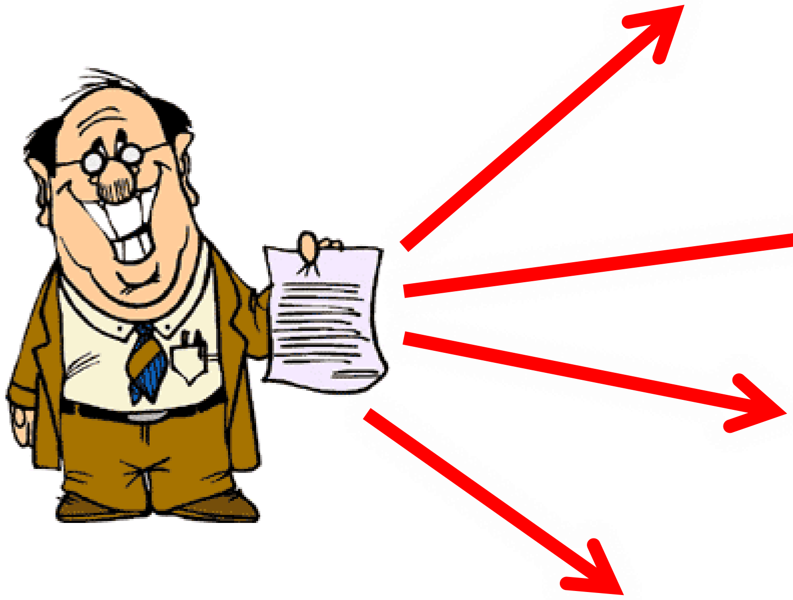
An incremental and DO-178C compliant process for Autopilot software development to make drones safer

Frédéric POTHON - ACG Solutions
frederic.pothon@acg-solutions.fr
Tel: (33)4. 67. 609.487
www.acg-solutions.fr

Amin ELMRABTI - Sogilis
amin@sogilis.com
Tel: (33)4. 26.780.507
www.sogilis.com

Valentin Brossard - Hionos
Valentin.brossard@hionos.com
Tel: (33)6.31.691.836
www.hionos.com

Agenda



1- Drones certification aspects

2- CAP2018 project

3- AGILE SW life cycle

4- Multi supplements

1- Drones certification aspects

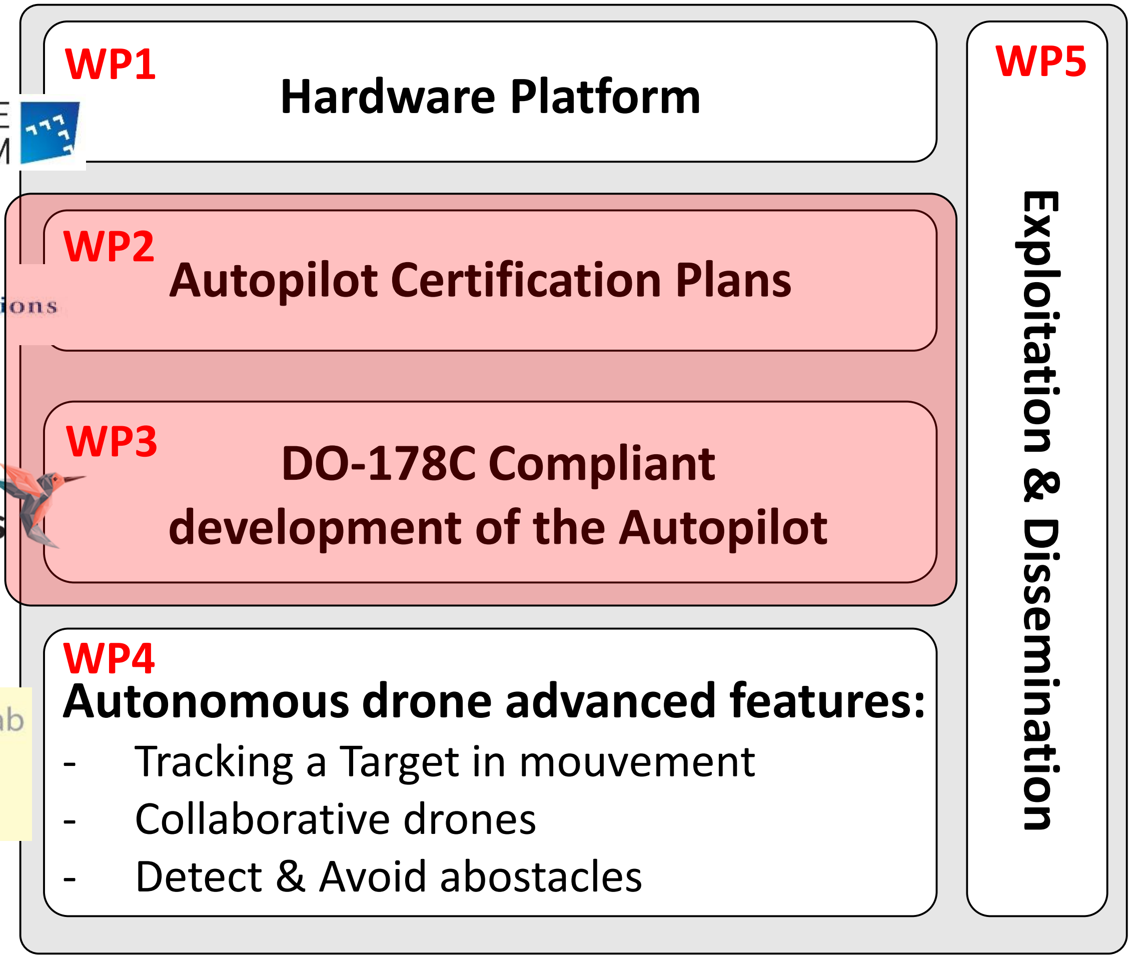
- Ongoing works
- Drone classification
  - Operations Risk-Based
    - Open
    - Specific
    - Certified
  - Classifications/categories still under discussion
- JARUS guidelines Specific Operations Risk Assessment (SORA) for the drones Specific category.
- EASA shall develop AMC (Acceptable Means of Compliance) and CSs (Certification specifications)
- Ongoing discussion on EUROCAE WG 105 work group to define applicable standards.

→ **We decide to anticipate and to develop drone software in compliance with DO-178C, Level A** →

C P2018

2- CAP2018 Project

- **CAP2018: Certified AutoPilot for 2018**
- **Objectives:** Develop a reliable and safe autopilot for civil commercial drones (Specific and Certified categories), compliant with DO-178C, Level A in the software part.
- **Project Structure:**


WP1
Hardware Platform
WP2
Autopilot Certification Plans
WP3
DO-178C Compliant development of the Autopilot
WP4
Autonomous drone advanced features:
- Tracking a Target in mouvement
- Collaborative drones
- Detect & Avoid abostacles
WP5
Exploitation & Dissemination

- **Partners:**
  - Sogilis
  - ACG-Solutions
  - AdaCore
  - Squadrone System
  - SNCF
  - CEA-Leti
  - Gipsa-Lab

2- CAP2018 Project

- SDVP : Software Development and Verification Plan
- SCMP : Software Configuration Management Plan
- SQAP : Software Quality Assurance Plan
- PSAC : Plan for Software Aspects of Certification
- Standards (Specification, Design, Coding)

- Deployment of an Agile Software lifecycle
- Deployment of various tools: Generic Test Framework, AdaCore Tools (for Ada2012 and Spark), Requirement Manager Tools, Versioning Management, Continuous Integration
- Deployment of a Multi-supplement processes

WP1 Hardware Platform

WP2 Autopilot Certification Plans

WP3 DO-178C Compliant development of the Autopilot

WP4 Autonomous drone advanced features:
- Tracking a Target in mouvement
- Collaborative drones
- Detect & Avoid abostacles

WP5 Exploitation & Dissemination

CAP2018

3- AGILE Software Life Cycle

Basic principles

- Incremental development

  To develop progressively a limited number of new functions/features. Iteration consists in adding/modifying some features in the data developed during the previous iteration cycles.

- Test Driven Requirement

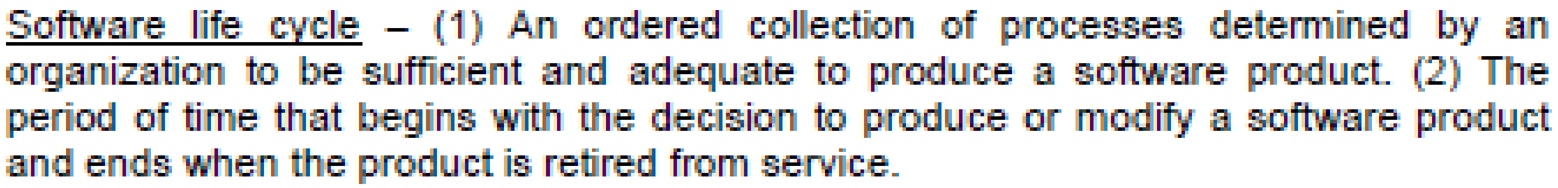
  Tests cases and requirements developed in the same process. All tests re-executed at each delivery

- Continuous improvements

  Application of plans and standards analyzed at the end of each iteration and identification of possible improvements

3- AGILE Software Life Cycle

Life Cycle "processes"



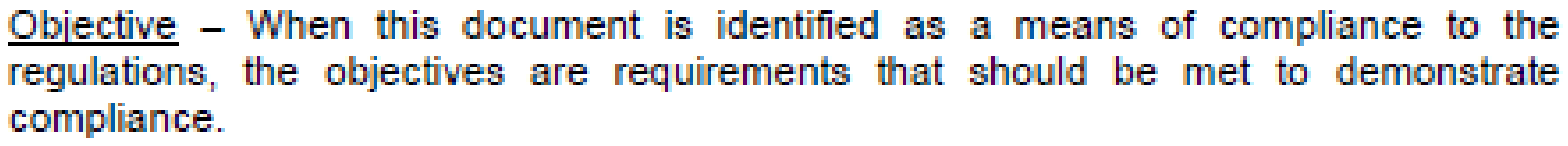
Software life cycle – (1) An ordered collection of processes determined by an organization to be sufficient and adequate to produce a software product. (2) The period of time that begins with the decision to produce or modify a software product and ends when the product is retired from service.

DO-178C identifies several processes: planning, development processes including requirements, design, coding, integration, verification, CM, SQA, and certification liaison.

But is it possible to propose another "ordered collection of processes"?

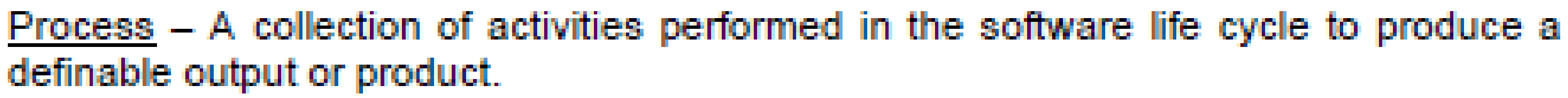
3- AGILE Software Life Cycle

Life Cycle “processes”



Objective – When this document is identified as a means of compliance to the regulations, the objectives are requirements that should be met to demonstrate compliance.

The purpose of the life cycle is to organize the activities in order to satisfy the objectives. Activities are grouped to form processes



Process – A collection of activities performed in the software life cycle to produce a definable output or product.

Yes, processes may be organized as needed to create an efficient life cycle, and to satisfy the objectives

3- AGILE Software Life Cycle

CAP 2018 Selected Life Cycle

Development and verification activities are grouped into “ticket”

A ticket is a “process”

Tickets are

- Type 1: Process or System Improvement
- Type 2: Software Requirements
- Type 3: Architecture
- Type 4: Component
- Type 5: Source code
- Type 6: Integration
- Type 7: PDI Instance
- Type 8: Problem Report
- Type 9 Delivery

3- AGILE Software Life Cycle

CAP 2018 Selected Life Cycle

Each ticket include several activities

Example Ticket Type 2: Software Requirements

- *Software requirement development*
- *PDI usage domain definition*
- *HW/SW interface description*
- *Software test cases development*
- *Software requirement self-check (no independence)*
- *Software requirement review (independence)*
- *Software Requirement coverage analysis*

During implementation, tickets are re-assigned to ensure independence, when applicable

3- AGILE Software Life Cycle

CAP 2018 Selected Life Cycle

Based on ticket implementation, status changes

⇒ **Open** at the ticket creation

⇒ **ToDo:** To be implemented during the iteration

⇒ **Assigned**: Resources identified and transition criteria met

⇒ **Implemented**: Data developed and/or updated and verified (no independence)

⇒ **Verified**: Data developed and/or modified verified with independence

⇒ **Done**: at the end of iteration when all activities have been completed and in case of errors or incompleteness, one or several Tickets Type 8 have been opened.

3- AGILE Software Life Cycle

CAP 2018 Selected Life Cycle

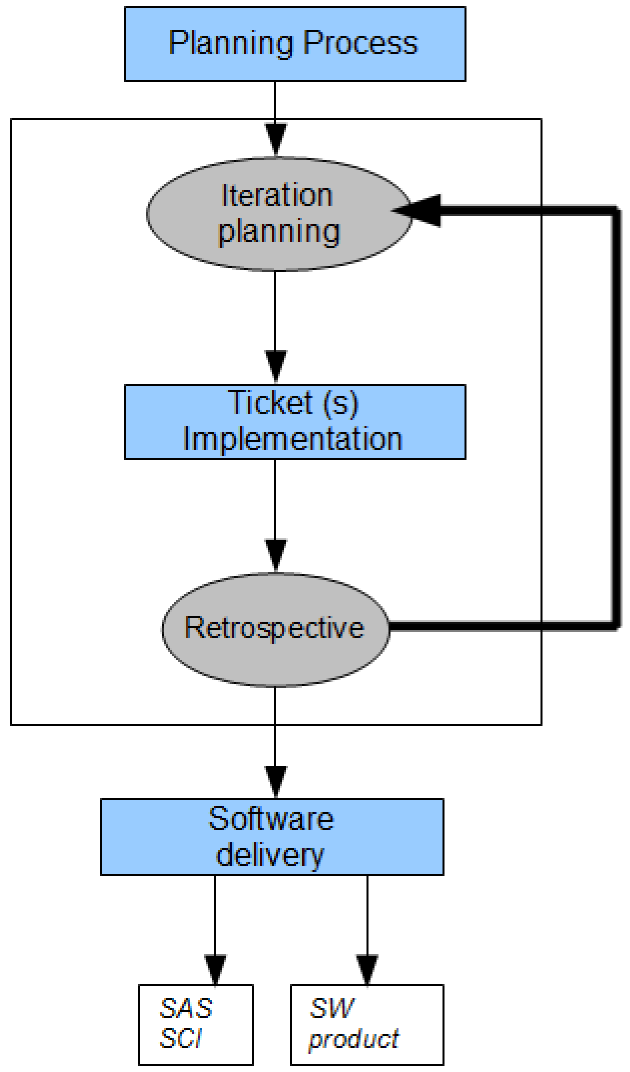
The iteration planning phase identifies a set of tickets to be addressed during the iteration in regard of

- Transition criteria (defined for each ticket)
- Priorities
- Available resources
- Need for delivery

Problems are recorded into ticket type 8. They are analyzed and one or several tickets of other types are created to correct the problem.

3- AGILE Software Life Cycle

CAP 2018 Selected Life Cycle



4- Multi-Supplements

Component development: 3 methods may be applied:

- Simulink Model and QGen qualified code generator

  => DO-331/ED-218 - Model Based Supplement (and tool qualification)

- Formal Model (SPARK functional specifications also called Contracts)

  => DO-333/ED-216 Formal Model Supplement (and tool qualification)

- Textual requirements in natural language and use of Object Oriented Techniques

  => DO-332/ED-217 OOT Supplement

4- Multi-Supplements

Our rules:

- One component = One method: No mix of methods inside the same component

- Considerations on architecture
  Decomposition allows the component development using a single development method

- Simulink Model is identified as a single component in the architecture. All requirements allocated to Simulink model are allocated to that component

4.1- Multi-Supplements - Model

Model Development with Simulink and QGen

- One software requirement => One or several functional “blocks”
- The blocks may be decomposed in sub-blocks as necessary.
- As part of this decomposition, the developer identifies in each development branch when a block represents a component requirement.
- Then a trace data is provided between the block and one or several software requirements.
- No derived requirements in the model
- Trace data developed using naming convention between block name and Requirement_Id
- Code Ada generated with QGen (Qualified as Criteria 1/TQL-1)

4.1- Multi-Supplements - Model

Model Verification

- Model simulation performed based on Software Requirements allocated to the model.
- Reviews of simulation data (independence).
- Model coverage analysis

Use of Mathworks simulation environment and tools.

Credit claimed of using QGen on source code verification and component tests

4.2- Multi-Supplements - OOT

Component Development using OOT features

OOT: Part of coding process (not design!)

Component requirements: Textual requirements (No OOT aspects)

Coding:

- One component ⬄ One Ada package. Use of naming convention
- UML class and/or sequence diagrams may be developed for the component (optional)
- OOT implemented with Ada features

4.2- Multi-Supplements - OOT

Vulnerabilities analysis: Most of the vulnerabilities are addressed through coding standard rules. Verification automated with Gnatcheck tool.

Example: Inheritance : Combination of 3 means

Vulnerabilities directly addressed, as not possible in Ada2012

- multiple implementation inheritance
- class' attributes overriding

Limitations through coding standards:

- No static dispatch except for subclasses calling their parents.
- No deactivated code: In every overriding method, the first statement is the call to the overridden method.
- multiple interface inheritance: a subclass cannot inherit from two interfaces having the same procedure name

Pessimistic testing: When inheritance is used, all added behaviour is tested.

4.3- Multi-Supplements – Formal method

Component Development and verification using Ada contracts

- Formal analysis is applied on component for which it is possible to describe all their requirements in form of Ada contracts.

SPARK technology is based on

- A formal model called SPARK contracts
- A formal analysis performed on SPARK contracts using GNATProve tool.

4.3- Multi-Supplements – Formal method

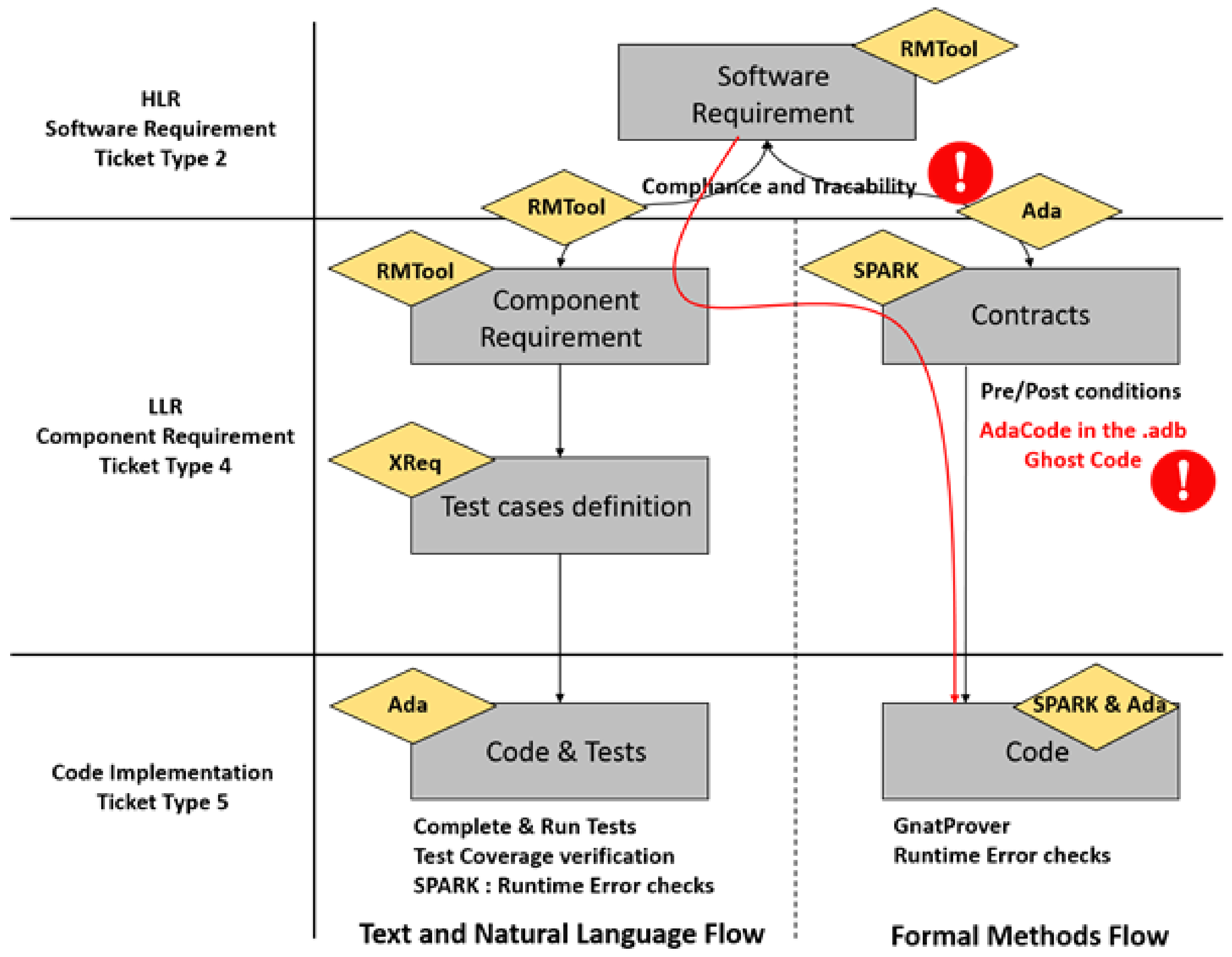


Figure 12. Evolution of the development process with formal methods techniques

4.3- Multi-Supplements – Formal method

Formal method: Key points

Credit claimed
Source code verification objectives (compliance to LLR, traceability, accuracy, consistency) : Directly satisfied by the Formal analysis:
Components tests : Satisfied by the Formal analysis and the property preservation between source and object code

“Soundness of method”
Description of the method in the PSAC, using international publications references.

“Property preservation between source and object code”
Demonstration achieved through a second execution of the software tests on EOC including the contracts in form of assertions

4.3- Multi-Supplements – Formal method

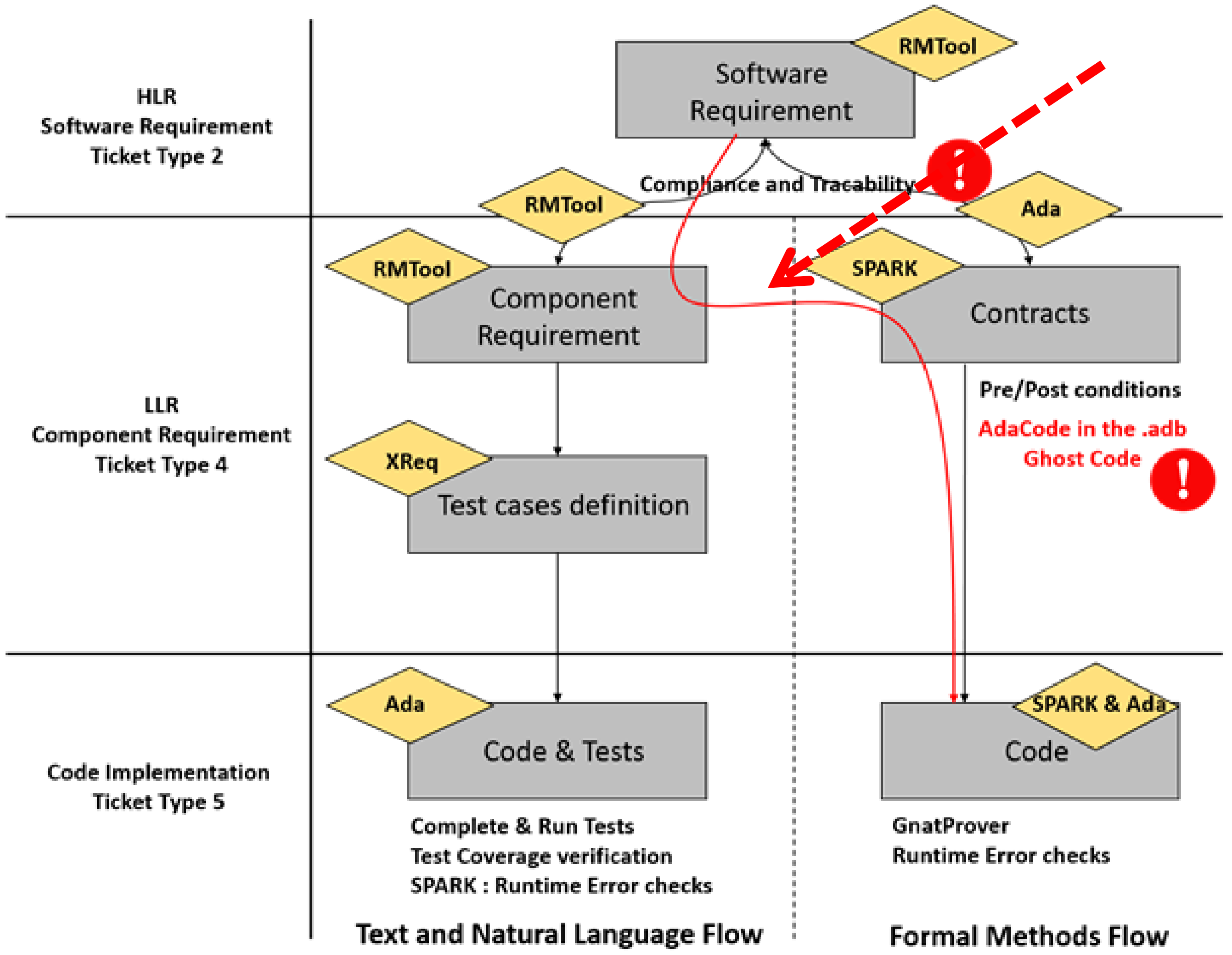


Figure 12. Evolution of the development process with formal methods techniques

4- Multi-Supplements

PSAC: DO-178C objective coverage presentation

Table A-6 example

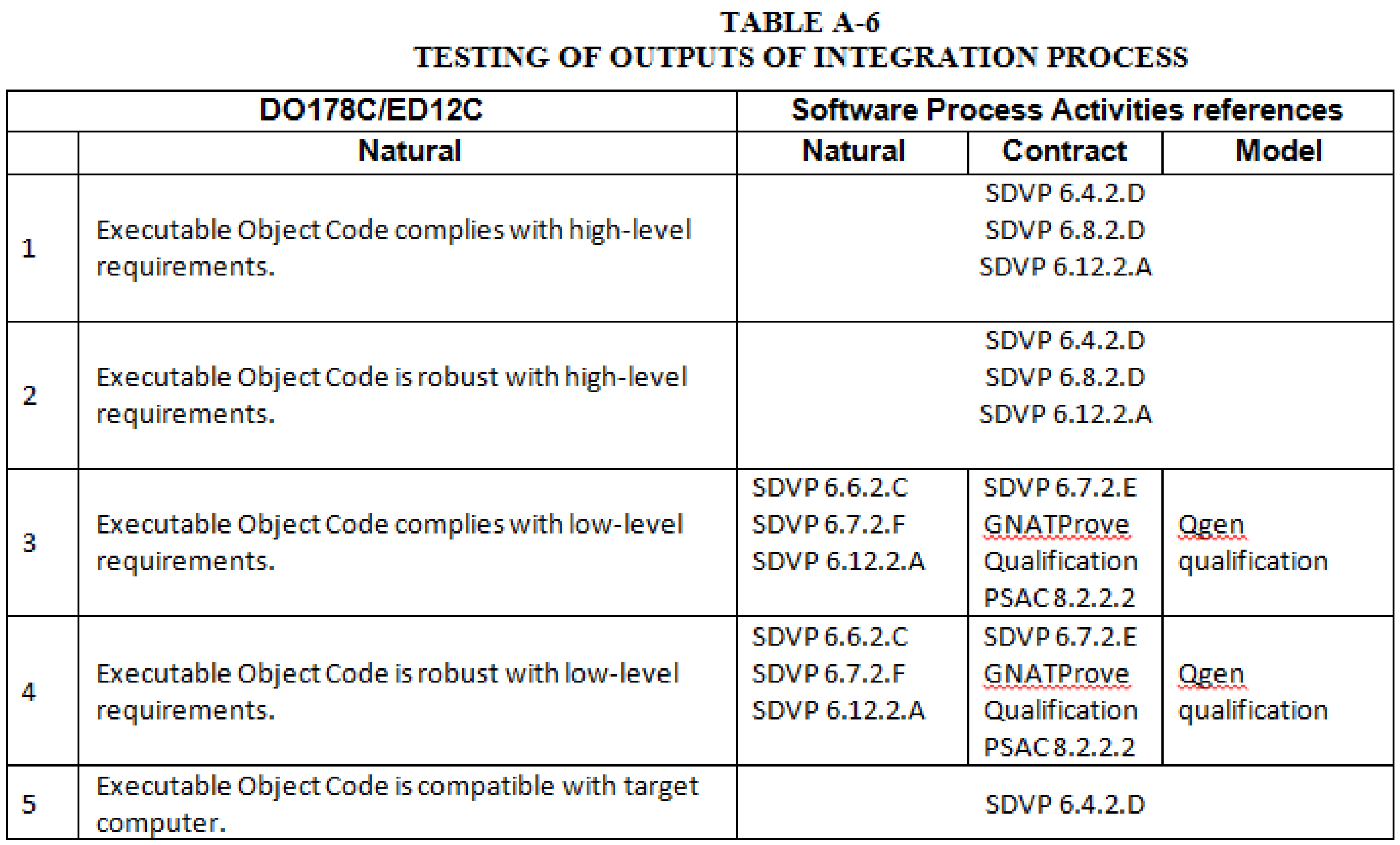


TABLE A-6
TESTING OF OUTPUTS OF INTEGRATION PROCESS

DO178C/ED12C		Software Process Activities references		
	Natural	Natural	Contract	Model
1	Executable Object Code complies with high-level requirements.	SDVP 6.4.2.D SDVP 6.8.2.D SDVP 6.12.2.A		
2	Executable Object Code is robust with high-level requirements.	SDVP 6.4.2.D SDVP 6.8.2.D SDVP 6.12.2.A		
3	Executable Object Code complies with low-level requirements.	SDVP 6.6.2.C SDVP 6.7.2.F SDVP 6.12.2.A	SDVP 6.7.2.E GNATProve Qualification PSAC 8.2.2.2	Qgen qualification
4	Executable Object Code is robust with low-level requirements.	SDVP 6.6.2.C SDVP 6.7.2.F SDVP 6.12.2.A	SDVP 6.7.2.E GNATProve Qualification PSAC 8.2.2.2	Qgen qualification
5	Executable Object Code is compatible with target computer.	SDVP 6.4.2.D		

Conclusion

POWER
SPI
USB
TELEM1
TELEM2
SPKT/DSM
ADC6.6V
I2C
CAN
GPS
SWITCH
SERIAL 4/5
BUZZER
FMU
PWR
B/E
ACT
B/E
PWR
IO
pixhawk
2.4.8
AUX OUT
MAIN OUT
SBUS
RCIN
desk 02

Conclusion

Conclusion

Conclusion

Pulsar Autopilot

Safety

- DO-178 Compliant Software (DAL A)
- Use of Formal Proof Mechanism
- 5 different Failsafes (GPS, Com, Batt, Compass)

DO-178C

Features

- Manual Flight
- Automatic Flight (including takeoff and landing)
- Available for different Frames and Electronics

HW / Frame agnostic

Configurability

- Dedicated board for customization
- Free Software Development Kit
- Free Libraries

Easily customizable

Hionos™

hionos.com

contact@hionos.com