Automatic Code Generation, an OEM perspective

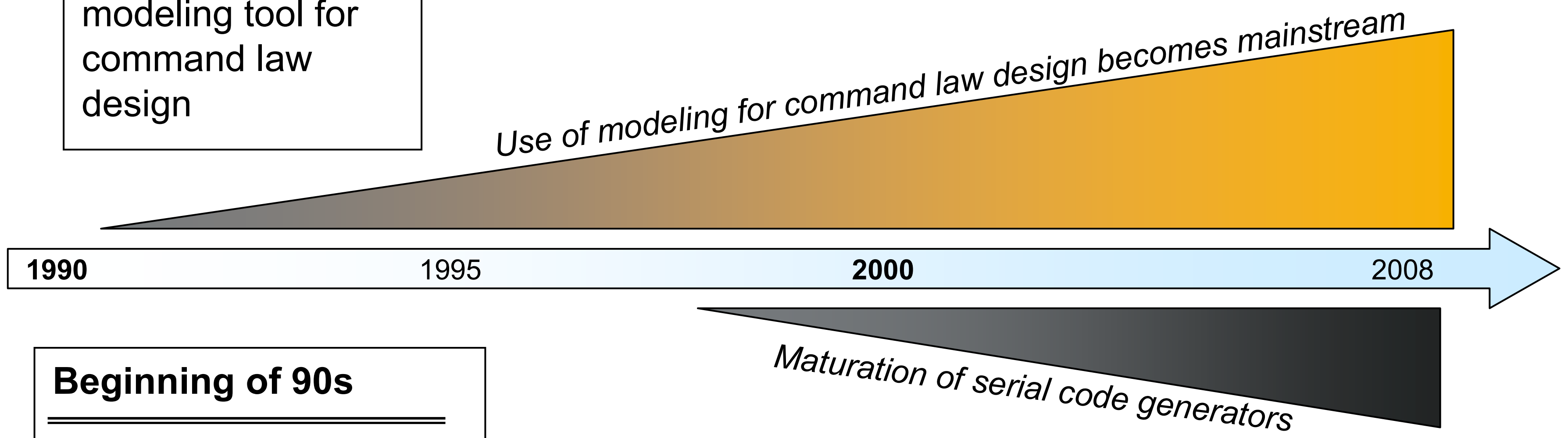
Automotive Electronics and Electrical Systems Forum

- Frédéric MONDOT / Caroline LU

Introduction to ACG

Beginning of 90s

First use of modeling tool for command law design


Use of modeling for command law design becomes mainstream
1990
1995
2000
2008
Maturation of serial code generators

Beginning of 90s

Code generators exist but are not able to produce code meeting serial constraints. They are mainly used for:
- Simulation
- Rapid prototyping

Beginning of 2000s

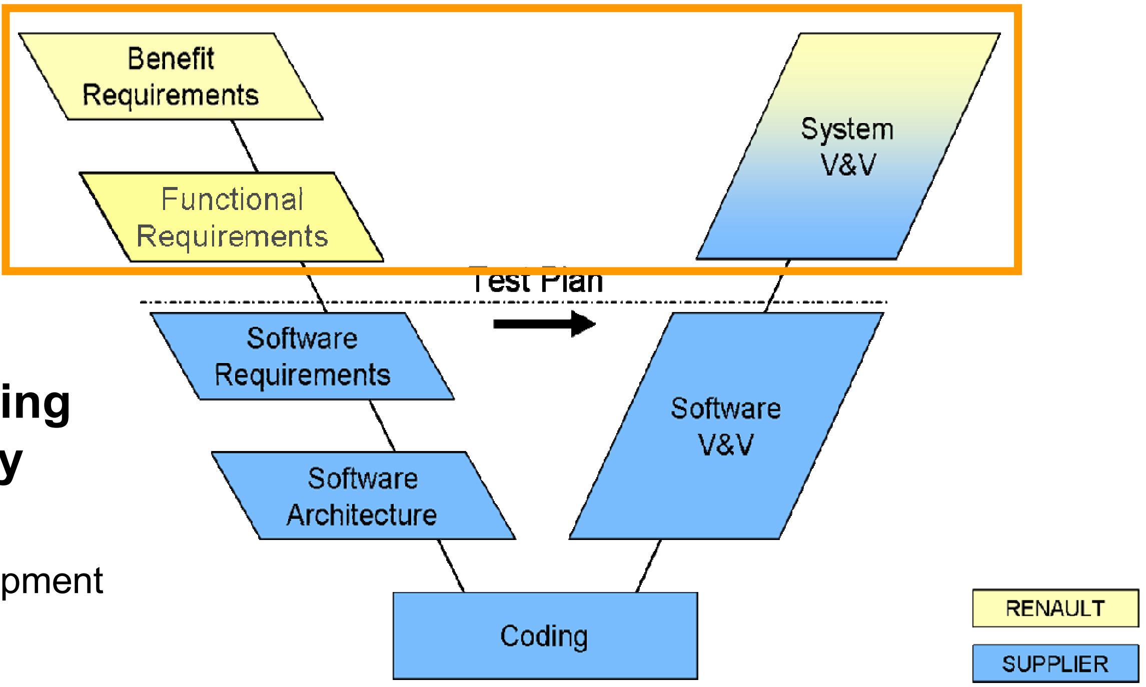
Code generators adapted to embedded serial code begin to be mature enough to be used in production

Table of Content

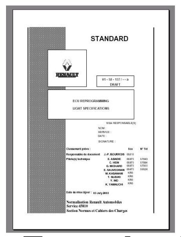
Table of Content

Renault V-cycle

- **Shared activities between supplier and Renault**
- **Functional Requirements include models**
- **Renault activities concerning software : Software Quality Assurance**
  - Management of the SW development cycle at the supplier

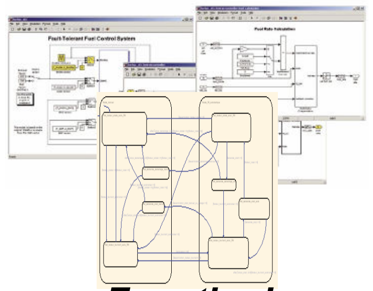


Renault Process flow to the coding phase



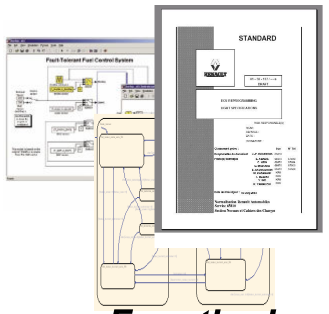
Function's Specification

- ### Specification of the function
  - ***Requirements*** *the function should satisfy (data processing, performances, interface…)*
  - ***Structured*** *in a* ***document***



Function's Design

- ### Design of the function
  - ***Decomposition*** *of the function* ***into elementary modules*** = *«* ***Functional architecture of the function*** *»*
  - *Can be formalised in a* ***functional model****, often executable*
  - *Is an entry for the software coding activity*



Function's Validation Plan

- ### Validation plan of the function
  - ***Test cases needed to verify each requirement.***
  - *Use of the validated executable functional model as a behavioral reference* **(MIL/SIL)**

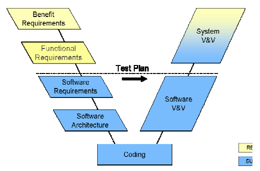
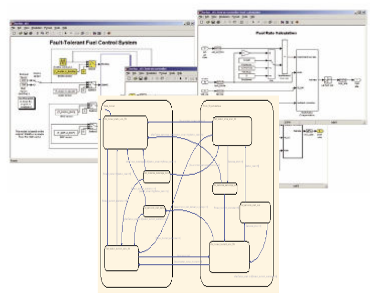


Table of Content

Manual vs Automatic Code Generation

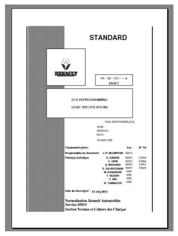
Input data

Functional model



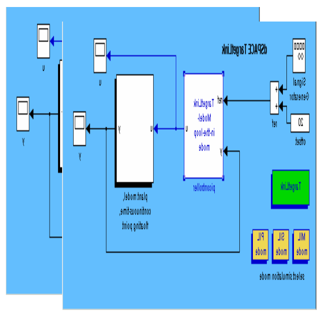
AND

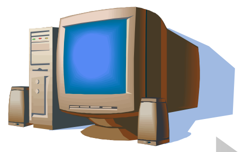
Function specification



CONVERSION

Implementation model

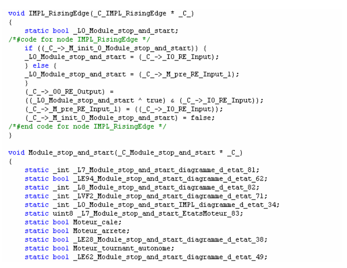




GENERATION

Code generator

C source code

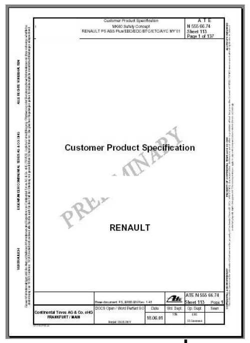


```c
void IMPL_RisingEdge(_C_IMPL_RisingEdge * _C_)
{
    static bool _L0_Module_stop_and_start;
/*#code for node IMPL_RisingEdge */
    if ((_C_->_M_init_0_Module_stop_and_start)) {
    _L0_Module_stop_and_start = (_C_->_I0_RE_Input);
    } else {
    _L0_Module_stop_and_start = (_C_->_M_pre_RE_Input_1);
    }
    (_C_->_O0_RE_Output) =
    ((_L0_Module_stop_and_start ^ true) & (_C_->_I0_RE_Input));
    (_C_->_M_pre_RE_Input_1) = ((_C_->_I0_RE_Input));
    (_C_->_M_init_0_Module_stop_and_start) = false;
/*#end code for node IMPL_RisingEdge */
}

void Module_stop_and_start(_C_Module_stop_and_start * _C_)
{
    static _int _L7_Module_stop_and_start_diagramme_d_etat_81;
    static bool _LE94_Module_stop_and_start_diagramme_d_etat_62;
    static _int _L8_Module_stop_and_start_diagramme_d_etat_82;
    static _int _LVF2_Module_stop_and_start_diagramme_d_etat_71;
    static _int _L0_Module_stop_and_start_IMPL_diagramme_d_etat_34;
    static uint8 _L7_Module_stop_and_start_EtatsMoteur_83;
    static bool Moteur_cale;
    static bool Moteur_arrete;
    static bool _LE28_Module_stop_and_start_diagramme_d_etat_38;
    static bool Moteur_tournant_autonome;
    static bool _LE62_Module_stop_and_start_diagramme_d_etat_49;
```

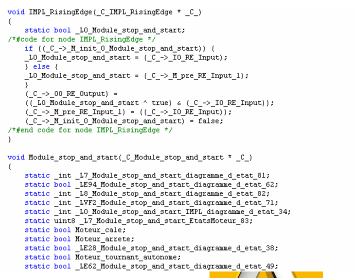
SW Designer

SW design



Developer

C source code

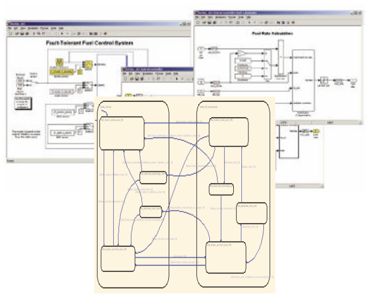


```c
void IMPL_RisingEdge(_C_IMPL_RisingEdge * _C_)
{
    static bool _L0_Module_stop_and_start;
/*#code for node IMPL_RisingEdge */
    if ((_C_->_M_init_0_Module_stop_and_start)) {
    _L0_Module_stop_and_start = (_C_->_I0_RE_Input);
    } else {
    _L0_Module_stop_and_start = (_C_->_M_pre_RE_Input_1);
    }
    (_C_->_O0_RE_Output) =
    ((_L0_Module_stop_and_start ^ true) & (_C_->_I0_RE_Input));
    (_C_->_M_pre_RE_Input_1) = ((_C_->_I0_RE_Input));
    (_C_->_M_init_0_Module_stop_and_start) = false;
/*#end code for node IMPL_RisingEdge */
}

void Module_stop_and_start(_C_Module_stop_and_start * _C_)
{
    static _int _L7_Module_stop_and_start_diagramme_d_etat_81;
    static bool _LE94_Module_stop_and_start_diagramme_d_etat_62;
    static _int _L8_Module_stop_and_start_diagramme_d_etat_82;
    static _int _LVF2_Module_stop_and_start_diagramme_d_etat_71;
    static _int _L0_Module_stop_and_start_IMPL_diagramme_d_etat_34;
    static uint8 _L7_Module_stop_and_start_EtatsMoteur_83;
    static bool Moteur_cale;
    static bool Moteur_arrete;
    static bool _LE28_Module_stop_and_start_diagramme_d_etat_38;
    static bool Moteur_tournant_autonome;
    static bool _LE62_Module_stop_and_start_diagramme_d_etat_49;
```

ACG Tools Examples

Functional model

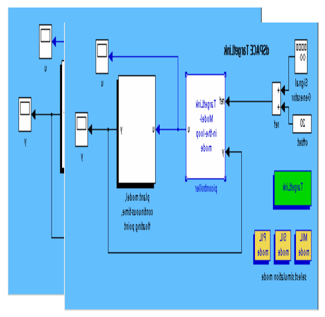


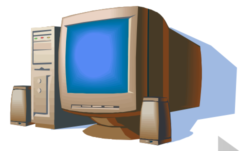
<u>Modeling tools</u>

Simulink/Stateflow™
Statemate™
SCADE™
ASCET™

CONVERSION

Implementation model





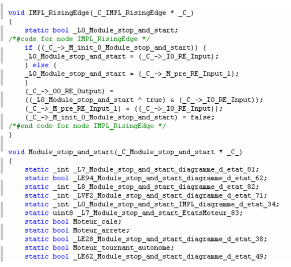
GENERATION

Code generator

<u>Generators</u>

Targetlink™ RTW Embedded Code™
STM Developer™
SCADE™
ASCET™

C source code



```
void IMPL_RisingEdge(_C_IMPL_RisingEdge * _C_)
{
    static bool _L0_Module_stop_and_start;
/*#code for node IMPL_RisingEdge */
    if ((_C_->_M_init_0_Module_stop_and_start)) {
    _L0_Module_stop_and_start = (_C_->_I0_RE_Input);
    } else {
    _L0_Module_stop_and_start = (_C_->_M_pre_RE_Input_1);
    }
    (_C_->_O0_RE_Output) =
    ((_L0_Module_stop_and_start ^ true) & (_C_->_I0_RE_Input));
    (_C_->_M_pre_RE_Input_1) = ((_C_->_I0_RE_Input));
    (_C_->_M_init_0_Module_stop_and_start) = false;
/*#end code for node IMPL_RisingEdge */
}

void Module_stop_and_start(_C_Module_stop_and_start * _C_)
{
    static _int _L7_Module_stop_and_start_diagramme_d_etat_81;
    static bool _LE94_Module_stop_and_start_diagramme_d_etat_62;
    static _int _L8_Module_stop_and_start_diagramme_d_etat_82;
    static _int _LVF2_Module_stop_and_start_diagramme_d_etat_71;
    static _int _L0_Module_stop_and_start_IMPL_diagramme_d_etat_34;
    static uint8 _L7_Module_stop_and_start_EtatsMoteur_83;
    static bool Moteur_cale;
    static bool Moteur_arrete;
    static bool _LE28_Module_stop_and_start_diagramme_d_etat_38;
    static bool Moteur_tournant_autonome;
    static bool _LE62_Module_stop_and_start_diagramme_d_etat_49;
```

Table of Content

New activities introduced by ACG

Adaptation to generator's constraints

Optimization

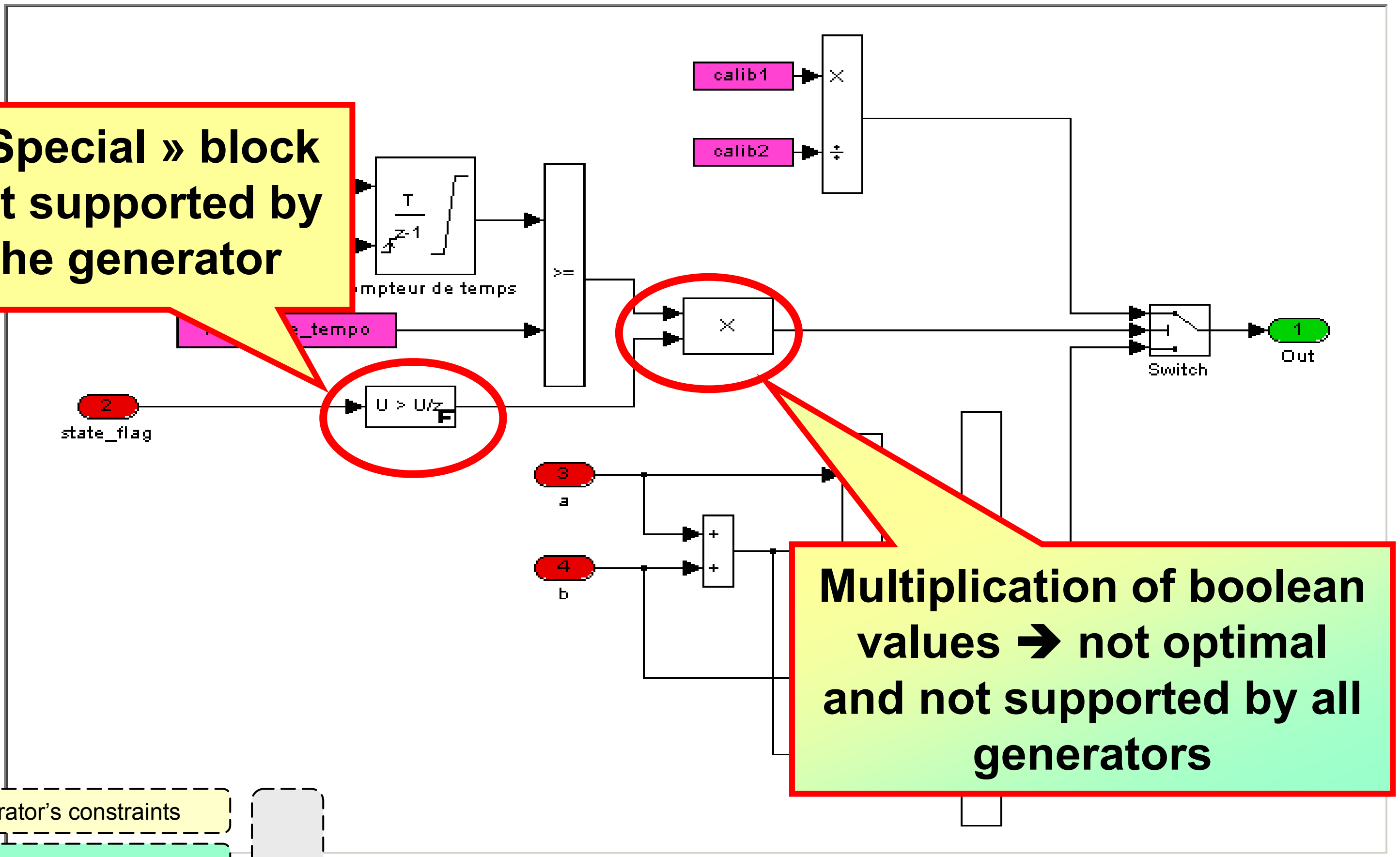
Robustification

Scaling

Software integration

V&V

Example – Adaptation to generator's constraints 1/2



« Special » block not supported by the generator
Multiplication of boolean values ➔ not optimal and not supported by all generators
calib1
calib2
T
z-1
mpteur de temps
>=
_tempo
U > U/z
2
state_flag
3
a
4
b
1
Out
Switch

Adaptation to generator's constraints

Optimization

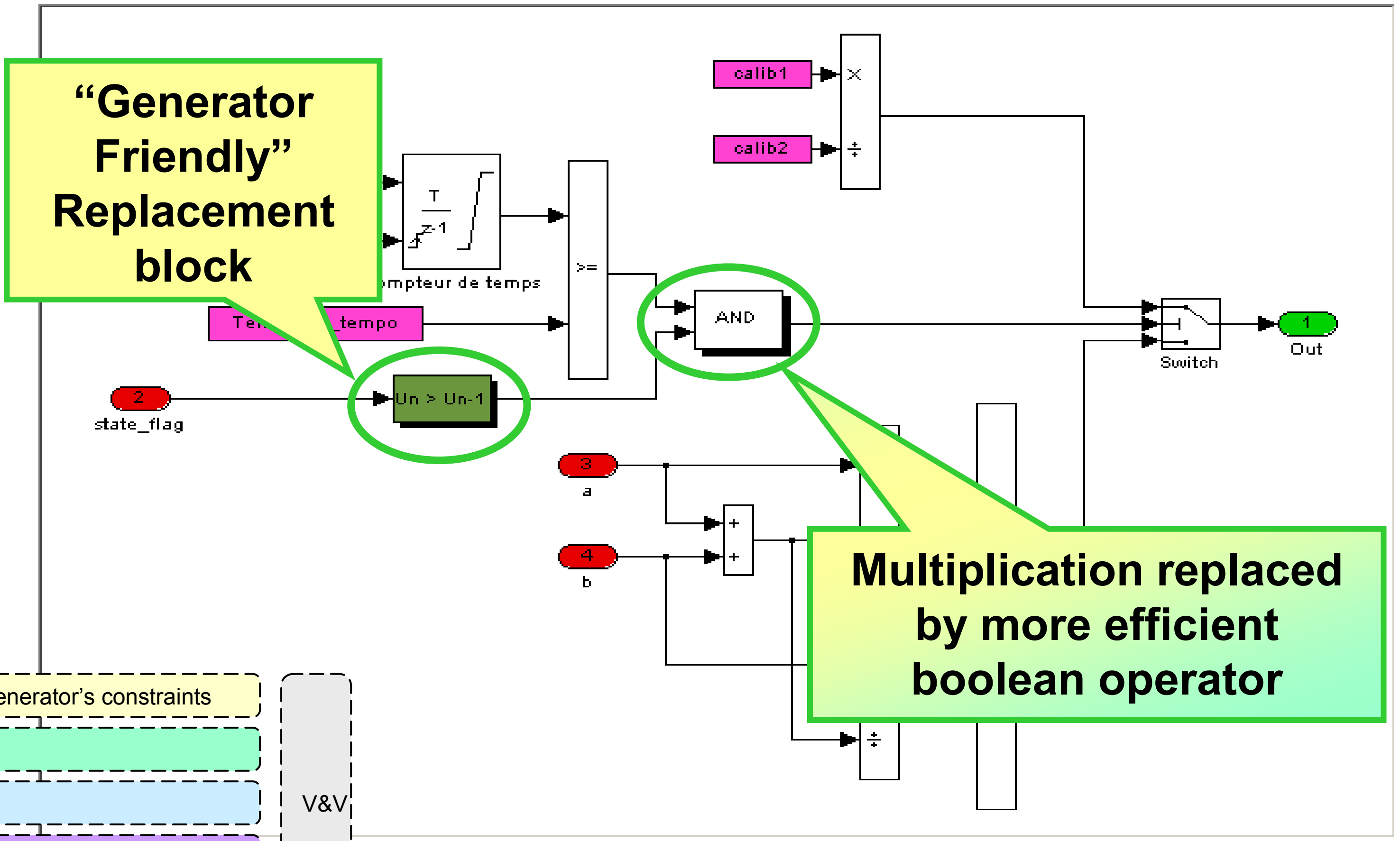
Robustification

Scaling

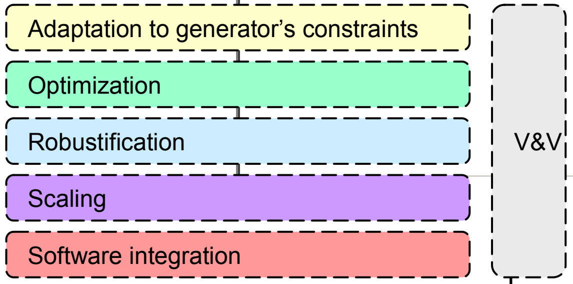
Software integration

V&V

Example – Adaptation to generator’s constraints 2/2



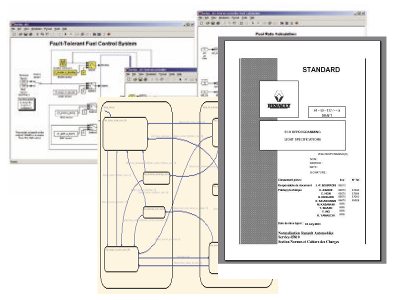
“Generator Friendly” Replacement block
calib1
calib2
T / z-1
mpteur de temps
>=
Te _tempo
AND
Switch
1
Out
2
state_flag
Un > Un-1
3
a
4
b
Multiplication replaced by more efficient boolean operator



Current business model and problems

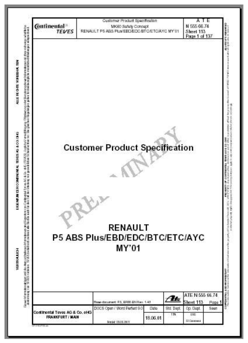
OEM → **Supplier**

Functional model
And specification



SW Designer

SW design

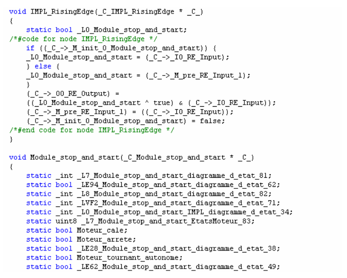

Customer Product Specification

PRELIMINARY

RENAULT
P5 ABS Plus/EBD/EDC/BTC/ETC/AYC
MY'01

Developer

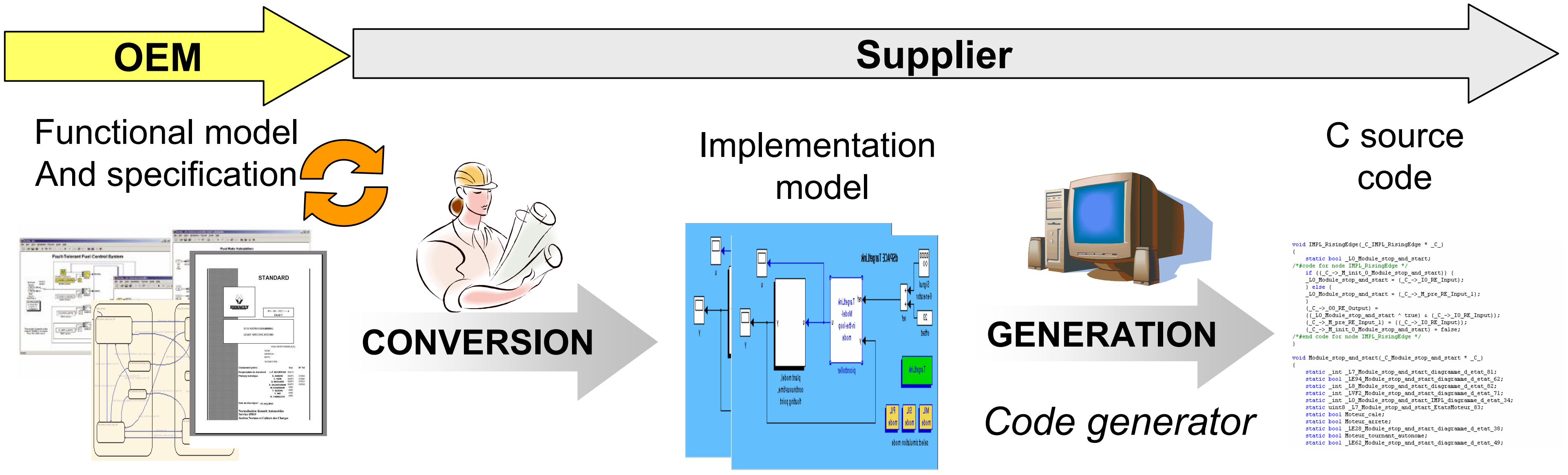
C source
code



```c
void IMPL_RisingEdge(_C_IMPL_RisingEdge * _C_)
{
    static bool _L0_Module_stop_and_start;
/*#code for node IMPL_RisingEdge */
    if ((_C_->_M_init_0_Module_stop_and_start)) {
    _L0_Module_stop_and_start = (_C_->_I0_RE_Input);
    } else {
    _L0_Module_stop_and_start = (_C_->_M_pre_RE_Input_1);
    }
    (_C_->_O0_RE_Output) =
    ((_L0_Module_stop_and_start ^ true) & (_C_->_I0_RE_Input));
    (_C_->_M_pre_RE_Input_1) = ((_C_->_I0_RE_Input));
    (_C_->_M_init_0_Module_stop_and_start) = false;
/*#end code for node IMPL_RisingEdge */
}

void Module_stop_and_start(_C_Module_stop_and_start * _C_)
{
    static _int _L7_Module_stop_and_start_diagramme_d_etat_81;
    static bool _LE94_Module_stop_and_start_diagramme_d_etat_62;
    static _int _L8_Module_stop_and_start_diagramme_d_etat_82;
    static _int _LVF2_Module_stop_and_start_diagramme_d_etat_71;
    static _int _L0_Module_stop_and_start_IMPL_diagramme_d_etat_34;
    static uint8 _L7_Module_stop_and_start_EtatsMoteur_83;
    static bool Moteur_cale;
    static bool Moteur_arrete;
    static bool _LE28_Module_stop_and_start_diagramme_d_etat_38;
    static bool Moteur_tournant_autonome;
    static bool _LE62_Module_stop_and_start_diagramme_d_etat_49;
```

Current business model and problems



```
void IMPL_RisingEdge(_C_IMPL_RisingEdge * _C_)
{
    static bool _L0_Module_stop_and_start;
/*#code for node IMPL_RisingEdge */
    if ((_C_->_M_init_0_Module_stop_and_start)) {
    _L0_Module_stop_and_start = (_C_->_I0_RE_Input);
    } else {
    _L0_Module_stop_and_start = (_C_->_M_pre_RE_Input_1);
    }
    (_C_->_O0_RE_Output) =
    ((_L0_Module_stop_and_start ^ true) & (_C_->_I0_RE_Input));
    (_C_->_M_pre_RE_Input_1) = ((_C_->_I0_RE_Input));
    (_C_->_M_init_0_Module_stop_and_start) = false;
/*#end code for node IMPL_RisingEdge */
}

void Module_stop_and_start(_C_Module_stop_and_start * _C_)
{
    static _int _L7_Module_stop_and_start_diagramme_d_etat_81;
    static bool _LE94_Module_stop_and_start_diagramme_d_etat_62;
    static _int _L8_Module_stop_and_start_diagramme_d_etat_82;
    static _int _LVF2_Module_stop_and_start_diagramme_d_etat_71;
    static _int _L0_Module_stop_and_start_IMPL_diagramme_d_etat_34;
    static uint8 _L7_Module_stop_and_start_EtatsMoteur_83;
    static bool Moteur_cale;
    static bool Moteur_arrete;
    static bool _LE28_Module_stop_and_start_diagramme_d_etat_38;
    static bool Moteur_tournant_autonome;
    static bool _LE62_Module_stop_and_start_diagramme_d_etat_49;
```

Adaptation of Models to ACG constraints

➔Request for evolution of the functional model to avoid the risk of an important rework during conversion phase.

Exhaustive textual specification with the model

➔ Without textual specification the model difficult to understand by the supplier, the supplier may have to reverse engineer the model to get the requirements (needed for test coverage)

Situation 1

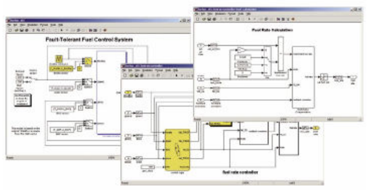
OEM

Supplier

- Generator Constraints
- Optimization
- Robustification
- Scaling
- Software Integration

Model Modifications potentially « heavy »

Functional Model
X



Generator constraints

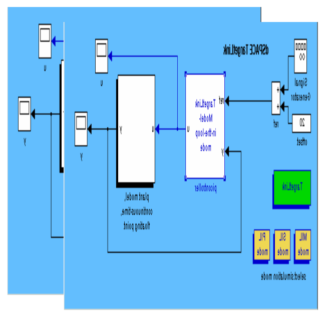
Robustification

Scaling

Optimization

Integration

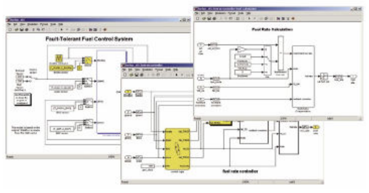
Implementation Model



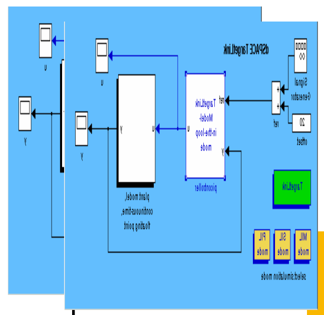
Modif

Activities potentially long : models could be very different

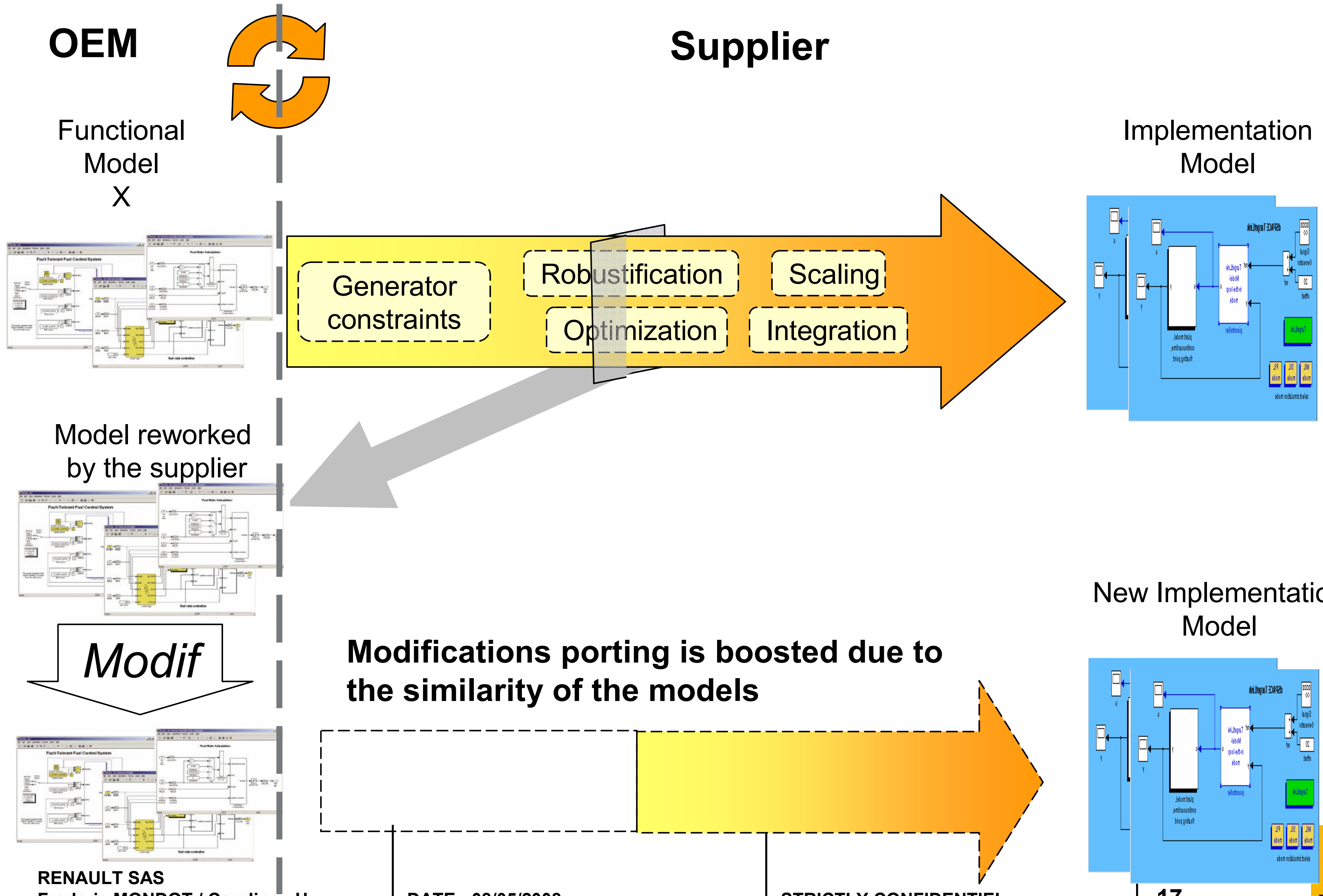
New Implementation Model



Impact Analysis
and modification of the implementation model

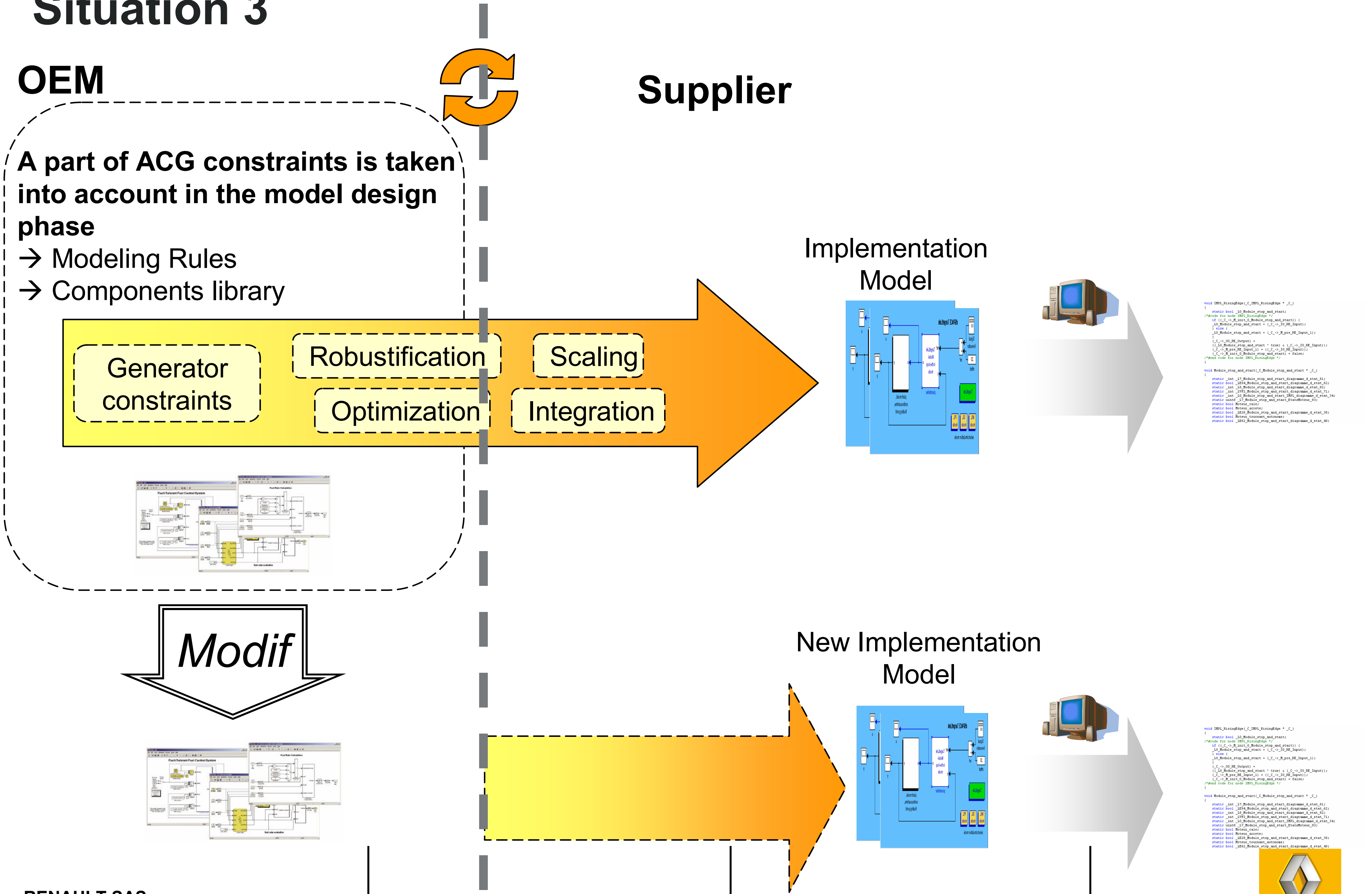


Situation 2



Modifications porting is boosted due to the similarity of the models

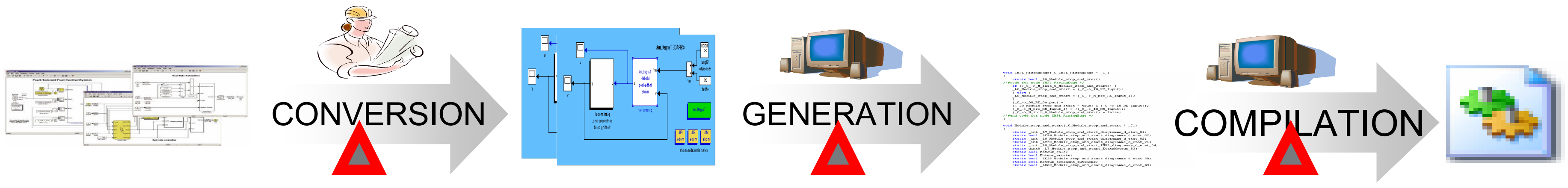
Situation 3



A part of ACG constraints is taken into account in the model design phase
→ Modeling Rules
→ Components library

Table of Content

Quality



ACG greatly reduces the number of interpretation and coding errors
ONLY IF

The quality of conversion phase is under control

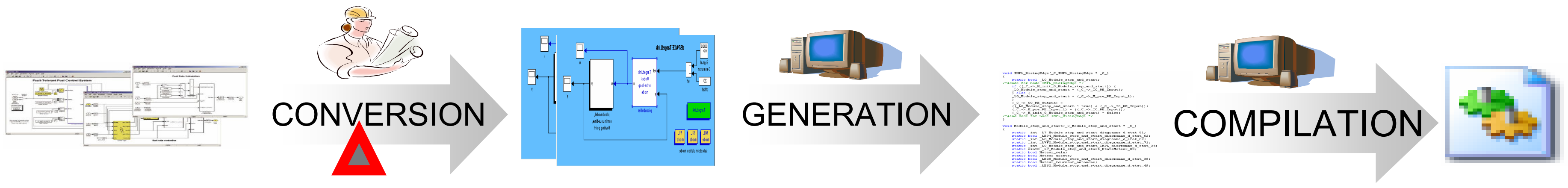
➔ Implementation models need to be validated

➔ Functional models need to be closer to implementation ones in order to reduce the risk

Deviations due to generator or compiler are under control

➔Unit testing on the compiled code is mandatory

Productivity

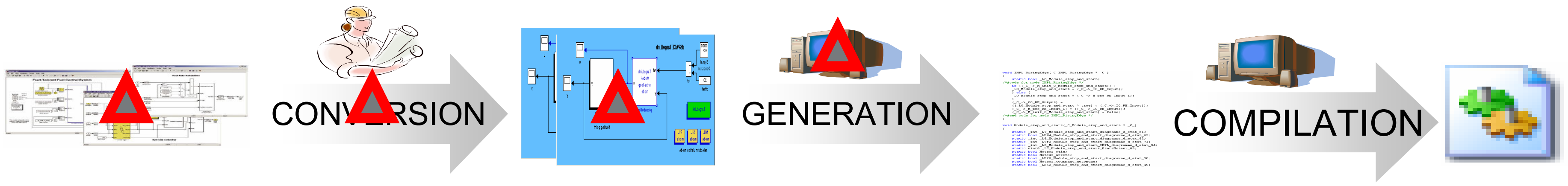


ACG can bring up to 20% gain on productivity
ONLY IF

The conversion workload is under control

➔ Taking some of the ACG constraints at the level of functional modeling can greatly reduce the conversion effort at a very low cost

Code performance



ACG has a relatively small (0 to 30%) perf overhead over manual coding
ONLY IF

<u>The model is "optimized"</u>
➔ As for compilation, a non optimized model will produce non optimized code. At the latest, the model needs to be optimized during conversion.

<u>The generator produces optimized code</u>
➔The performance of generated code highly depends on the generator's
- configuration
- optimization mechanisms

Conclusion

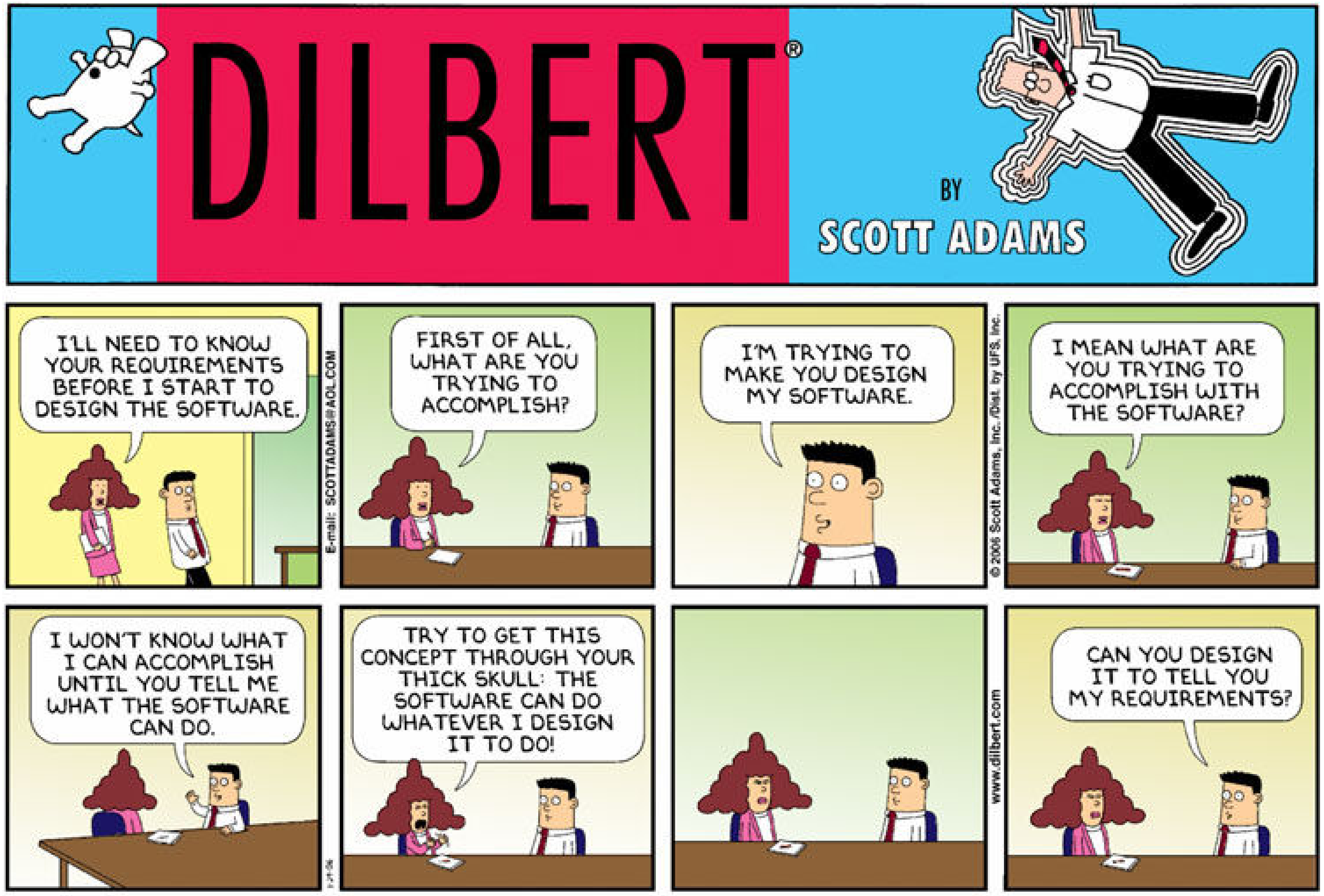
- A trend : More and more suppliers wish to use and will use Automatic Code Generation (ACG)

- ACG is not a “push button” process only

  - A lot of work is necessary before and after the production of the code
  - V&V activities are always present
  - New business model (new development flow)

- ACG allow to reduce systematic coding errors

Questions / Answers


DILBERT®
BY SCOTT ADAMS
I'LL NEED TO KNOW YOUR REQUIREMENTS BEFORE I START TO DESIGN THE SOFTWARE.
E-mail: SCOTTADAMS@AOL.COM
FIRST OF ALL, WHAT ARE YOU TRYING TO ACCOMPLISH?
I'M TRYING TO MAKE YOU DESIGN MY SOFTWARE.
© 2006 Scott Adams, Inc./Dist. by UFS, Inc.
I MEAN WHAT ARE YOU TRYING TO ACCOMPLISH WITH THE SOFTWARE?
I WON'T KNOW WHAT I CAN ACCOMPLISH UNTIL YOU TELL ME WHAT THE SOFTWARE CAN DO.
TRY TO GET THIS CONCEPT THROUGH YOUR THICK SKULL: THE SOFTWARE CAN DO WHATEVER I DESIGN IT TO DO!
www.dilbert.com
CAN YOU DESIGN IT TO TELL YOU MY REQUIREMENTS?