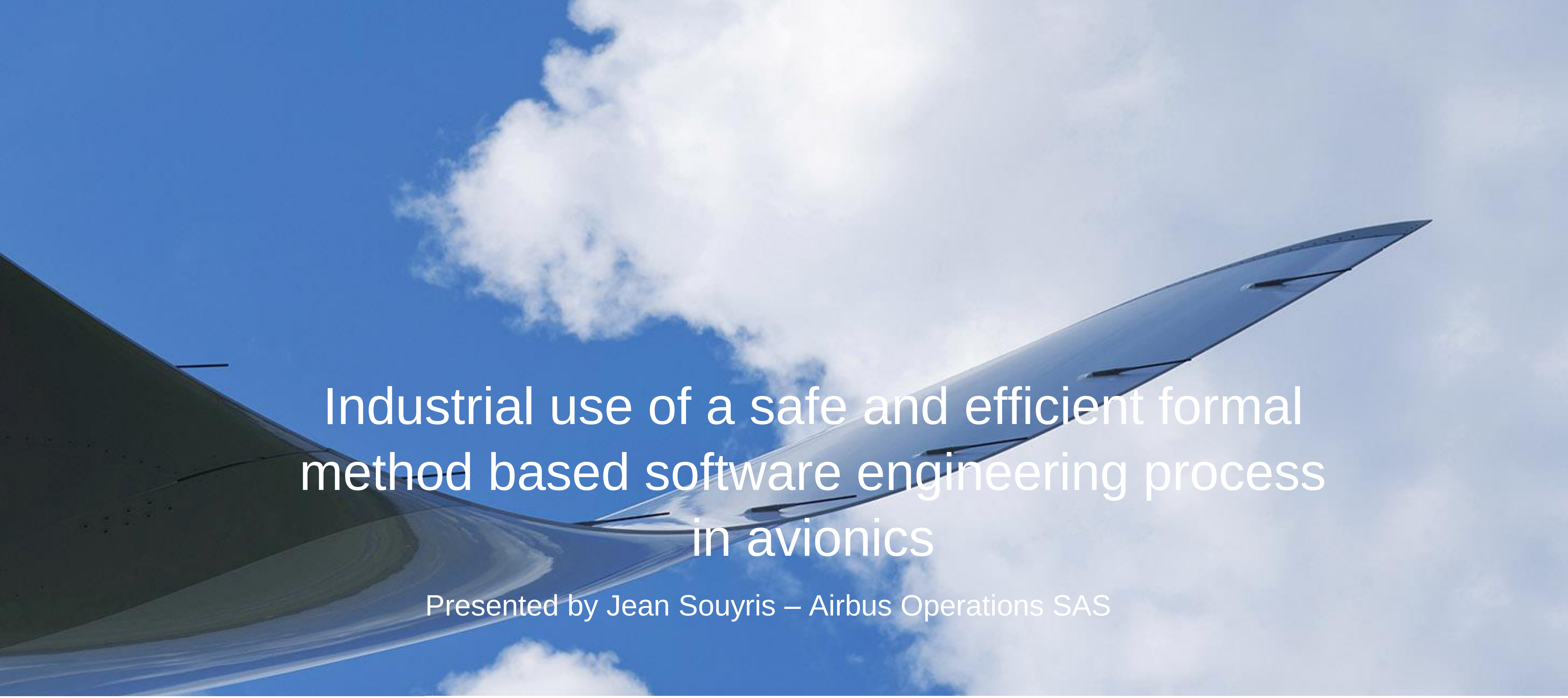


Industrial use of a safe and efficient formal method based software engineering process in avionics

Presented by Jean Souyris – Airbus Operations SAS

Abderrahmane Brahmi*, Marie-Jo Carolus*, David Delmas*, Mohamed Habib Essoussi*, Pascal Lacabanne*, Victoria Moya Lamiel*, Famantanantsoa Randimbivololona** and Jean Souyris*.

*Airbus Operation S.A.S

**Cepresy September 2019

ERTS 2020

AIRBUS

Summary

New Avionics development process (= New Way Of Working)

Artefacts, activities and verification objectives

The New Way Of Working (NWOW) Workshop

Formal design

Functional and non functional automated verification

Compilation

Process management and build

Industrial deployment and feedback

Deployment pre-requisites, statistics, positive aspects and room for improvement

Avionics development process (NWOW) (1/2)

Artefacts of the development process (DO178C):

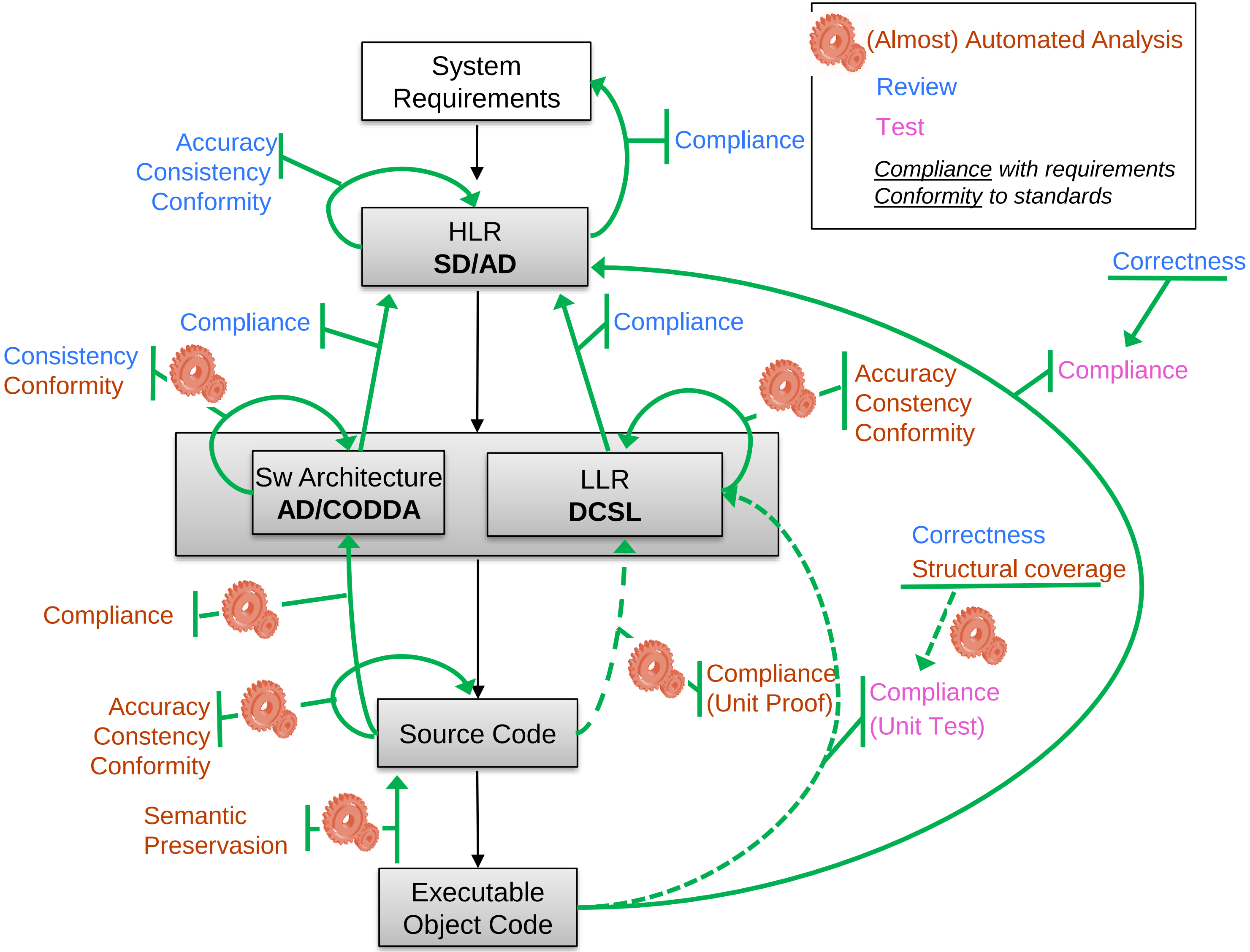
- High Level requirements
  - SD = Specification Data
  - AD = Architecture Data
- *Software Architecture*
  - *Static architecture in CODDA language*
- *Low Level Requirements*
  - *Formal contracts in DCSL language*
- *Source code: in C and Assembly languages*
- *Executable Object Code*

Kind of verification activities*:*

- *Review: checklist based reading*
- *Automated analysis*
- *Test (based on formal notation)*

Kind of verification objectives*:*

- *Accuracy, consistency*
- *Conformity, Compliance*
- *Semantic preservation*



Almost all activities from software design down are automated

Avionics development process (NWOW) (2/2)

Design formalization allows to automate:

- A big amount of reviews of the design data, for accuracy, consistency, and conformity to standards.
- Unit verification with two alternatives: either Unit Proof or Unit Test.
  - Unit Proof is for C source code
  - Unit Test is the back-up of the Unit Proof for
    - Assembly code,
    - C code that cannot be proved (e.g. linked list)

Automatic process management:

- Process is made efficient by the tight integration of a number of automated techniques.
- This integration is orchestrated by a process management tool (Optimases).

Formalization and automaticity are key

The New Way Of Working Workhop (1/7)

Software design

Software architecture: CoDDA (Compilable Design Description Assistant)

- Method: static design by Abstract machines (adaptation of the Hood method)

- CoDDA language supports the formalization of the description of the abstract machines:
  - Exported interface and hidden implementation
  - Constants, types, resources (variables) and services (then implemented (coded) as C functions or assembly routines)

- CoDDA support for edition: CoDDA plug-in in Visual Studio Code

- The CoDDA tool main functionalities:
  - A checker of the design rules (correctness of the design)
  - A generator of: C or assembly code skeleton, documentation, traceability information and data for Unit Proof or Unit Test

The New Way Of Working Workhop (2/7)

Software design

Detailed design

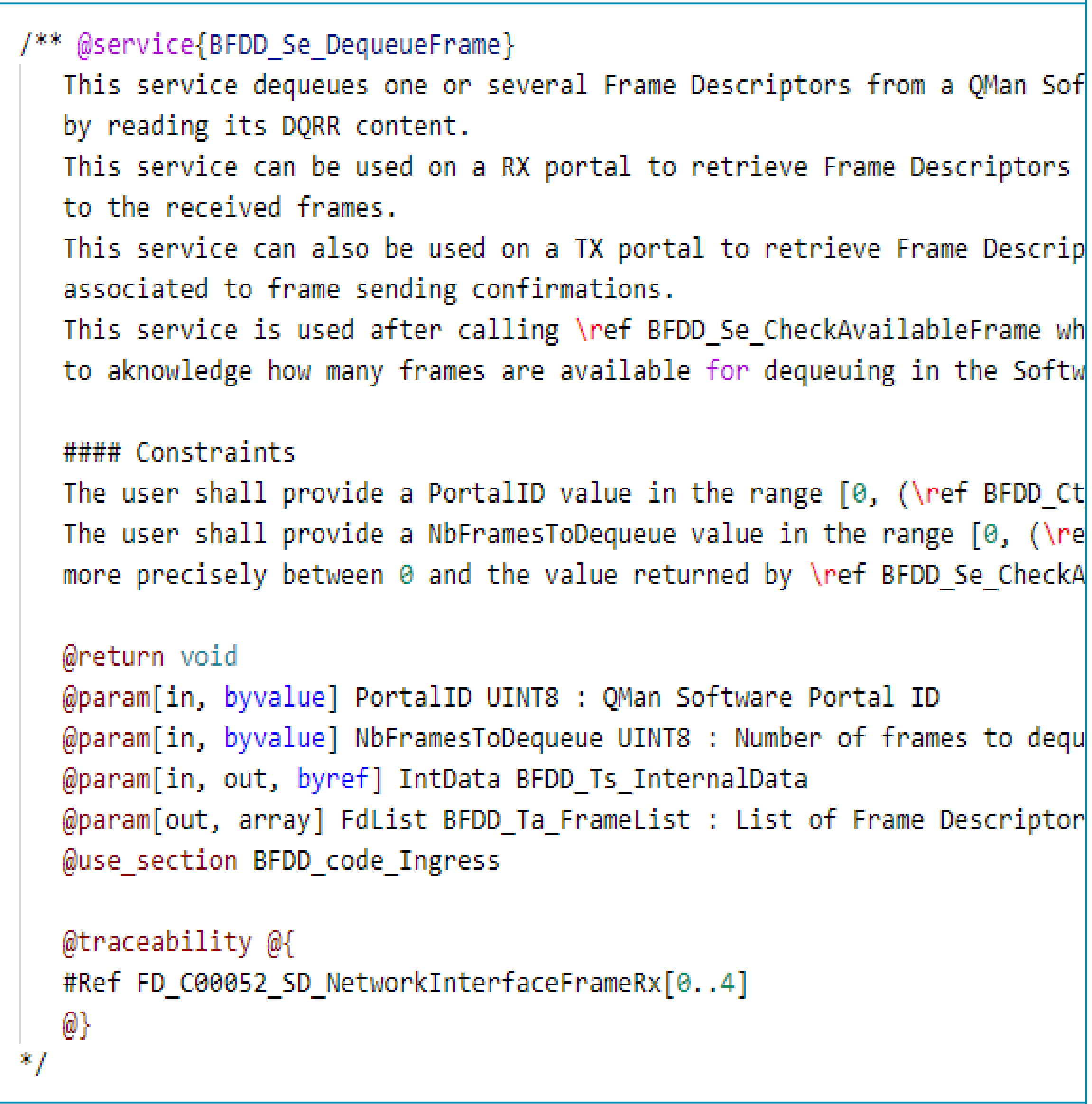
- Design Contract Specification Language, DCSL (Kind of Domain Specific Language for embedded software products)
  - Code-level Behavioral Interface Specification Language (BISL)
  - Based on ACSL (Ansi C Specification Language) with extensions and restrictions
  - Adapted to
    - Various kinds of software products/components
    - Component based development (+ notion of product line, variability)

- DCSL support for edition: DCSL plug-in in Visual Studio Code

- The DCSLC compiler
  - For proof: DCSL to ACSL translation + additional verification oriented constructs (e.g. handling of function calls, of accesses to volatile variables)
  - For tests: generation of C programs and declarations + predicate evaluators also in C (for test oracles)
  - For static analysis: generation of control/data flow annotations + value range annotations for validation of preconditions

Static and detailed designs are formal hence automatically exploited by verification tool chains

The New Way Of Working Workhop (3/7)

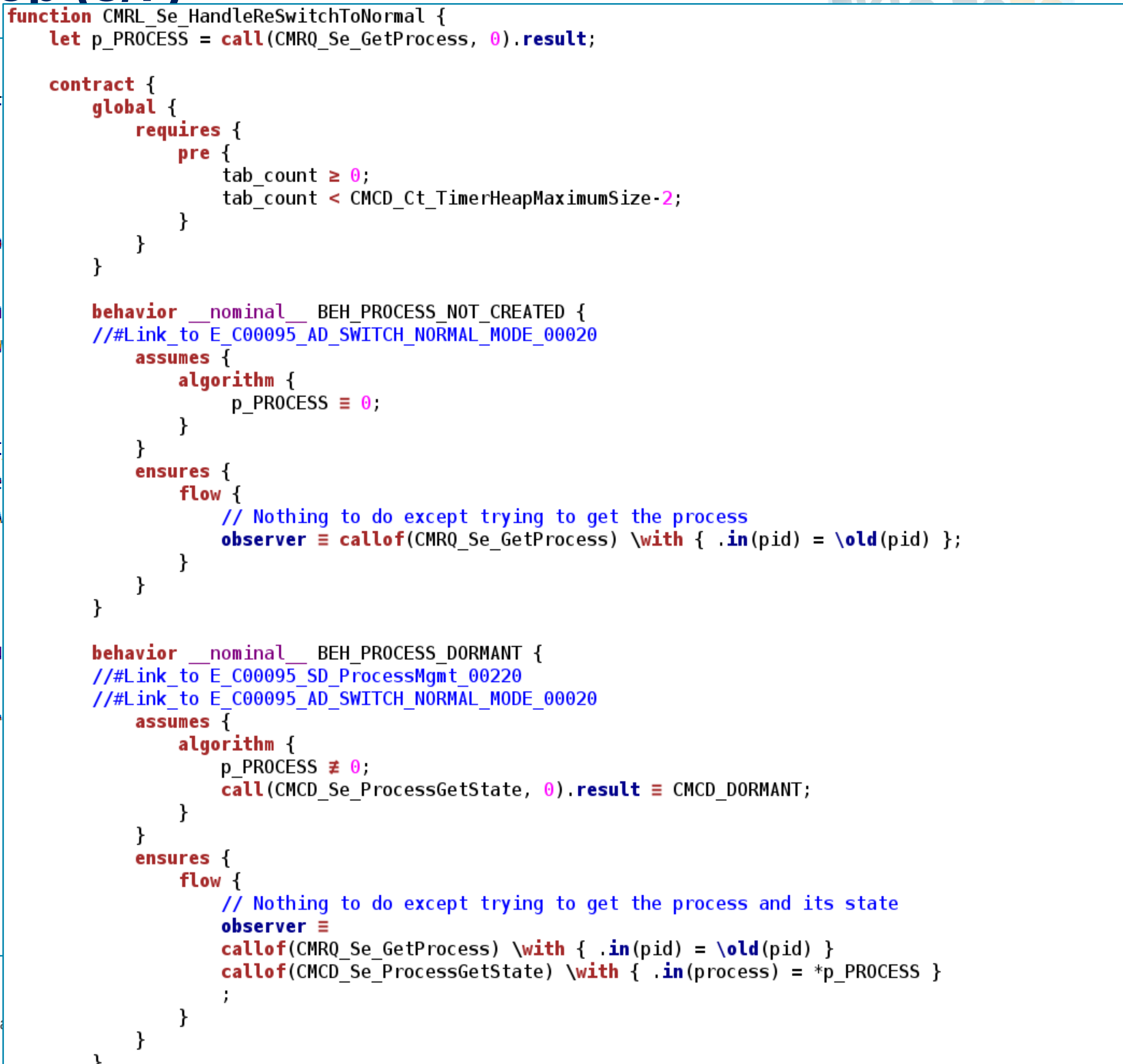


```
/** @service{BFDD_Se_DequeueFrame}
    This service dequeues one or several Frame Descriptors from a QMan Sof
    by reading its DQRR content.
    This service can be used on a RX portal to retrieve Frame Descriptors
    to the received frames.
    This service can also be used on a TX portal to retrieve Frame Descrip
    associated to frame sending confirmations.
    This service is used after calling \ref BFDD_Se_CheckAvailableFrame wh
    to aknowledge how many frames are available for dequeuing in the Softw

    #### Constraints
    The user shall provide a PortalID value in the range [0, (\ref BFDD_Ct
    The user shall provide a NbFramesToDequeue value in the range [0, (\re
    more precisely between 0 and the value returned by \ref BFDD_Se_CheckA

    @return void
    @param[in, byvalue] PortalID UINT8 : QMan Software Portal ID
    @param[in, byvalue] NbFramesToDequeue UINT8 : Number of frames to dequ
    @param[in, out, byref] IntData BFDD_Ts_InternalData
    @param[out, array] FdList BFDD_Ta_FrameList : List of Frame Descriptor
    @use_section BFDD_code_Ingress

    @traceability @{
    #Ref FD_C00052_SD_NetworkInterfaceFrameRx[0..4]
    @}
*/
```



```
function CMRL_Se_HandleReSwitchToNormal {
    let p_PROCESS = call(CMRQ_Se_GetProcess, 0).result;

    contract {
        global {
            requires {
                pre {
                    tab_count ≥ 0;
                    tab_count < CMCD_Ct_TimerHeapMaximumSize-2;
                }
            }
        }

        behavior __nominal__ BEH_PROCESS_NOT_CREATED {
        //#Link_to E_C00095_AD_SWITCH_NORMAL_MODE_00020
            assumes {
                algorithm {
                     p_PROCESS ≡ 0;
                }
            }
            ensures {
                flow {
                    // Nothing to do except trying to get the process
                    observer ≡ callof(CMRQ_Se_GetProcess) \with { .in(pid) = \old(pid) };
                }
            }
        }

        behavior __nominal__ BEH_PROCESS_DORMANT {
        //#Link_to E_C00095_SD_ProcessMgmt_00220
        //#Link_to E_C00095_AD_SWITCH_NORMAL_MODE_00020
            assumes {
                algorithm {
                    p_PROCESS ≢ 0;
                    call(CMCD_Se_ProcessGetState, 0).result ≡ CMCD_DORMANT;
                }
            }
            ensures {
                flow {
                    // Nothing to do except trying to get the process and its state
                    observer ≡
                    callof(CMRQ_Se_GetProcess) \with { .in(pid) = \old(pid) }
                    callof(CMCD_Se_ProcessGetState) \with { .in(process) = *p_PROCESS }
                    ;
                }
            }
        }
```

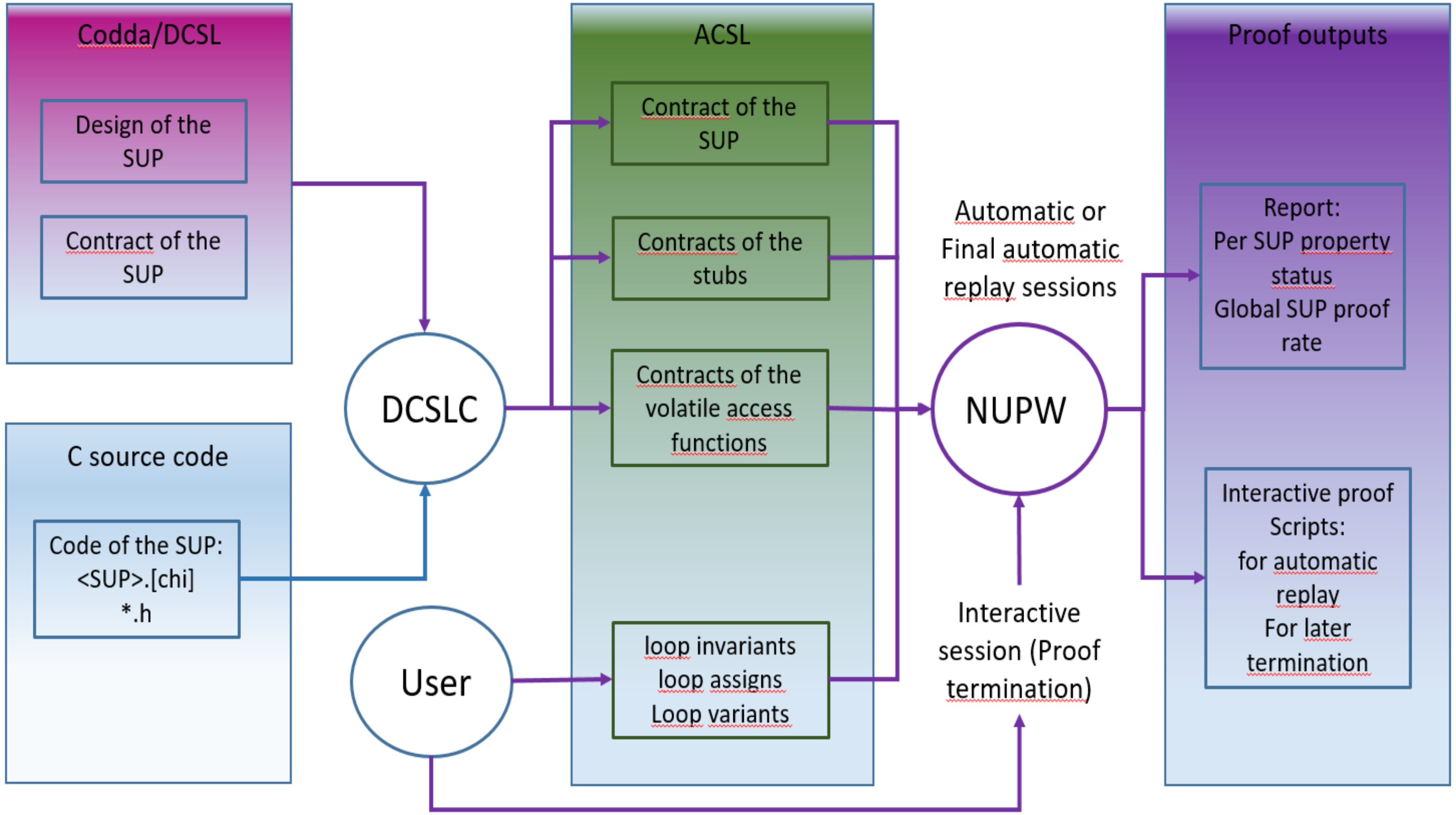
The New Way Of Working Workshop (4/7)

Functional Verification: the Unit Proof *tool* chain

- Proof of a C function against its DCSL contract, via translation DCSL to ACSL
- Proof principle: Dijkstra's Weakest Precondition + theorem proving
- **Proof tool: NUPW, based on "frama-c –wp" (CEA)**

- Fully automatic most of the time

- Loop annotations are provided by the user when loop unrolling is unsuccessful (most of the time, unfortunately)
- A set of guidelines support the user, mainly for writing loop annotations
- Cases of interactive proof termination are rare

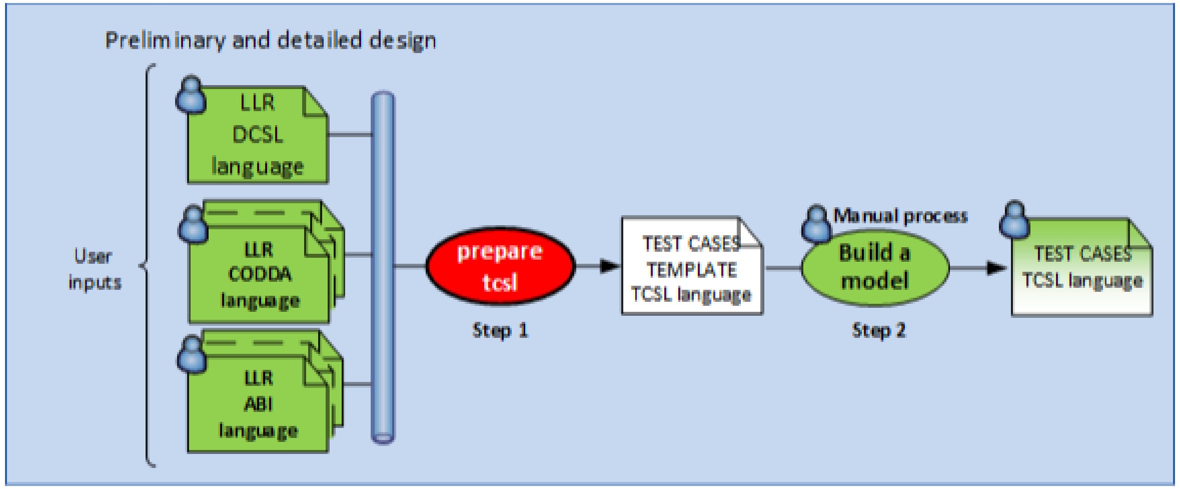


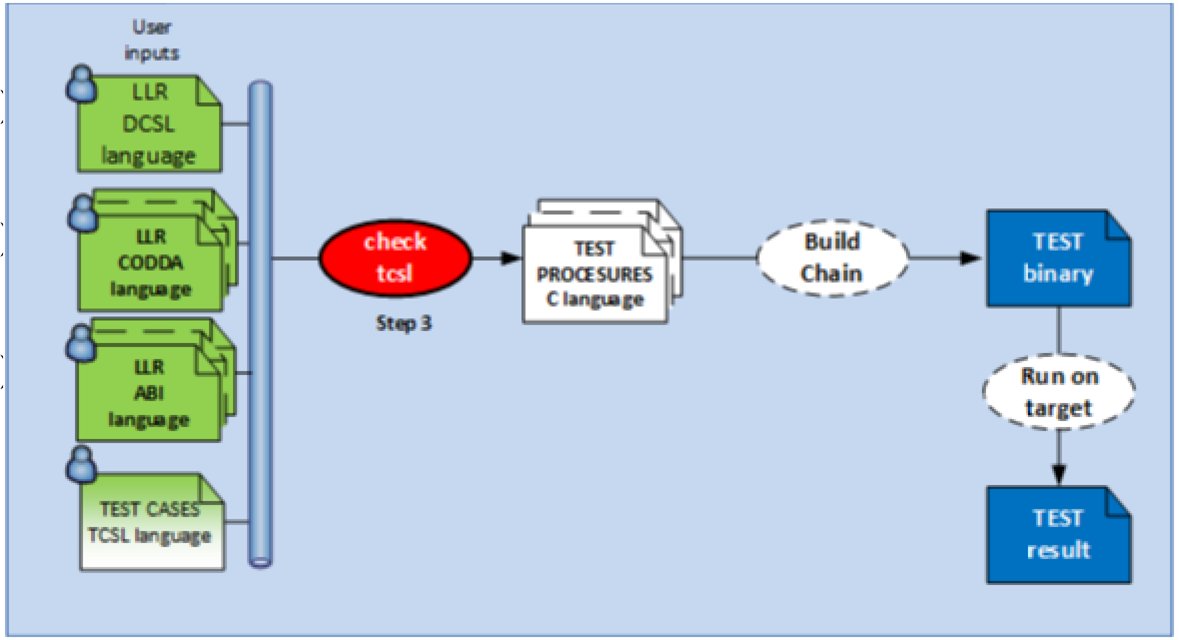
Codda/DCSL
Design of the SUP
Contract of the SUP
C source code
Code of the SUP: <SUP>.[chi] *.h
DCSLC
User
ACSL
Contract of the SUP
Contracts of the stubs
Contracts of the volatile access functions
loop invariants
loop assigns
Loop variants
Automatic or Final automatic replay sessions
NUPW
Interactive session (Proof termination)
Proof outputs
Report: Per SUP property status Global SUP proof rate
Interactive proof Scripts: for automatic replay For later termination

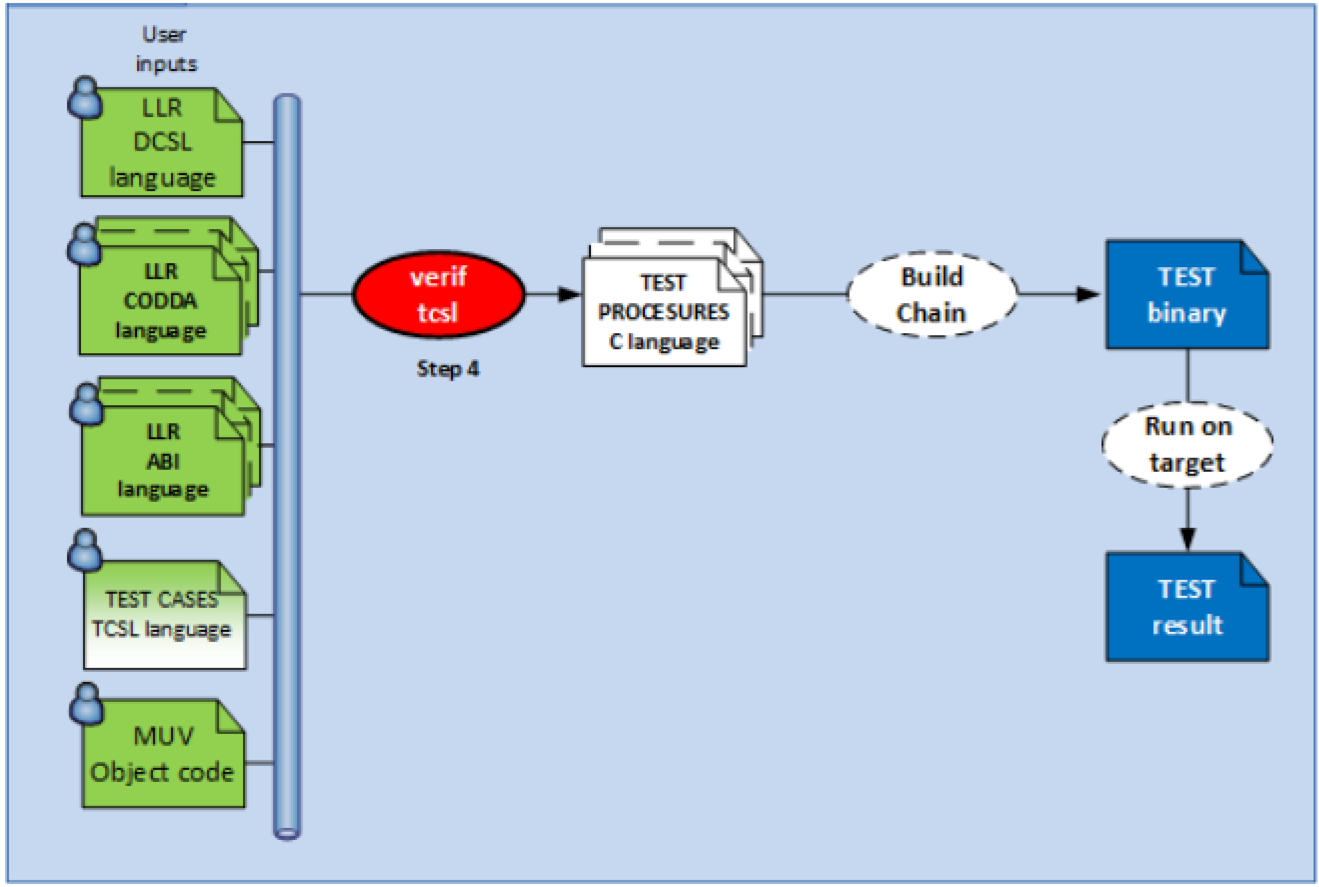
Fully automated Unit Proof tool chain

The New Way Of Working Workhop (5/7)

Functional Verification: Unit Test tool chain







Partially automated Unit Test tool chain

The New Way Of Working Workhop (6/7)

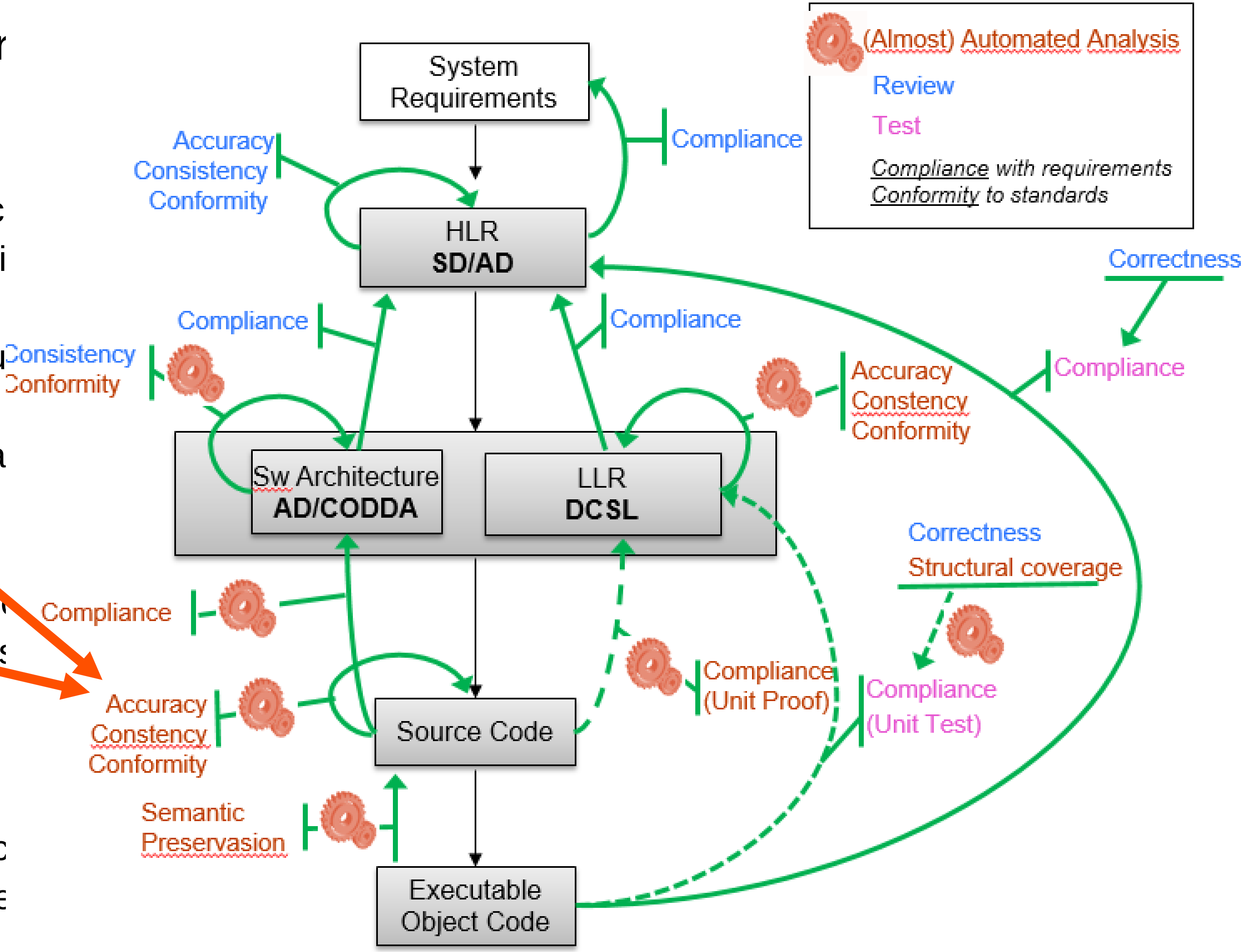
Non functional Verification (Abstract Ir

Anafloat toolchain

- Evaluation of the numerical accuracy of library c
- Automated activity: accuracy / consistency readi
- Main tool: **Fluctuat** (CEA)
    - Implementation error, i.e. between a compu computed
    - Error of method (e.g. polynomial approxima

CheckRTE toolchain

- Proof of absence of Runtime Errors (RTE) on a
    - RTE = division by zero, overflows, accesses etc
- Automated activities:
    - Accuracy / consistency analysis
    - Validation of DCSL preconditions unit verific
    - Validation of hypotheses unit verifications re
- Main tool: **Astrée** (AbsInt GmbH)

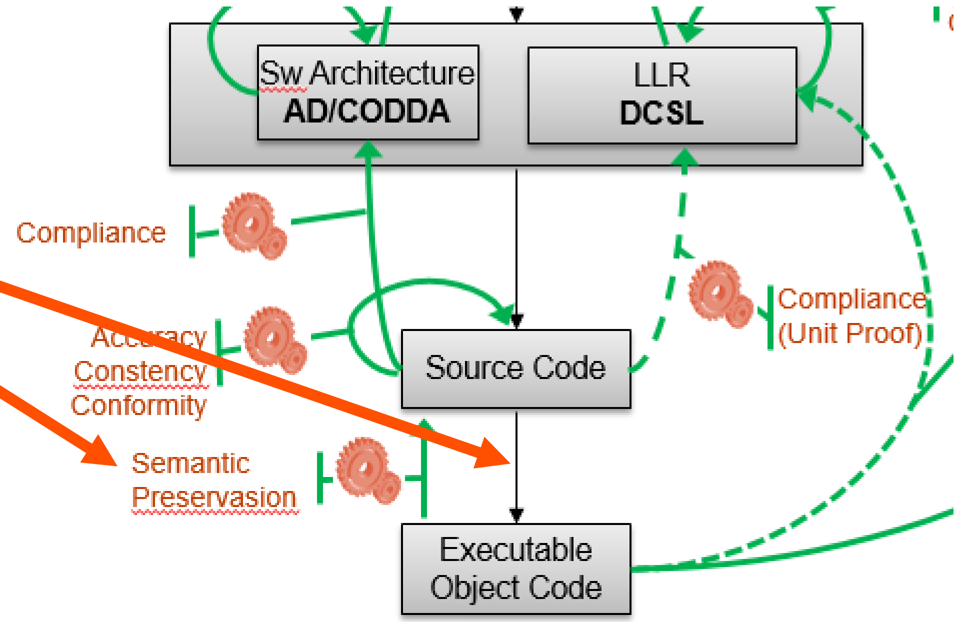


Abstract interpretation based static analyzers achieve what human readers can't

The New Way Of Working Workhop (7/7)

Compilation: CompCert (AbsInt GmbH + INRIA)

- *Formally developed C compiler*
- *High level of confidence ==> C code / Object semantic preservation is strongly established*
- *Proofs at C level are then lifted down to the object code*



CompCert contributes to the compliance to DO-333 (Formal Methods)

Optimases

- *Process management and build system*
- *Processes are configured through XML collections, with the notions of*
  - File types, tool definition, variables, process templates and variants

Optimases is the "orchestra conductor"

Industrial deployment and feedback (1/3)

The balance presented now is an ***intermediate*** “lesson learnt” after two years of exploitation of the NWOW

Achievements (pre-requisites) before starting the exploitation

- *Good maturity level achieved thanks to mock-ups*
- *Guidelines, methodological documents and trainings*
- *Support and maintenance organization and tool*

Some statistics

- *All three new avionics software product developments are made according to the NWOW*
- *About 60 developers have been working according to the NWOW*
- *179 abstract machines developed with CoDDA*
- *3315 C functions and 230 (0.65% of the total) assembly routines*
- 98.5 % of the C functions are Unit-proved, the other ones being Unit-tested
- 336 (10%) C functions necessitated the writing of loop invariants
- 75 (2.3%) functions required the interactive termination of some of their proofs

The NWOW is mandatory for every new development (in-house Airbus avionics products)

Industrial deployment and feedback (2/3)

On the positive side (major points, for details see the paper)

- *Adequacy to the regulatory framework*
  - ***Plans*** (Software Development Plan, Software Verification Plan, etc) were ***accepted*** by the authority
- *The adequacy to the applicative context needs*
  - Very good initially and ***continuous improvements*** (from users needs emerging during operation)
    - Examples: temporal logic extension of DCSL, the handling of complex structures (strings, linked lists)
- *The skills for performing and completing the activities*
  - Formal methods are taught in engineering schools / universities
  - Each developer ***follows a 12-day training*** on the NWOW before starting developing
  - ***Continuous support*** by NWOW specialists
- *Quality of the development artefacts and data*
  - ***Design*** (CoDDA) and ***detailed design*** (DCSL contracts) are ***a lot more rigorous***
  - ***Exhaustive verification*** of formal proof and abstract interpretation based static analysis
- *Respect of the development schedules*

Expected benefits of the NWOW are actually observed

Industrial deployment and feedback (3/3)

Room for improvement

- *Quality of the development artefacts and data*
  - “Excessive splitting” in machines/functions is sometimes observed
  - “Code writing before contract writing” happens sometimes
  - ***Improvement***: stricter process checks, enhanced reading checklists
- *Adequacy to the applicative context needs*
  - DCSL
    - Lack of DCSL operators/constructs
    - ***Improvement***: new specific constructs, user defined operators/functions
  - Unit Proof
    - ACSL appears as « yet another language to know », i.e. for writing loop contracts
    - ***Improvement***: give the user the capability to write invariants in DCSL
  - Unit Test
    - Test cases definition is up to the user
    - ***Improvement***: heuristics for deducing some test cases from the DCSL contracts
- *Skills for performing the activities*
  - The design of some abstract machines required more effort/rework than expected
  - ***Improvement***: strengthen the developer’s ability to master the writing of formal design from non-formal upstream artefacts

Some necessary adjustments

Conclusion

ERTS 2020

The NWOW is mandatory for every new development (in-house Airbus avionics products)

Expected benefits of the NWOW are actually observed

Some necessary adjustments

Globally very positive

Complete lessons learnt after completion of the first NWOW compliant developments

Thank you