Combination of AI support with A-SPICE: A case Study

Elisa Galliano – Head of Compliance

Vincenzo Sannino – AI Referent

OVERVIEW

- Introduction:
  - Goal
  - Context and Assumptions
- ASPICE framework definition
- Steps:
  - Functional requirements formulation
  - Algorithmic solutions and alternatives
  - Code generation based on SDD as input
  - Unit Test strategy formulation
  - Unit Test case generation
- Conclusions

Introduction and context

INTRODUCTION

- With the increasing complexity and availability of **Generative AI models**, the need to **integrate** them more and more **into corporate infrastructure** has become evident, making them active players in the development process.
- The use of Generative AI can **support engineers** at various stages of the product development lifecycle, acting as **“sparring partner”** for brainstorming and providing an efficient way to **accelerate time-consuming**, detailed implementation **tasks**.
- The goal of this presentation is to show how a Generative AI model can be **integrated into an existing A-SPICE-compliant company process** framework.

CONTEXT AND ASSUMPTIONS

- Attempts to integrate AI models have been carried out across various types of activities, applied to projects with varying levels of complexity.
- Taking into account the known limitations of LLMs, each test has been supervised by a subject matter expert and then submitted for review to potentially affected stakeholders.
- Each activity has also been supervised by the Quality Department to ensure that the outcomes align with the expected work product characteristics

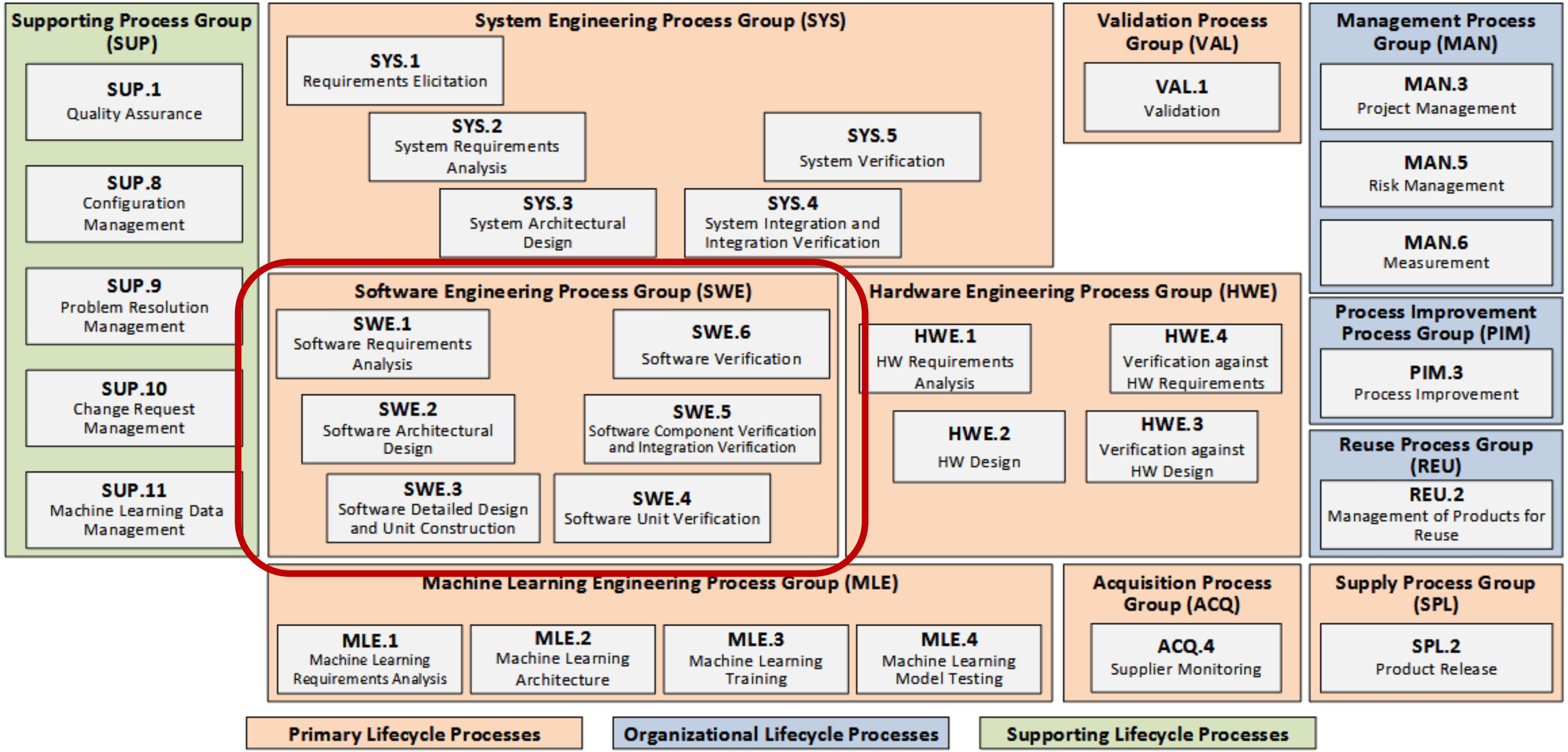
ASPICE FRAMEWORK DEFINITION

A-SPICE FRAMEWORK – 1/2

- Based on our accumulated experience with the capabilities of Generative AI systems, we decided to focus primarily on processes directly related to software development.

- This decision was made with several objectives in mind:
  - Optimising development time by delegating certain tasks to the AI
  - Adapting the process structure to support AI integration
  - Facilitating A-SPICE 4.0 process implementation through AI adoption

A-SPICE FRAMEWORK – 2/2

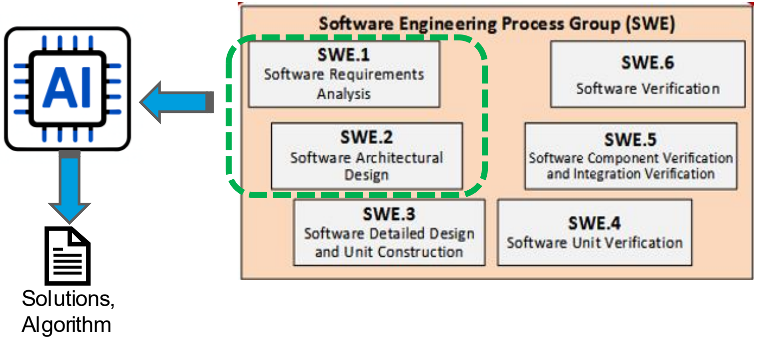


bylogix

SUBPROCESS SPECIFIC EXAMPLES

FUNCTIONAL REQUIREMENTS FORMULATION: SWE.1, SWE.2 TO SOLUTION PROPOSALS

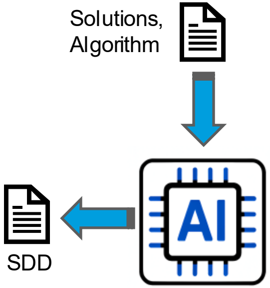
- Functional requirements related to a specific SW-Component are **extracted from SWE.1** and given as input to the AI Model.
- **System context is given in text form**, limited to the immediate functional **periphery** of the SW-Component.

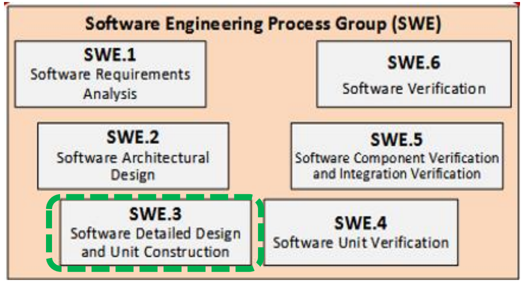


- Current AI models have **limitations** in the **analysis of complex UML hierarchical models**, more so in handling architectures with scopes bigger than single modules.
- For this reason, **allocation to architecture elements and interfaces (SWE.2)** is given directly and explicitly **by the operator**.
- Tool-supported extraction of architectural information from UML models in textual form is an option

SW DETAILED DESIGN SPECIFICATION GENERATION: SOLUTIONS TO SWE.3

- **Based** on the information presented from **SWE.1 and SWE.2**, the **AI model** is required to **figure out an algorithm** to implement the functional requirements.





- At this point **alternatives** can be explored and evaluated.
- Once an algorithm is selected, the AI is required to produce a **SW Detailed Design Specification**, considering the SW-Interfaces defined in SWE.2 and adding the specific SW-Interfaces deriving from the internal module architecture.
- The **tracking** of requirements from **SWE.1 and SWE.2 to SWE.3** is also carried out.

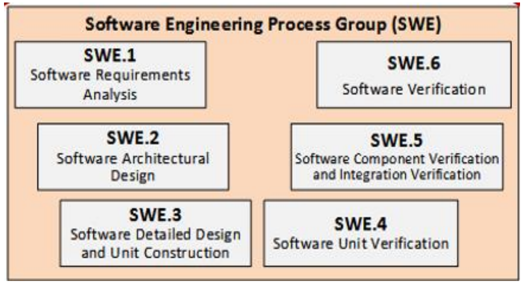
Prompts and outputs

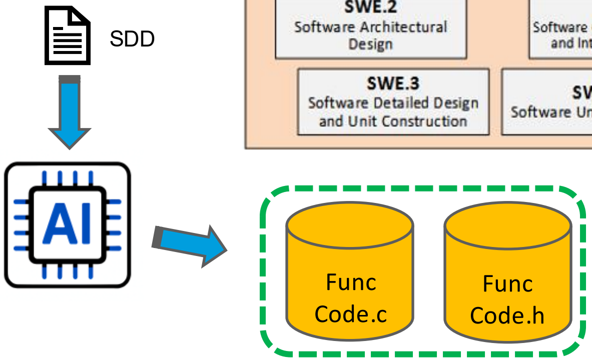
Prompt 3 SDD generation

Generated SDD

Code generation: SDD to code

- Once the SDD is refined and frozen, **code generation** can be started
- Depending on the prompt, code will consider:
  - **MISRA rules**
  - **FuSa/Cybersec** implementation guidelines
  - Custom **naming convention**
  - **API signature** defined in the SDD
- **Comments** can be generated in a given format e.g. **doxygen**
- **Requirement tracing** can be indicated in code as comments or in the heade each API/Data declaration





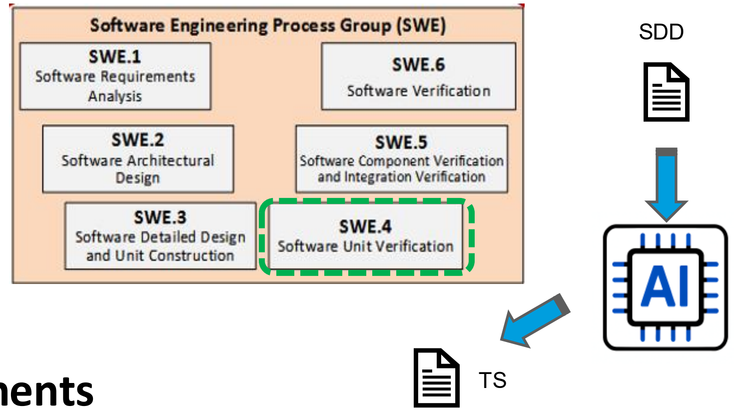
Prompts and outputs

Prompt 4 code mplementation.txt

TorqueGeneration.c

TorqueGeneration.h

UNIT TEST STRATEGY: SDD TO TS



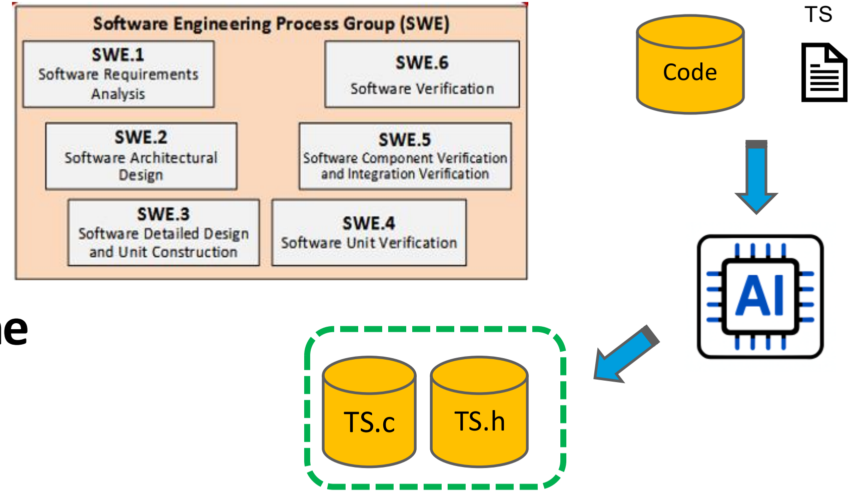
- Based on the SDD, the AI model is requested to **generate test cases** to cover all functional requirements and achieve statement, decision, branch coverage etc.
- Test cases will bear **reference** to the **tested requirements**
- By interacting with the AI model, it is possible to modify test strategies, to add specific test cases or achieve a better coverage
- Side note: as the previous contents were required to be ASPICE compliant, the TS is also compliant even without request in the prompt (in-context learning)

Prompts and outputs

Prompt 5 TS generation

Generated TS

Unit Test Generation: TC to Test Code



- Once the test specification is ready, **test cases** can be implemented **in code**
- Test scripts can be generated in a **test-frame agnostic way or based on known unit test frameworks** (e.g. GTest, Ceedling etc.)
- It is possible to submit some format information to the model in order to let it figure out specific test languages or configurations (e.g. Vector Cast).
- In the last case success rate varies significantly

Prompts and outputs

GTest Test Suite C-source

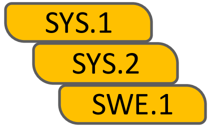
Ceedling Test Suite C-source

bylogix

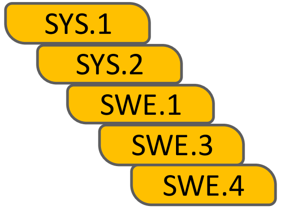
CONCLUSIONS

Conclusions

- AI can be used to rapidly explore a broad solution space for a problem, acting as an expert partner capable of delivering technical knowledge, with adjustable scope, depth, and technical detail.



- A similar approach can be applied when **standards** (e.g. ISO, IEEE, IEC), **regulations** (e.g. EC, EU, UNIECE) or **guidelines** (e.g. MISRA, internal QA frameworks) must be rapidly analysed to identify critical and relevant requirements or constraints.



- AI models inherently tend to **generate output variations**, even with identical or similar inputs. This is due to the underlying Transformer architecture.

- To enable **rigorous versioning and release management**, robust preconditions and **methods** must be implemented **to mitigate or control such variation** (e.g. precise prompt formulation, use of structured outputs, low temperature, top-k/top_p settings, etc.).

CONCLUSIONS

- Regarding SWE.4, AI models can provide **strong support** by generating a **large number of tests** to achieve wide coverage, as well as assisting in the rapid identification of **corner cases and negative tests**.

- AI models are capable of **generating unit test code**, either using a custom (proprietary) framework or compatible with open-source test frameworks (e.g. GTest, Ceedling, etc.).

- **Compatibility** with **commercial tools** requires **deeper evaluation**, as some formats are proprietary and therefore not publicly documented (resulting in limited training data availability). One approach is to **leverage in-context learning**, enabling the model to learn from examples and apply the acquired knowledge.

QUESTIONS?

THANK YOU!

CONTACTS

BYLOGIX srl

Strada del Portone, 159 - Grugliasco (TO) - Italy (Headquarters)
+39 011 740585

Via Costituzione, 9 - Maranello (MO) - Italy

bylogix

Francesco Ricciardi
CEO

francesco.ricciardi@bylogix.it