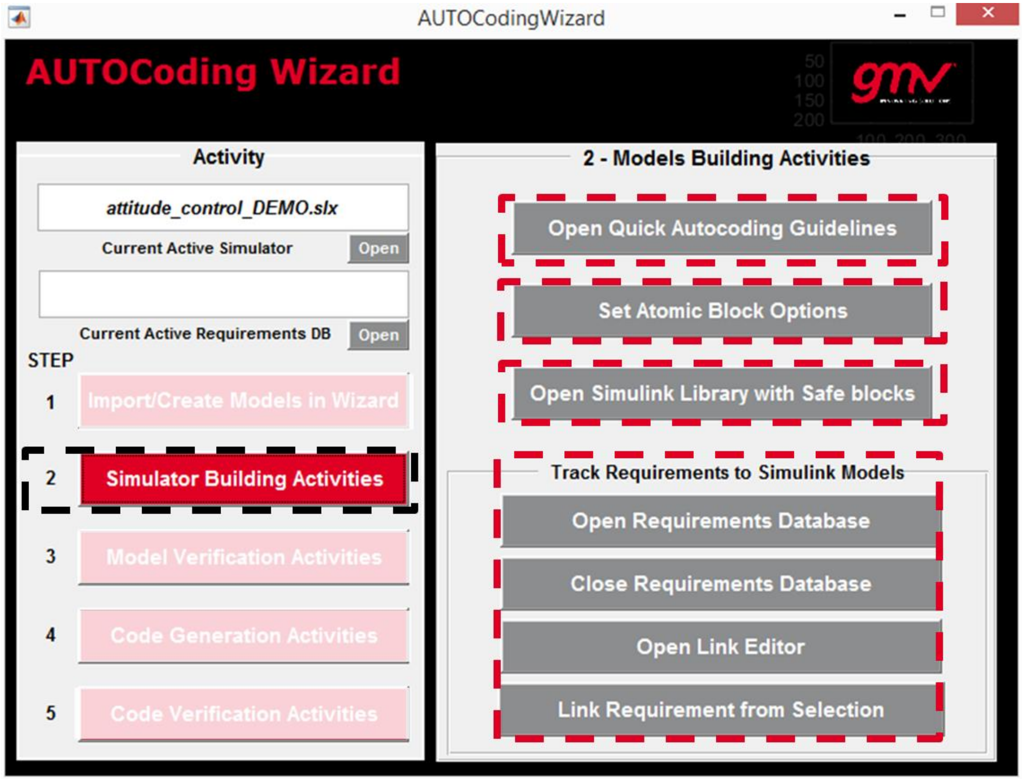
Wizard Support for Simulator Building

- **Open Quick Autocoding Guidelines**
  Open a quick HTML guideline reporting the most important modelling rules

- **Set Atomic Block Options**
  Set automatically a subsystem as atomic with the specific settings defined by the methodology

- **Open Simulink Library with Safe blocks**
  Open the Simulink library where only safe blocks (block totally compatible with autocoding and with AOCS/GNC models prototyping) are available

- **Track Requirements to Simulink Models**
  Track the requirements from a database (e.g. Word, Excel, DOORs, etc.) to the Simulink block



Wizard Support for Model Verification

- **Models Requirements Verification**

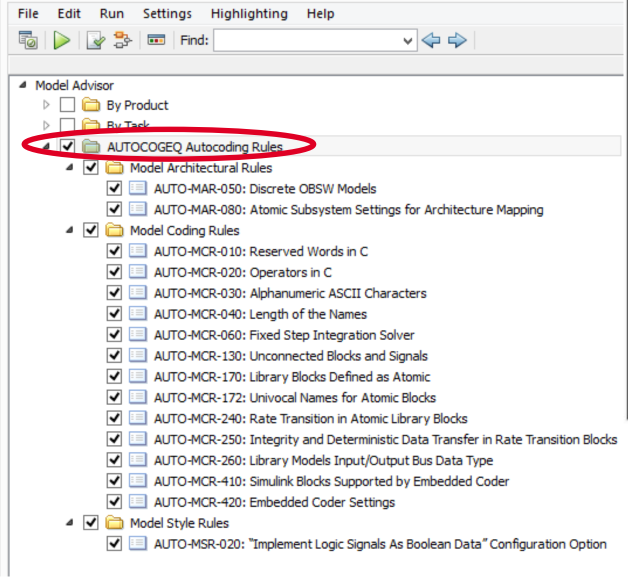
  Verify the links between the requirements and the Simulink models and produce traceability information

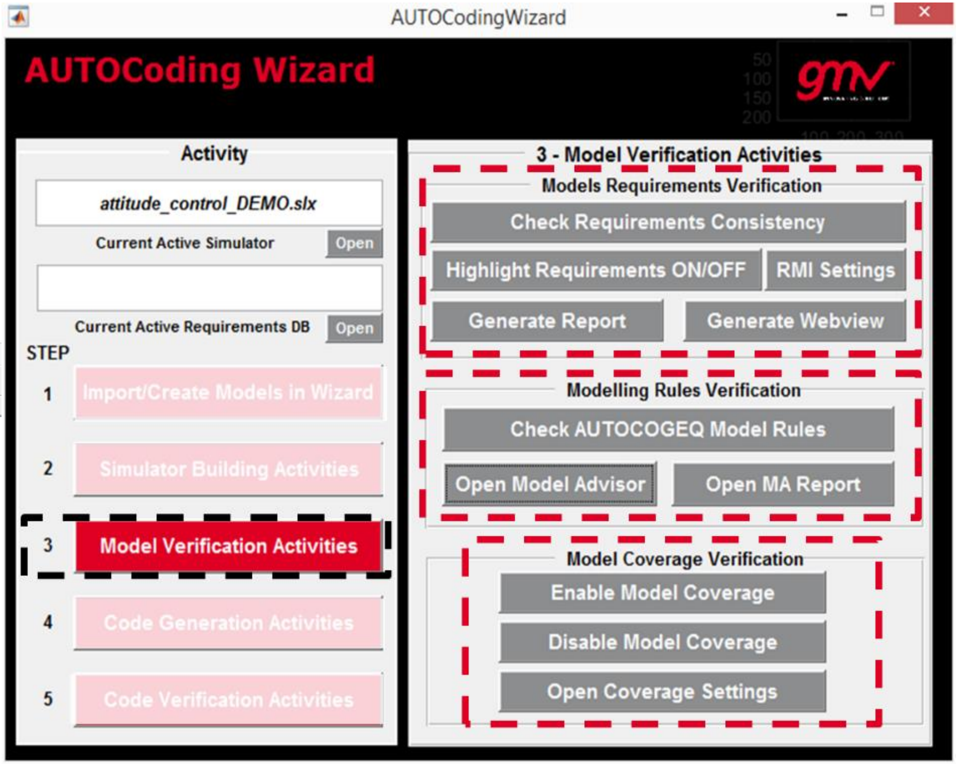
- **Modelling Rules Verification**

  Verify the compliance of the Simulink models with the modelling rules defined for AUTOCOGEQ and produce detailed reports.

  – Checks have been integrated into Model Advisor

  – New user defined checks can be integrated into Wizard and available into Model Advisor





- **Model Coverage Verification**

  Enable/Disable and configure the model coverage to be executed during the tests

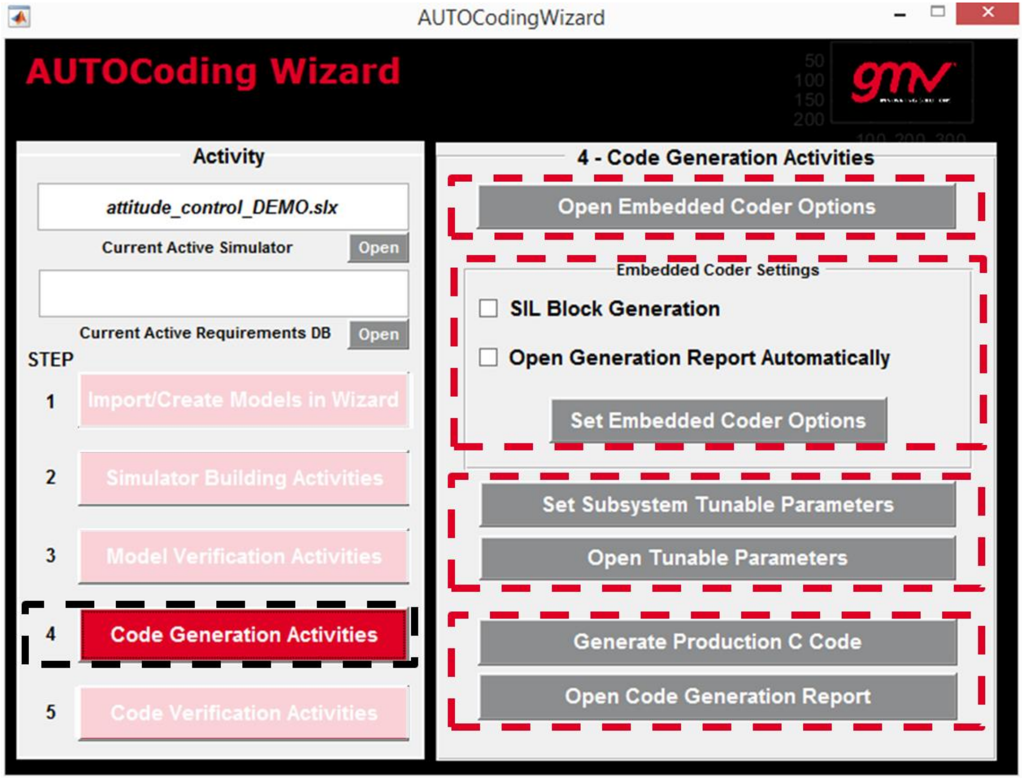
Wizard Support for Code Generation

- **Open Embedded Coder Options**
  Open the Embedded Coder GUI to check the options set to generate code

- **Set Embedded Coder Options**
  Set the Embedded Coder with the options defined by the autocoding methodology

- **Set Model Tuneable Parameters**
  Set the tuneable parameters for the model to generate code

- **Generate Production Code**
  Generate C code from the selected subsystem



Wizard Support for Code Verification

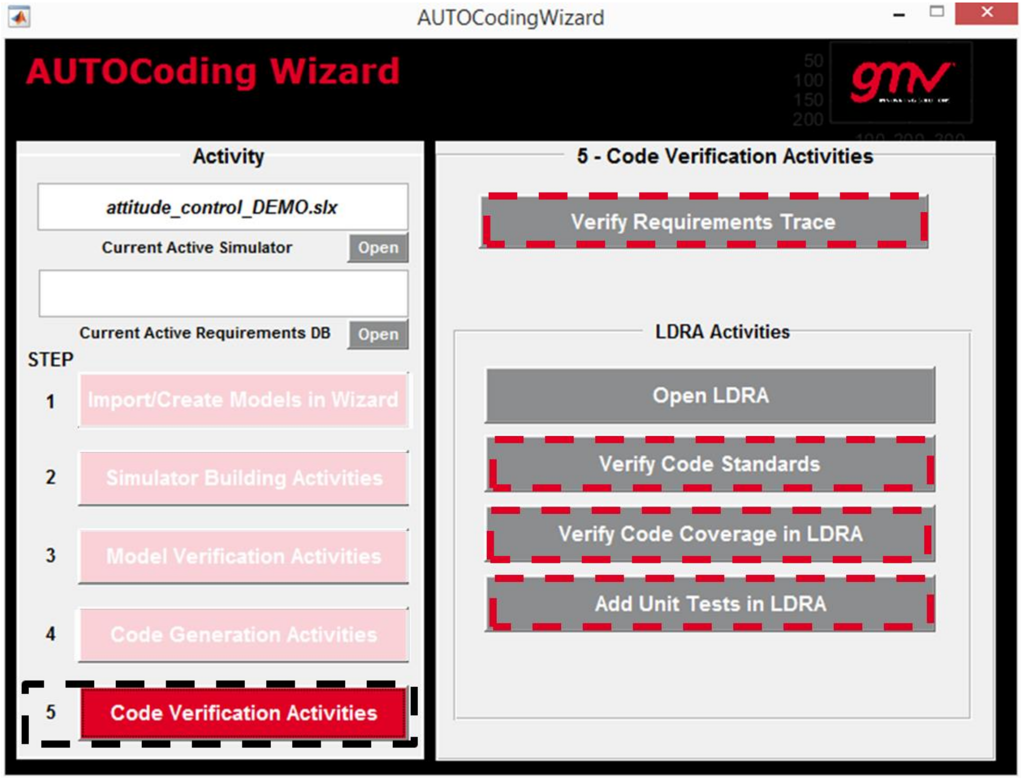
- **Verify Code Requirements Trace**
  Verify the links between the requirements and the code generated

- **Open LDRA**
  Locate the LDRA installation and open the LDRA tool

- **Verify Code Standards**
  Verify the compliance of the generated code with selected standard via LDRA and produce detailed reports

- **Verify Code Coverage**
  Instrument the generated code to produce coverage data from a test to be analyzed in the LDRA environment

- **Add Unit Tests in LDRA**
  Open the LDRA to manage the creation of additional unit tests to increase code coverage



End 2 End Methodology Demonstration

E2E Methodology Demonstration

- Demonstration of the complete End to End autocoding chain
- Wizard has been used to support the demonstration
- GNC demonstrator has been used as use case → Code generation for GNC subsystem
- Following verifications have been performed:
  - Requirements verification
  - Modelling Verification
  - Code Generation and Analysis
  - Code Verification
- Wizard provided HTML reports for all verifications

Requirements verification

- Requirements verification is supported by the Wizard

- The verification of the requirements trace is performed in two phases:

1. **Verify the consistency of the requirements**

   Check if the requirements links associated to the AOCS/GNC models are consistent (i.e. requirements document exists, correct links location inside the document, existing requirement ID, etc.)

2. **Verify the requirements trace**

   Generate the requirements traceability report to check if all the requirements have been linked to the GNC models.

- HTML reports are generated for both verifications by the Wizard

Modelling Verification

- Two verifications has been performed at model level by the Wizard:

1. **Modelling Rules Verification**

   The Wizard implements checks to automatically verify the rules defined the methodology

2. **100% Model Coverage**

   The wizard is used to set the coverage and reference tests are run to assess the percentage

   A strategy has been defined for additional tests to reach 100%:

   - Assess coverage of library models by specific unit tests
   - Add tests to execute models not covered and produce cumulative coverage data
   - Justify not-covered subsystem

- HTML reports are generated for the verifications

Code Generation and Analysis

- Production **C code** has been **generated** via Wizard

- Analysis of HTML **code generation report** has been performed to assess:
  – list of C files generated by the tool
  – mapping of the model subsystems to the generated code
  – C code interfaces (i.e. entry points and variables)
  – metrics of the generated code (i.e. cyclomatic complexity, variables size, etc.)

Code Verification

- Two verifications has been performed at code level by the Wizard:

1. **Coding Standard Verification**

   The Wizard verifies the MISRA-C 2012 standard for the code generated. Violations have been found:

   - Some can be solved by new implementation/settings
   - Some are related to automatic code generation not controlled by user

2. **100% Code Coverage**

   The wizard is used to verify 100% of code coverage:

   - Reference tests have been run in SIL (re-used tests from model coverage)
   - Add unit tests for coverage of specific C files
   - Justify not-covered code

- HTML reports are generated for the verifications

Conclusions & Lessons Learnt

Conclusions

- A detailed autocoding **methodology** has been **defined** to support development process of flight code (criticality of category B defined by ECSS standards) from Matlab/Simulink models
- A set of **modelling rules** and **guidelines** has been established by the methodology
- Commercial **tools** to support the autocoding methodology have been **evaluated**, selected and purchased in AUTOCOGEQ (integrated in the Wizard)
- A **Wizard** tool has been **developed** under Matlab to support the SW development phases → can be expanded and customized with integration of new rules
- The **autocoding methodology** proposed has been **demonstrated** using the Wizard on a real GNC simulator case
- The Wizard and methodology allows **quick verification** & recursive updates during all SW lifecycle
- Some **manual activity** still need to be performed for qualifying the generated code as category B as outcome of the analysis of the impact of autocoding on ECSS standards

Lessons Learnt

- Flight SW developed by models-based design and autocoding shall consider a well defined **methodology** from the **beginning** of lifecycle
- **Re-use** of models not implemented for generating flight code leads to a **big** adaptation **effort** → starting the SW development from scratch may be the best solution
- **Tailoring** of the code generation settings, modelling rules and code standards (e.g. MISRA-C) is needed according to projects needs
- **Tools** and automatic generation **cannot guarantee** the **qualification** of generated code as category B → tools support and complement the ECSS processes
- Still **additional manual activities** have to be performed to cover the complete ECSS processes for flight code qualification
- **Wizard** allows quick check of the rules and let the SW development process to be more flexible and recursive during all the phases but **does not make miracles** for generation of flight code

www.gmv.com

THANK YOU

Francesco Pace
fpace@gmv.com

gmv BLOG