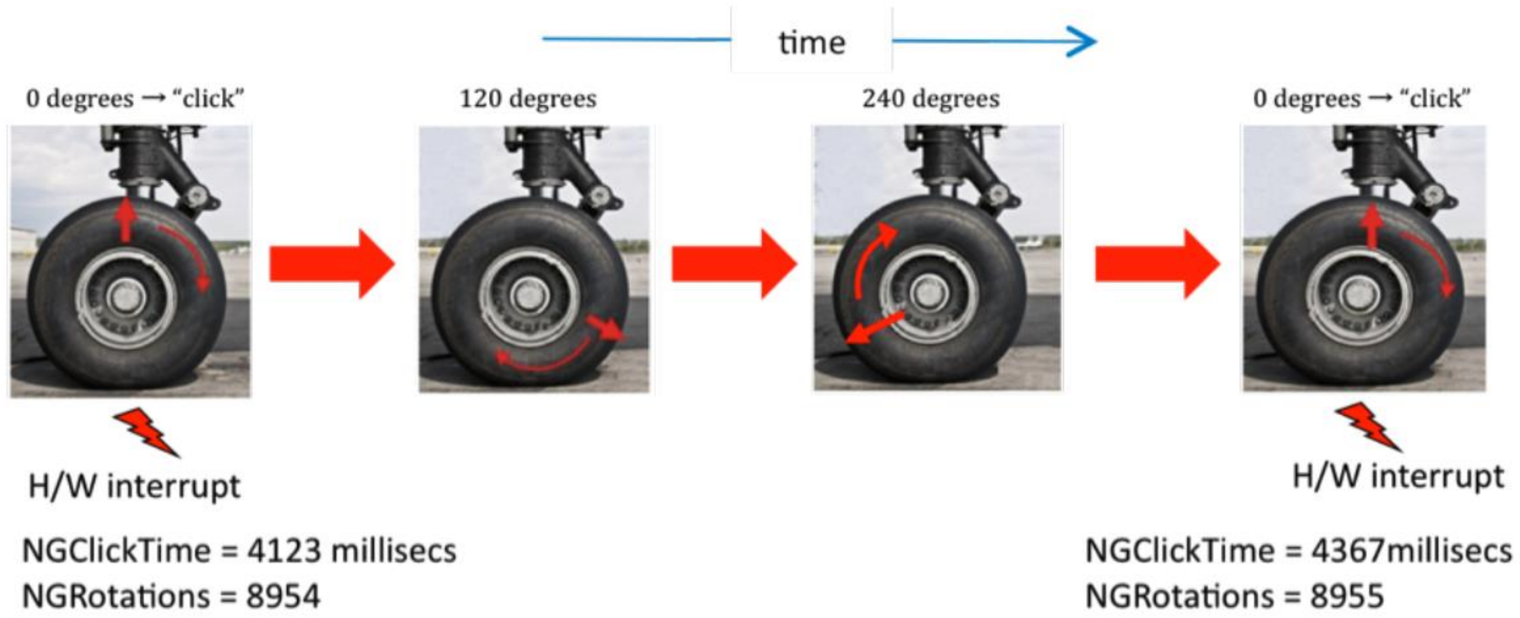
AdaCore

A Complete Solution to the Nose Gear Challenge

Yannick Moy
Senior Software Engineer

The Extended Nose Gear Challenge

The Original Nose Gear Challenge

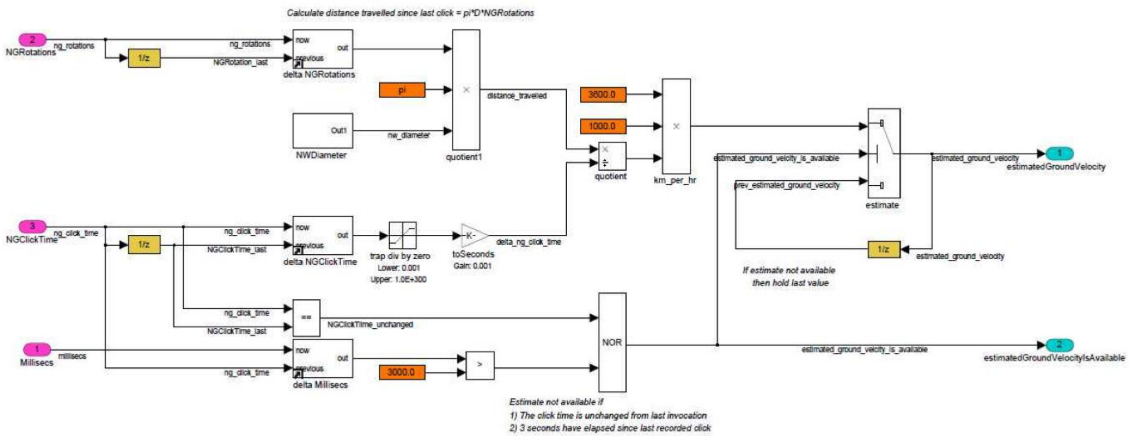


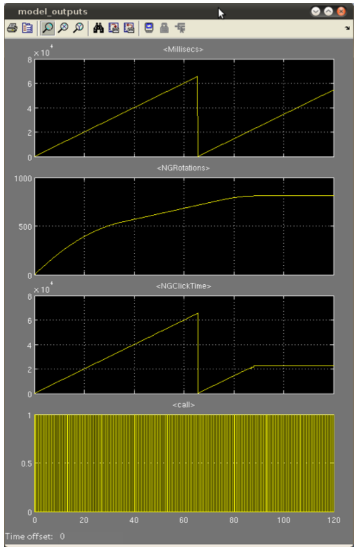
HLR 1: when available, computed velocity should be close to actual velocity

HLR 2: computed velocity should be available most of the time

Solutions from 2nd Workshop (2011)

Best solution so far presented by Colin O'Halloran: from Simulink to SPARK with CLawZ

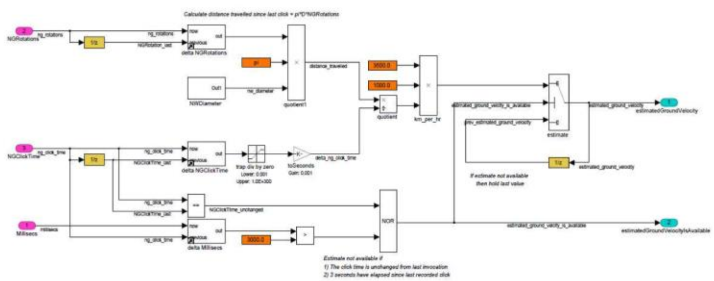


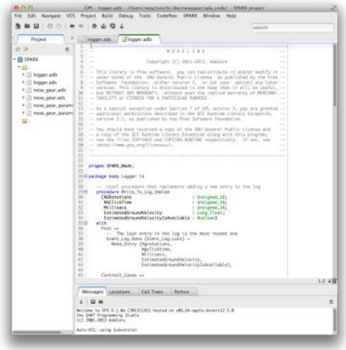


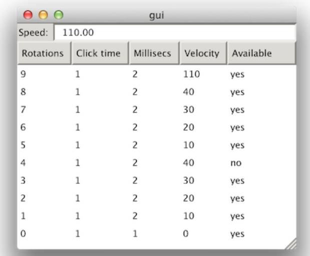
Other solutions use contract-based specification / verification with SPARK to:

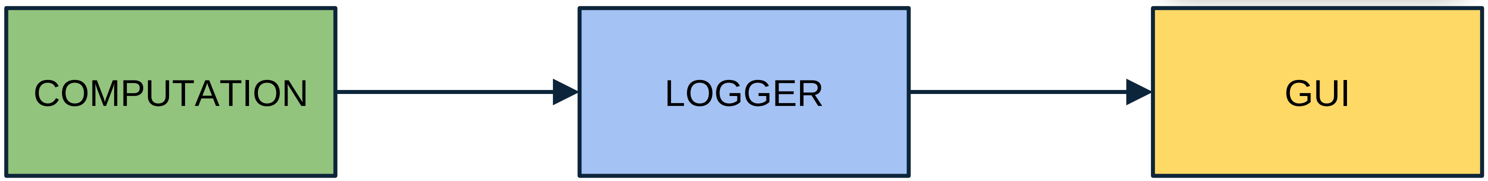
- guarantee absence of run-time errors
- prove that implementation conforms to contract

The Extended Nose Gear Challenge









HLR 1: when available, computed velocity should be close to actual velocity

HLR 2: computed velocity should be available most of the time

HLR 3: a log of all events of the latest five minutes shall be saved

HLR 4: the graphical user interface shall show

1. the estimated velocity computed
2. a warning message if the velocity is not available
3. all events collected

A Solution Focused on Integrity Preservation

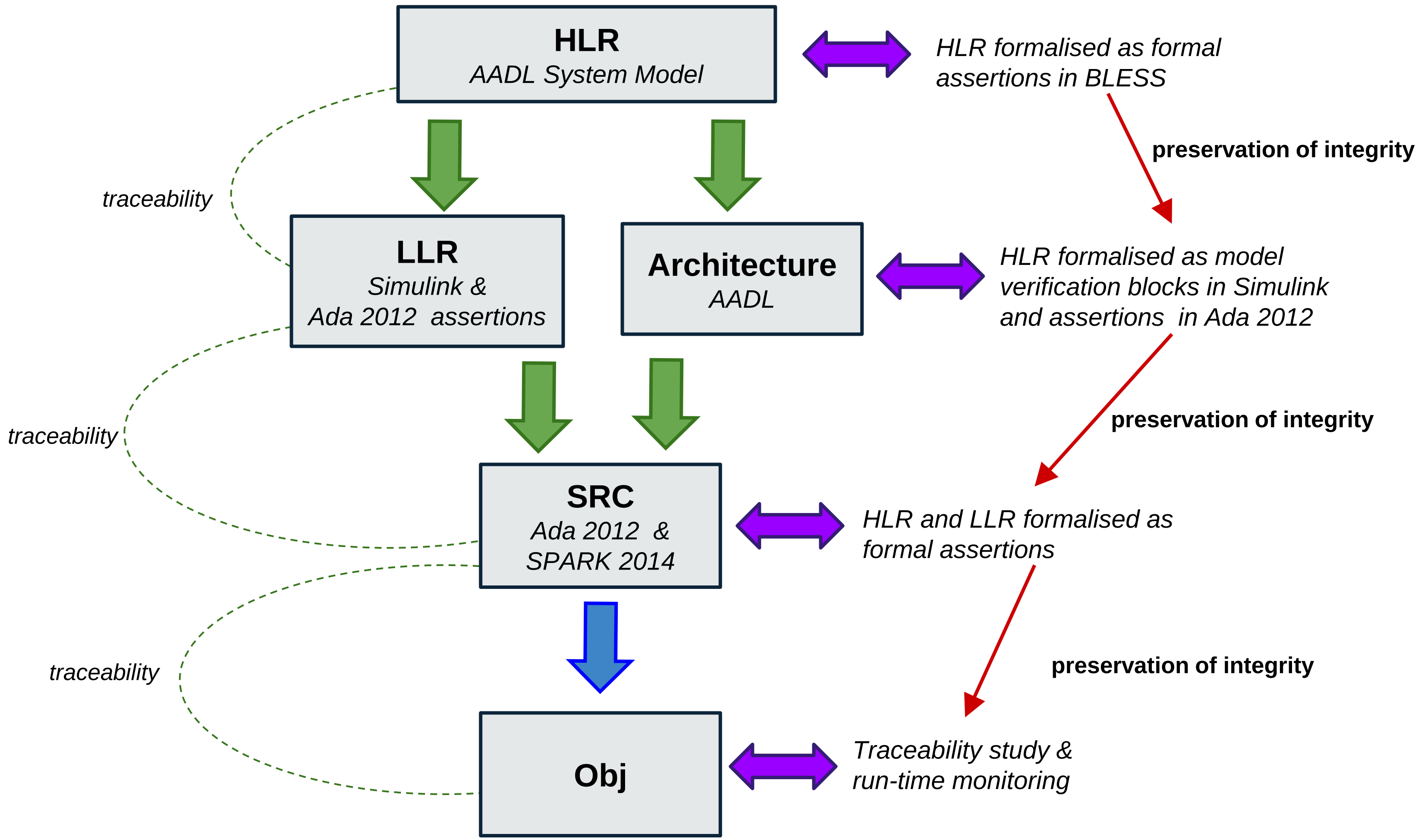
System to Software Integrity Preservation

Our main goal for the Nose Gear Challenge

6 ways to preserve integrity:

1. peer review at different levels (classical approach)
2. extensive testing at different levels and compare output (Simulink vs gen. code)
3. qualifiable automatic code generation (SCADE, GNAT Pro Simulink)
4. formalize requirement as source code contracts (Ada 2012, SPARK)
5. translate contracts across different levels (Simulink assertion to SPARK contract)
6. extract properties at different levels and compare them (CLawZ, Mathworks)

System to Software Integrity Preservation



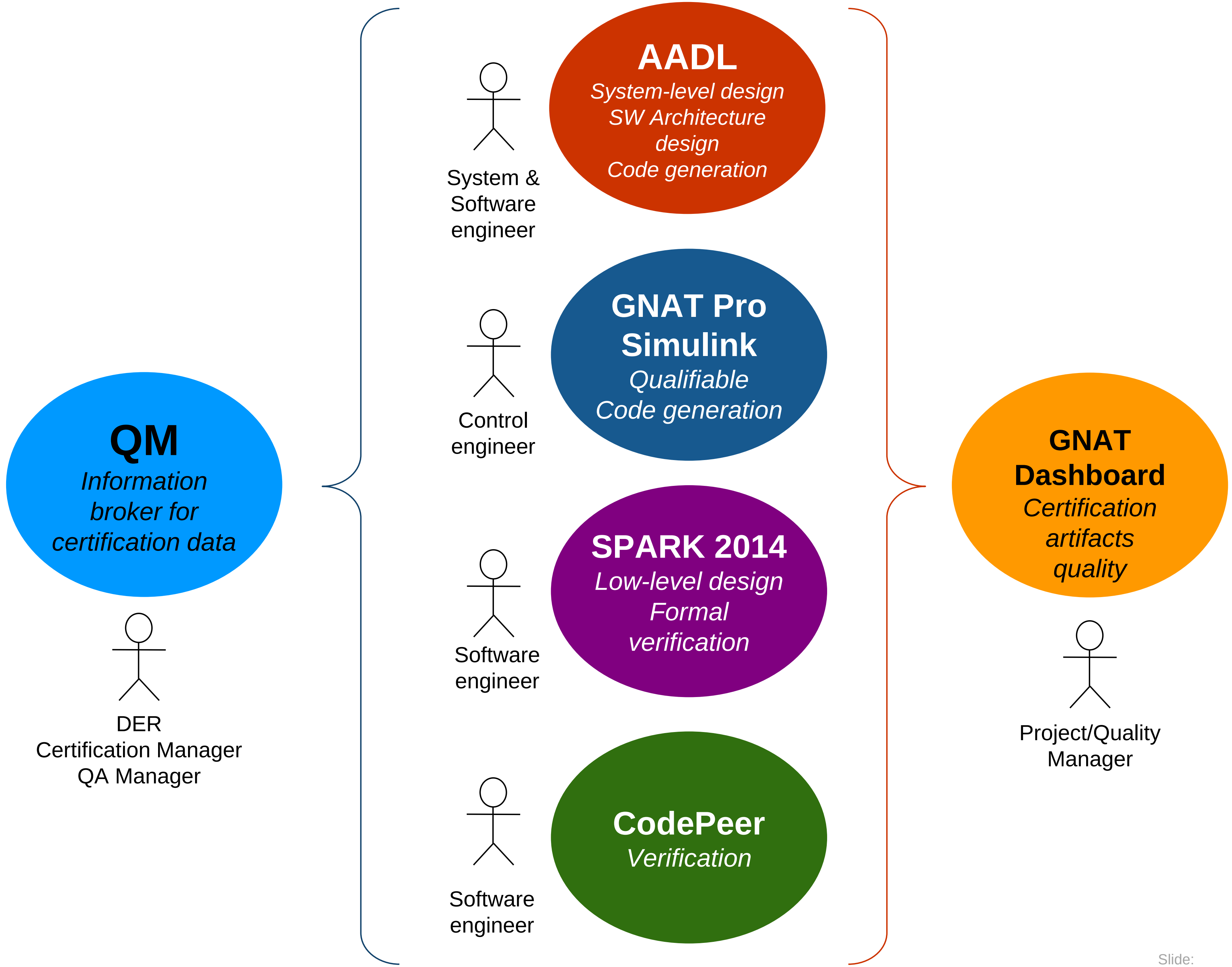
Our Choice of Languages and Tools

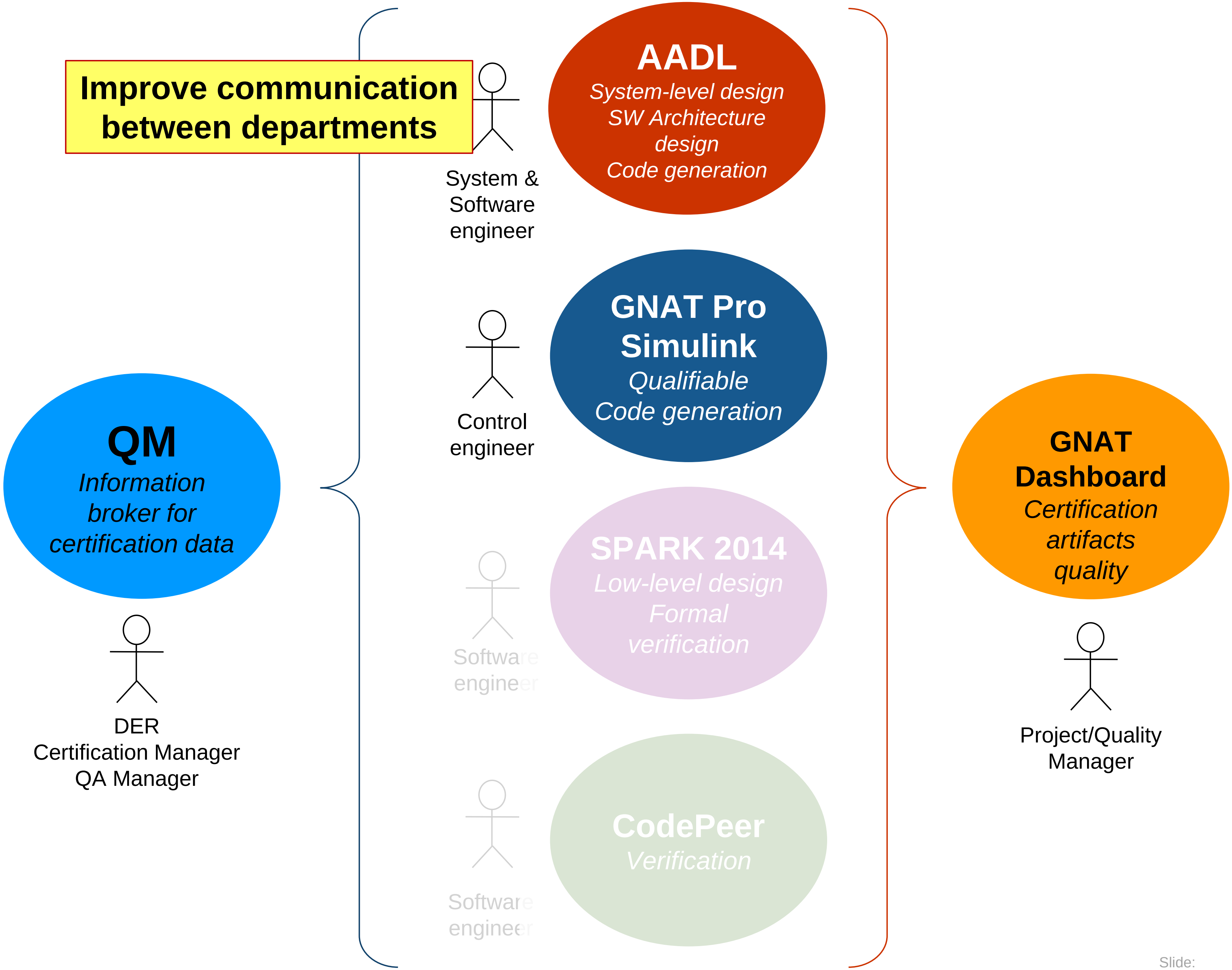
Languages:

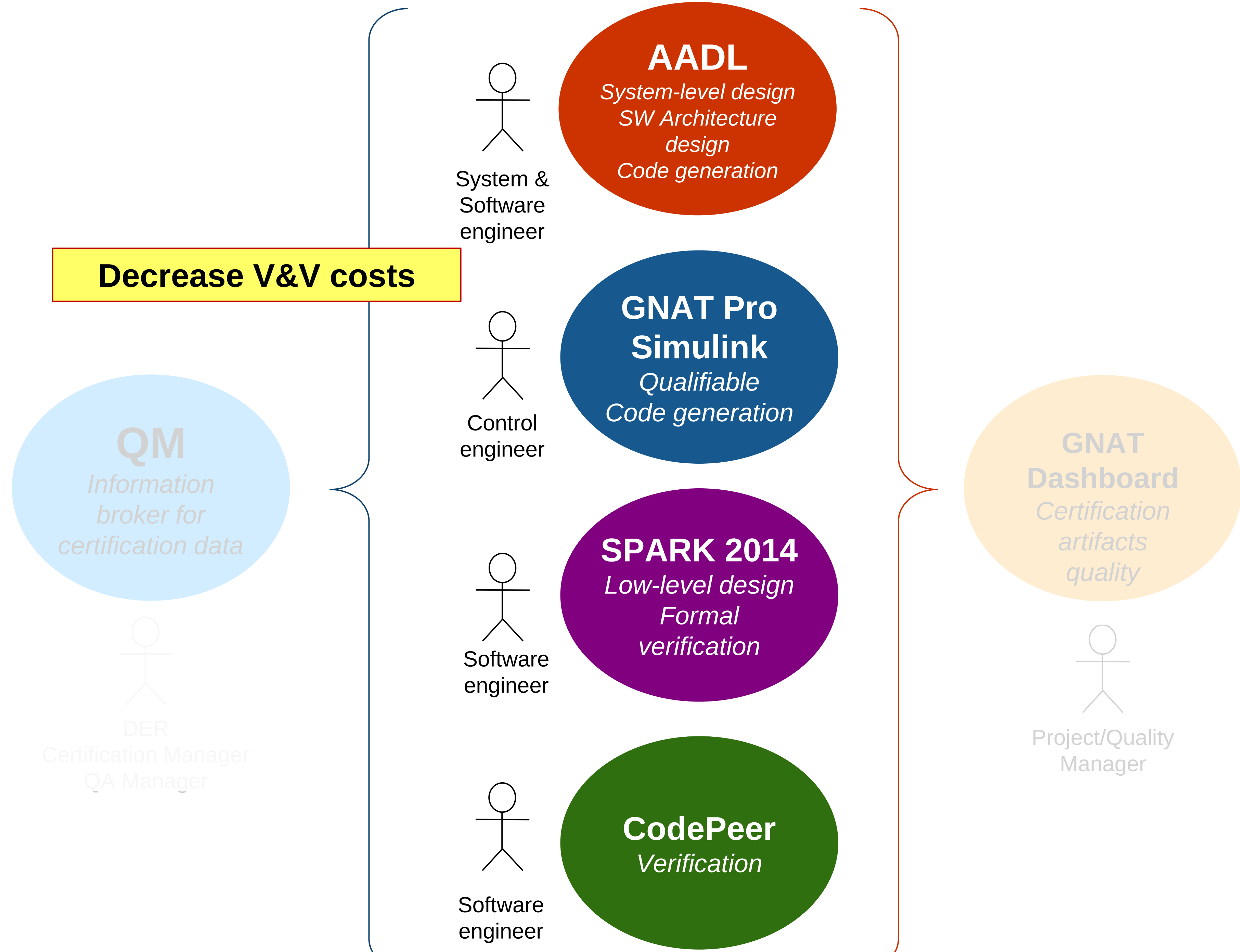
- AADL architecture description language
- Simulink modeling language
- Ada 2012 programming language (with contracts)
- SPARK 2014 subset of Ada for formal verification

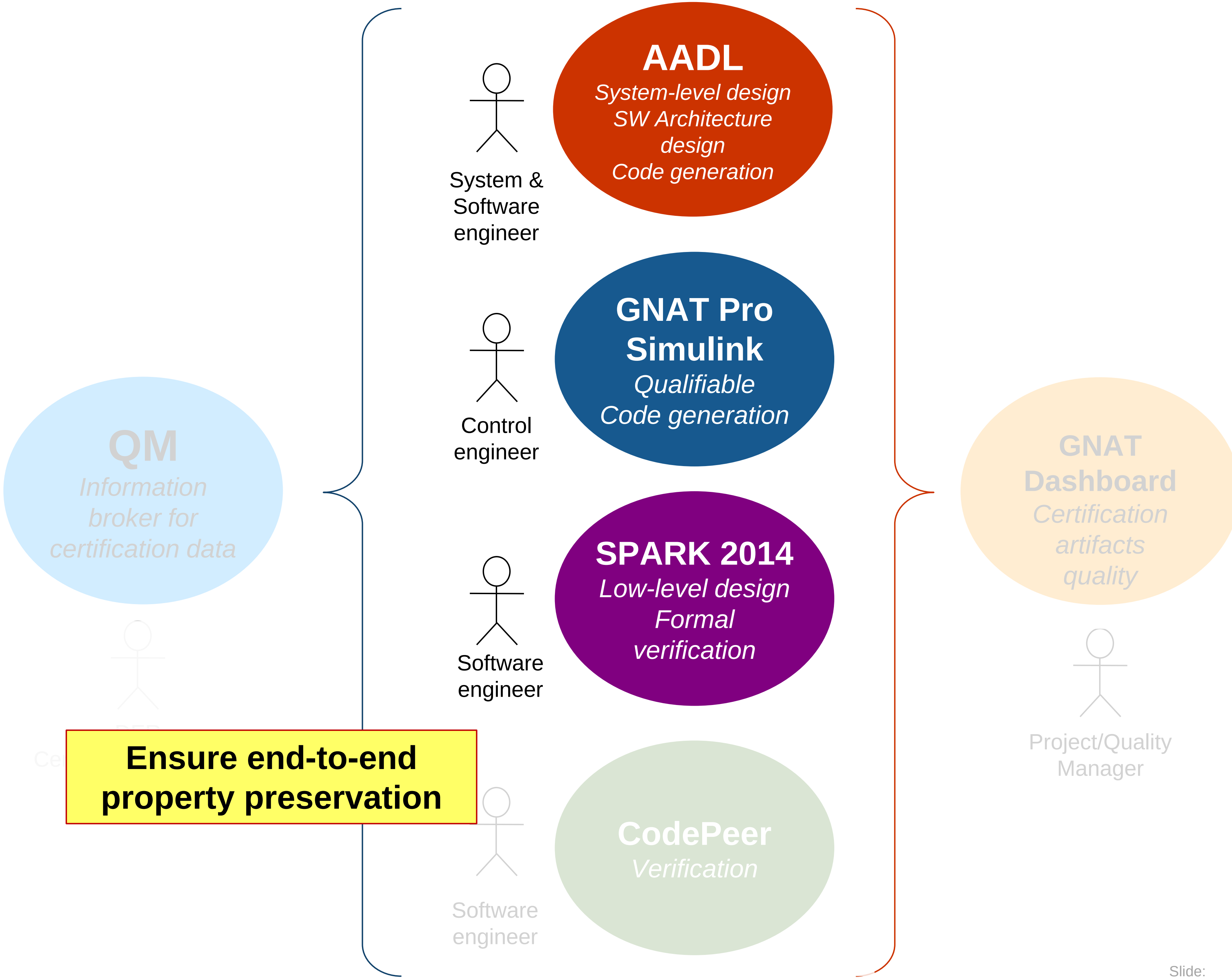
Tools:

- Ocarina code generator: AADL → Ada
- GNAT Pro for Simulink (qualifiable): Simulink → Ada
- SPARK formal verification toolset: SPARK → proofs
- CodePeer static analyzer: Ada → potential errors
- GNAT Pro: Ada → executable
- GNAT Dashboard: Ada → visualization of certification artifacts
- Qualifying Machine (QM): artifacts → agile qualification management

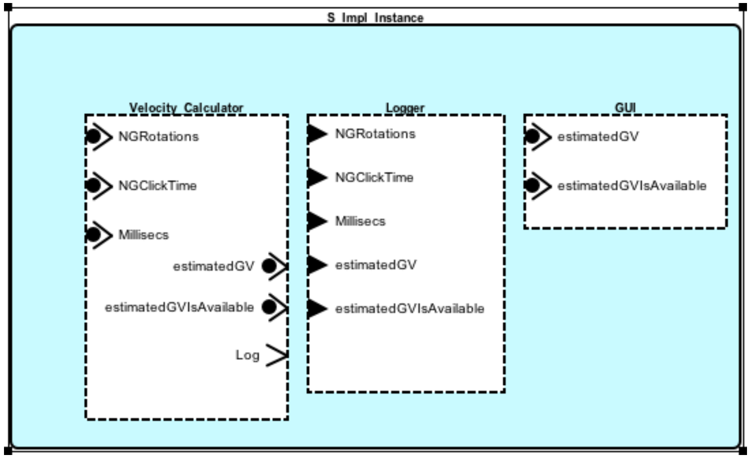

QM
Information broker for certification data
DER
Certification Manager
QA Manager
AADL
System-level design
SW Architecture design
Code generation
System & Software engineer
GNAT Pro Simulink
Qualifiable Code generation
Control engineer
SPARK 2014
Low-level design
Formal verification
Software engineer
CodePeer
Verification
Software engineer
GNAT Dashboard
Certification artifacts quality
Project/Quality Manager
Slide:


Improve communication between departments
AADL
System-level design
SW Architecture design
Code generation
System & Software engineer
GNAT Pro Simulink
Qualifiable Code generation
Control engineer
SPARK 2014
Low-level design
Formal verification
Software engineer
CodePeer
Verification
Software engineer
QM
Information broker for certification data
DER
Certification Manager
QA Manager
GNAT Dashboard
Certification artifacts quality
Project/Quality Manager


Decrease V&V costs
AADL
System-level design
SW Architecture design
Code generation
System & Software engineer
GNAT Pro Simulink
Qualifiable Code generation
Control engineer
SPARK 2014
Low-level design
Formal verification
Software engineer
CodePeer
Verification
Software engineer
QM
Information broker for certification data
DER
Certification Manager
QA Manager
GNAT Dashboard
Certification artifacts quality
Project/Quality Manager


AADL
System-level design
SW Architecture design
Code generation
System & Software engineer
GNAT Pro Simulink
Qualifiable Code generation
Control engineer
SPARK 2014
Low-level design
Formal verification
Software engineer
CodePeer
Verification
Software engineer
QM
Information broker for certification data
GNAT Dashboard
Certification artifacts quality
Project/Quality Manager
Ensure end-to-end property preservation

System-level Specification in AADL




```
abstract Velocity_Calculation
features
   NGRotations : in data port Integer;
   NGClickTime : in data port Date;
   Millisecs   : in data port Date;
   estimatedGroundVelocity : requires data access Velocity;
   estimatedGroundVelocityIsAvailable : requires data access Boolean;
properties
   Dispatch_Protocol => Periodic;
   Period => 500 Ms;
   Compute_Entrypoint => classifier (Velocity_Calculation_Spg);
   Compute_Execution_Time => 10 Ms .. 100 Ms;
```

System I/O

Real-time properties and allocation

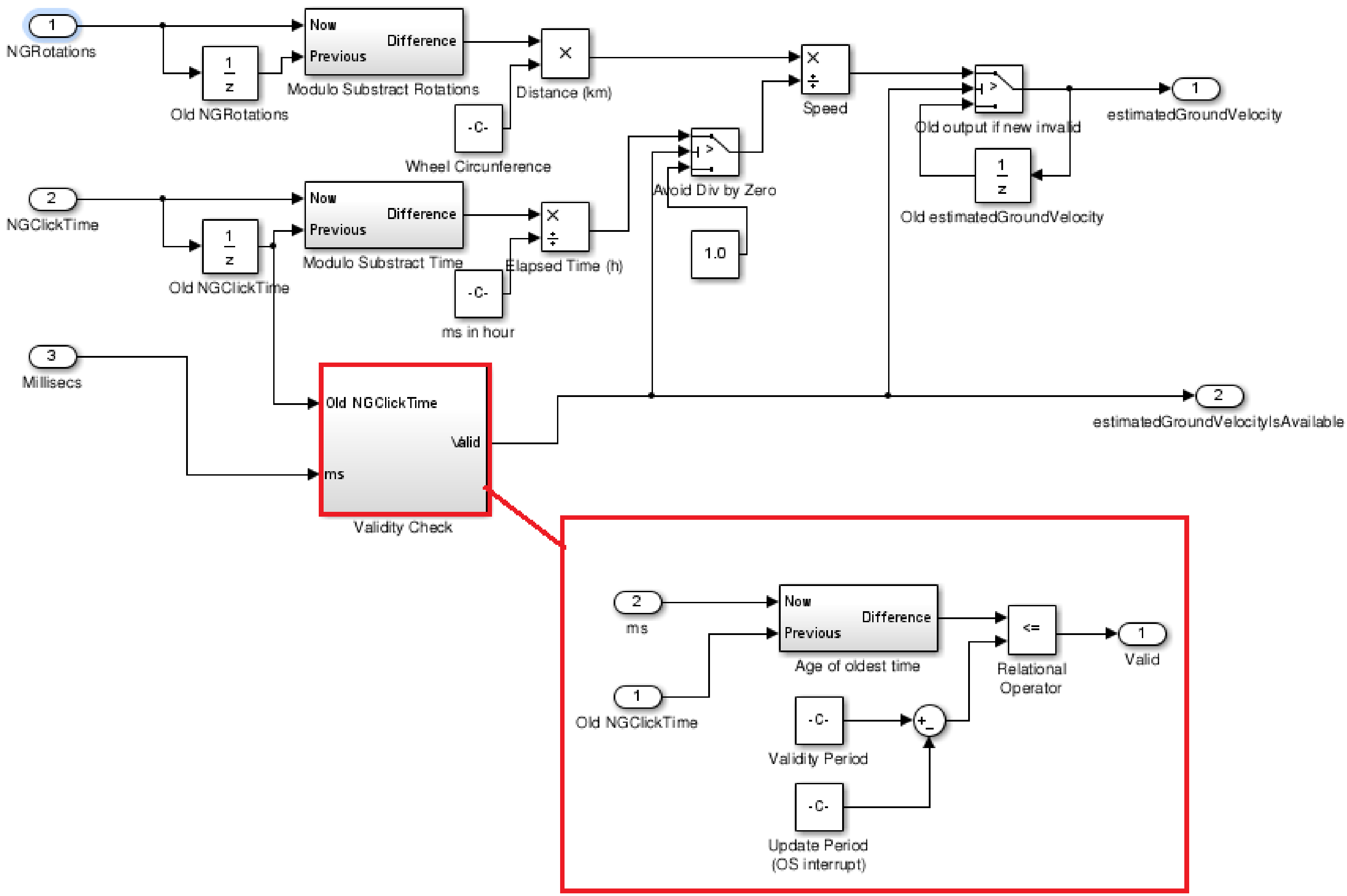
System-level Specification in AADL

```
thread Velocity_Calculation
...
assert
  <<hlr_availability: :
    (((Millisecs + NGClickTime^(-1)) - Timing_Properties::Period) <= 3000)
       iff estimatedGroundVelocityIsAvailable >>
states
  s0 : initial state;
  s1 : complete state;
transitions
   s0 -[ ]-> s1 {};
   s1 -[ on dispatch ]-> s1 {
      Velocity_Calculation_Spg(
          NGRotations, NGClickTime, Millisecs,
          estimatedGroundVelocity, estimatedGroundVelocityIsAvailable)
      << hlr_availability() >>
   };
end Velocity_Calculation;
```

HLR formalised as assertions

Formal specification of behaviour (skeleton) plus verification of assertions

Simulink Model (LLR)



Generated Code in SPARK

Only code currently generated, contract manually translated

In the future: contract generated from Simulink observer

```
procedure nose_gear_comp
  (NGRotations : Unsigned_16;
        NGClickTime : Unsigned_16;
        Millisecs : Unsigned_16;
        estimatedGroundVelocity : out Long_Float;
        estimatedGroundVelocityIsAvailable : out Boolean)
 with Post =>
        --  @llr Compute
        --  The ground velocity shall be available only if the time difference
        --  between the current calculation and the previous one is less than
        --  2500.
        (EstimatedGroundVelocityIsAvailable =
           (Millisecs + 500 - Old_NGClickTime_memory <= 3000));
```

HLR 3: a log of all events of the latest five minutes shall be saved

events scheduled at rate of one every 500 ms → 600 events in 5 mn

API of logger should give:

- function to retrieve content of the log Log_Content
- procedure to update content of the log Write_To_Log

Most natural specification cannot be expressed as contract: “Log_Content returns the set of events that have been added to the log by calls to Write_To_Log”

Use contract on Write_To_Log instead

Formal Specification and Verification in SPARK

```
procedure Write_To_Log (E : Log_Entry)
--  @llr Write_To_Log
with Contract_Cases =>
       --  The logger component shall be able to accept a new logging message.
       --  For an old empty log, the new content is the new entry alone.
       (Is_Empty =>
                 Log_Content = Singleton_Log (E),

       --  For an old full log, the new content is the old one, with the
       --  oldest entry removed, plus the new entry.
       Is_Full =>
                 Log_Content =
                 Log_Content'Old (Log_Content'Old'First + 1 .. Log_Content'Old'Last)
                 & E,

       --  For an old log neither empty not full, the old content is
       --  preserved, and the new entry added.
       others =>
                 Log_Content = Log_Content'Old & E);
```

Formal Specification and Verification in SPARK

automatic formal verification of contract

→ verification of HLR 3

+ automatic formal verification of absence of run-time errors

work in progress, current tool limitation does not allow 100% proof…

Summary of Verification Strategies for HLR 1 - 4

HLR 1: when available, computed velocity should be close to actual velocity

→ simulation in Simulink, same as done by Colin O'Halloran in 2011

HLR 2: computed velocity should be available most of the time

→ BLESS annotation in AADL → observer in Simulink → contract in SPARK

→ formally verified against implementation

HLR 3: a log of all events of the latest five minutes shall be saved

→ contract in SPARK → formally verified against implementation

HLR 4: the graphical user interface shall show …

→ tests

Agile Management of Certification Artifacts

Problem: “big-freeze” in certification

Development is frozen after start of certification, due to high cost of manual certification activities

Solution: automatic management of artifacts dependencies

Demo of the Qualifying Machine

Progress on Verification Activities

Initial Experiments

Use of static analysis (CodePeer) and formal verification (SPARK) detected errors in manually-written contracts...

and one error (!) in the code generator:

```
Sum_out_1 := Integer_32
 ((NGRotations_out_1) - (Old_NGRotations_out_1));
```

should be

```
Sum_out_1 := Integer_32 (NGRotations_out_1) -
             Integer_32 (Old_NGRotations_out_1);
```

Initial code generation strategy used many type conversions

→ Hard to analyze automatically

New code generation strategy preserves types

→ Much better automation of proof

Simulink has no concept of bounded integer types

→ Information on ranges is not passed on to generated code

Suitable assertion blocks in Simulink can give this information

→ Possible use in code generator to generate ranges in Ada code

Warnings!

- **You may feel a sense of over engineering**
  - A side effect of showing several tools applied to a simple system
  - Real systems REALLY demand the use of several tools
- **Tool maturity**
  - CodePeer is the most mature one
  - SPARK 2014 is close to be a used product
  - AADL and AADL code generation have been tested in several projects
  - GNAT Pro Simulink is being tested on industrial use cases
  - QM and GNAT Dashboard are used internally