A Design-by-Contract approach to distributed embedded software development

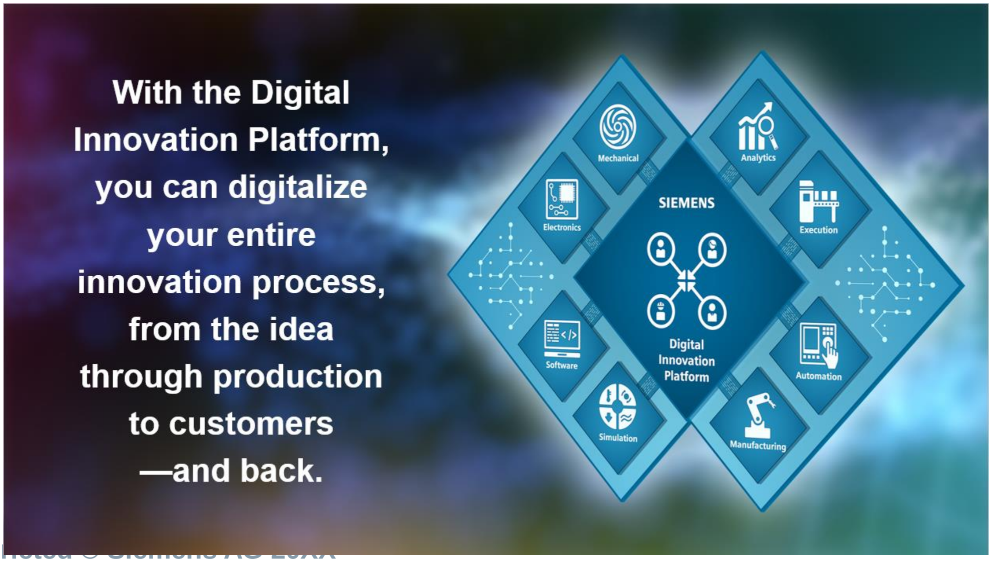
Content

- **Introduction**
- Contracts and assume guarantee analysis.
- Keeping the traces
- Conclusion

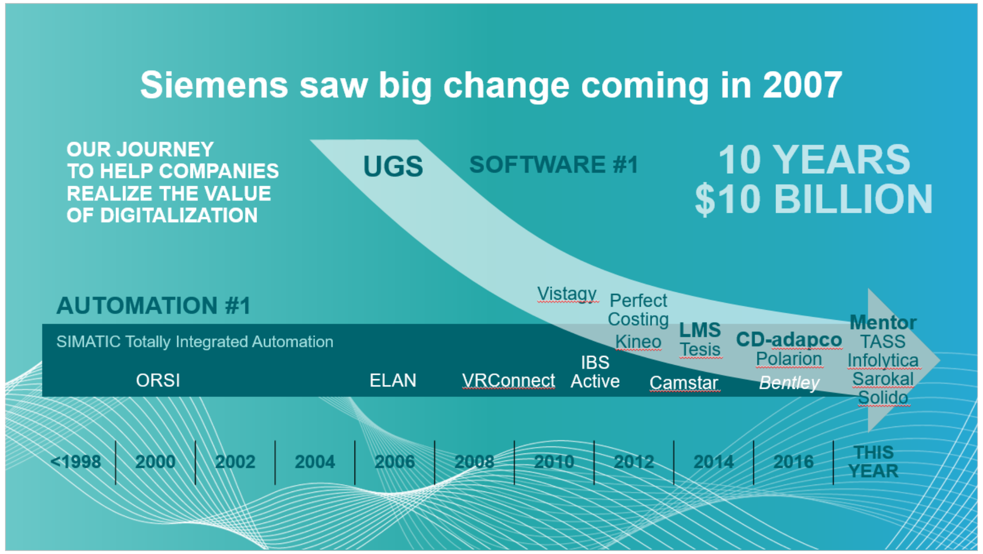
Siemens

SIEMENS
Ingenuity for life

170 years
of Siemens



With the Digital Innovation Platform, you can digitalize your entire innovation process, from the idea through production to customers —and back.



Siemens saw big change coming in 2007

OUR JOURNEY TO HELP COMPANIES REALIZE THE VALUE OF DIGITALIZATION

10 YEARS
$10 BILLION

We are truly driving on top of software

Recent trends in industry, have led to an exponential increase in software size and complexity.

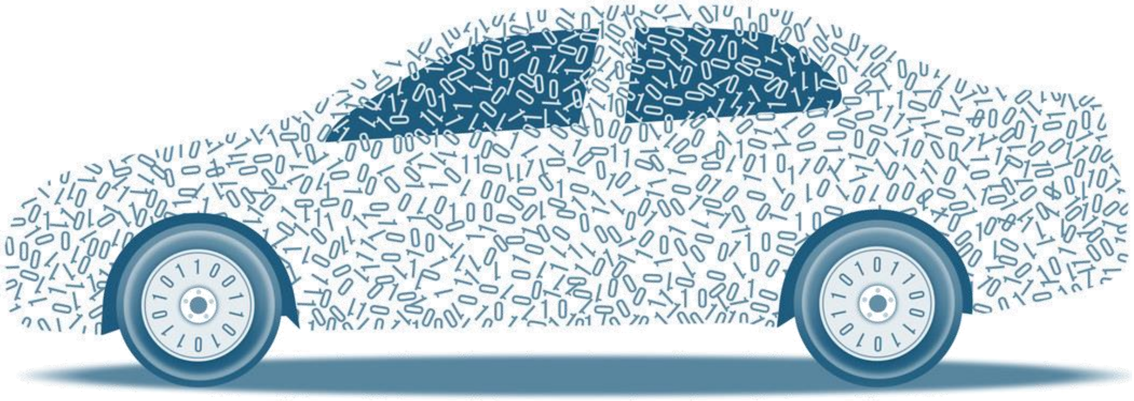
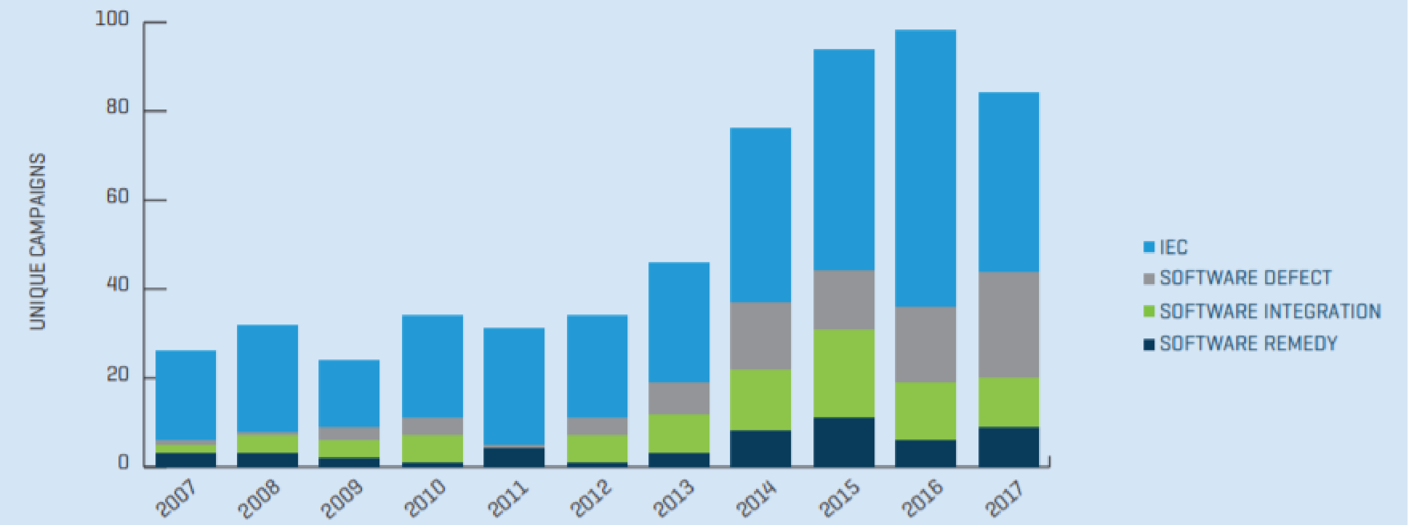


FIGURE 11 / RECALLS OF ELECTRONIC COMPONENTS BY YEAR*



With it the number of recalls has increased as well.

Source: 2018 Automotive warranty & recall report; Stout;

Use case

Electrical Vehicle Temperature control

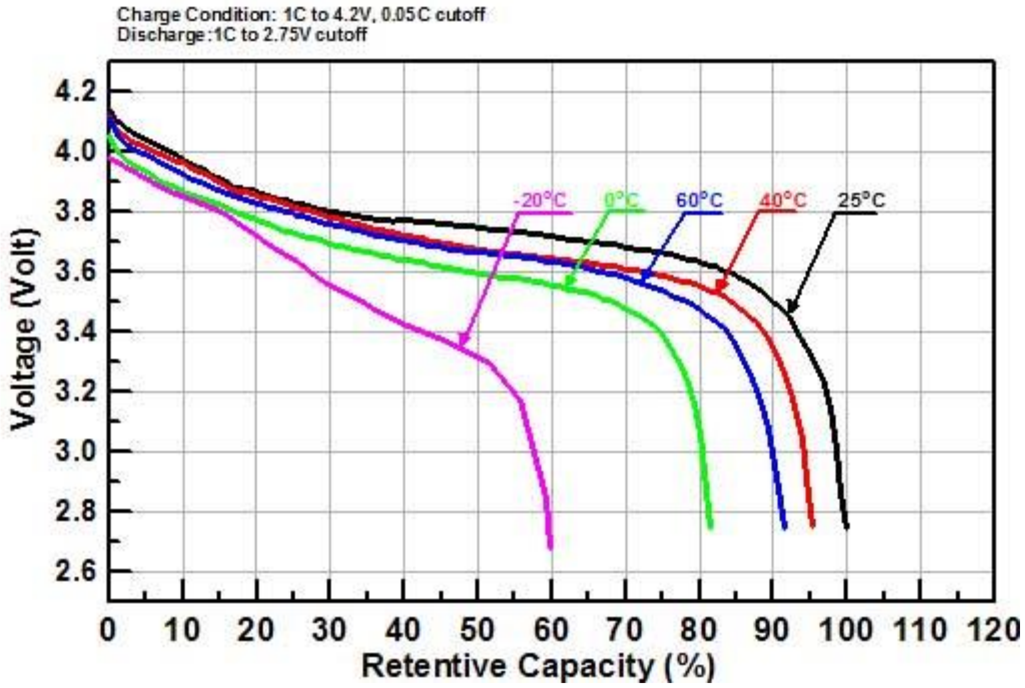
Simple Example:

- Batteries are sensitive to temperature:
  - Range is drastically reduced in cold,
  - Life time of battery is drastically reduced when to warm

- Consumers have grown accustom to
  - car ranges
  - thermal comfort in the car

- Cooling and heating have to be used both for battery and human comfort.

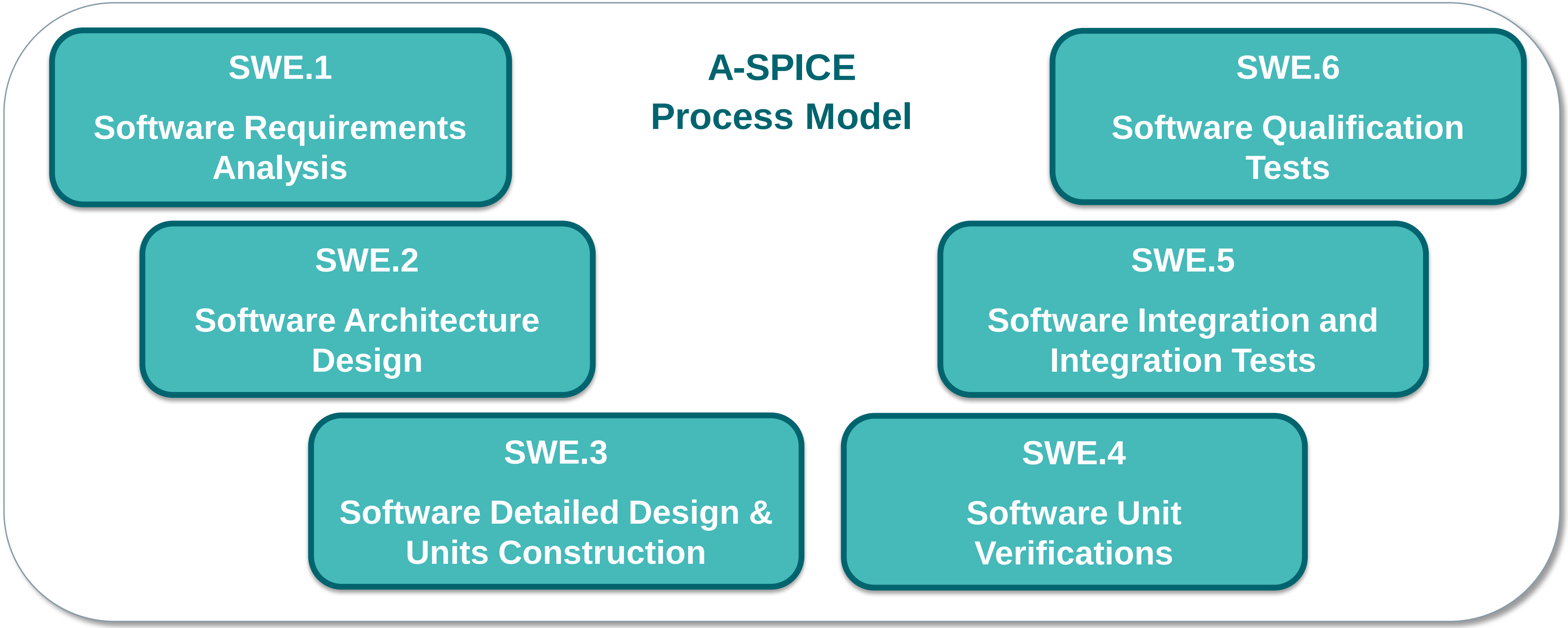
- This increase the complexity of the system and hence the software needed to control it.



https://optibike.com/lithium-battery-performance-in-cold-temperatures/

The process

Any software development process will do



A-SPICE Process Model

SWE.1

Software Requirements Analysis

SWE.2

Software Architecture Design

SWE.3

Software Detailed Design & Units Construction

SWE.4

Software Unit Verifications

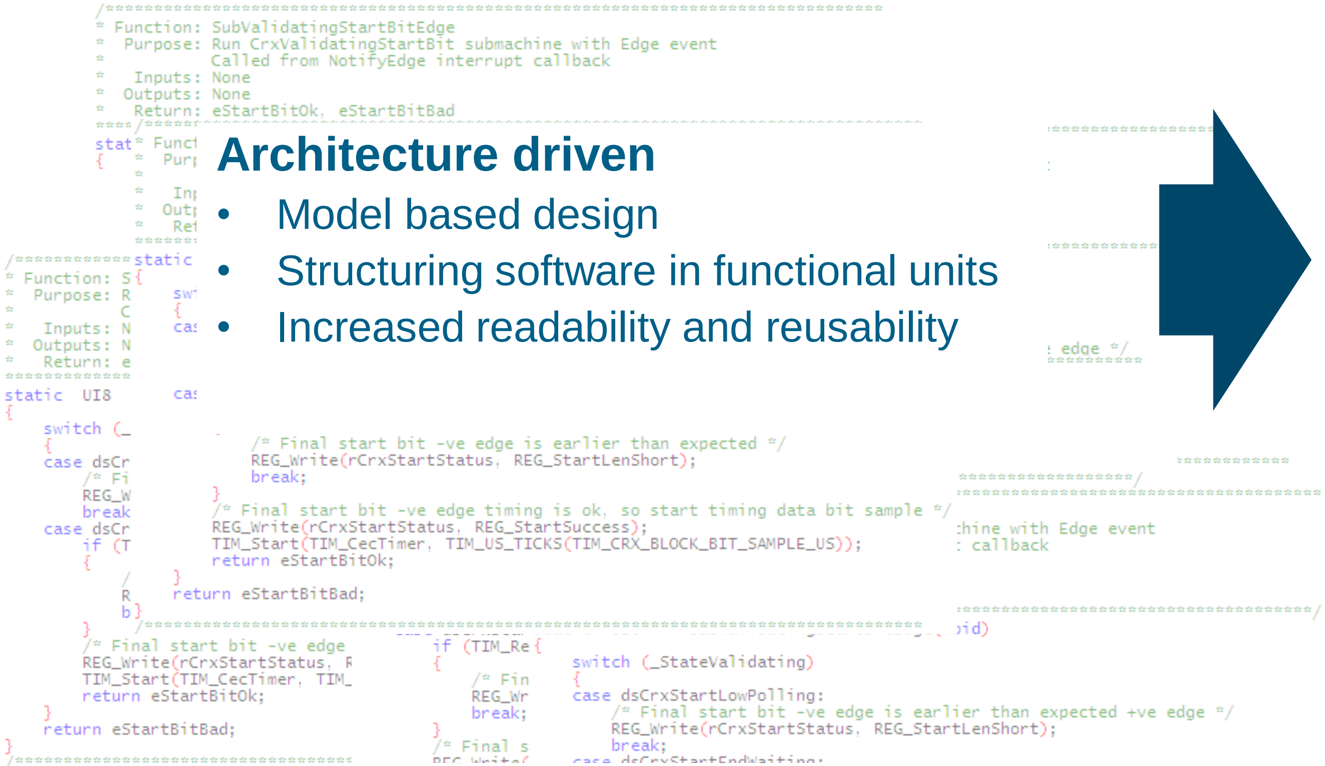
SWE.5

Software Integration and Integration Tests

SWE.6

Software Qualification Tests

Using the software architecture as a single source of truth



```
/*********************************************************************************
*  Function: SubValidatingStartBitEdge
*   Purpose: Run CrxValidatingStartBit submachine with Edge event
*            Called from NotifyEdge interrupt callback
*    Inputs: None
*   Outputs: None
*    Return: eStartBitOk, eStartBitBad
```

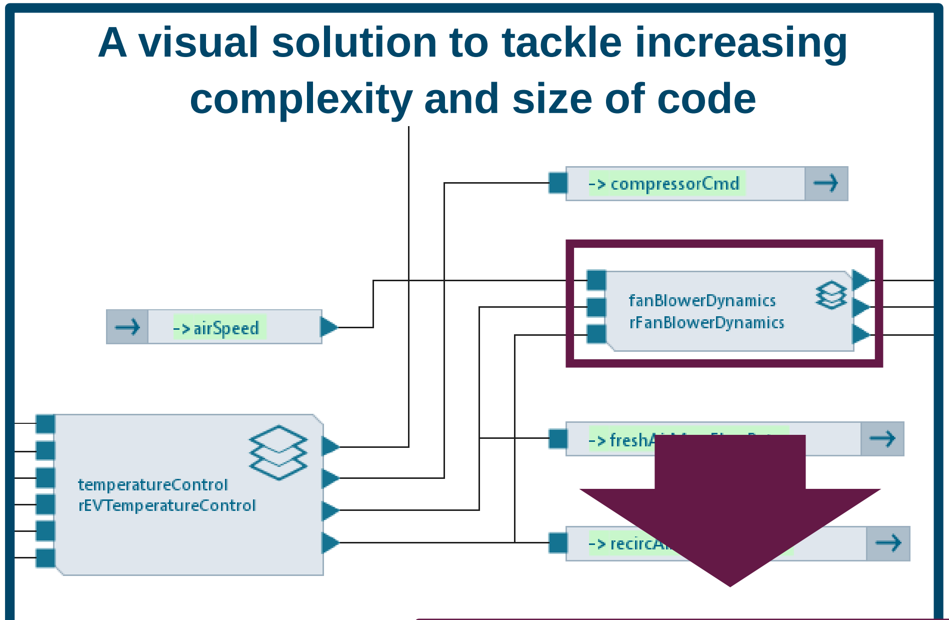
```
        /* Final start bit -ve edge is earlier than expected */
        REG_Write(rCrxStartStatus, REG_StartLenShort);
        break;
    }
    /* Final start bit -ve edge timing is ok, so start timing data bit sample */
    REG_Write(rCrxStartStatus, REG_StartSuccess);
    TIM_Start(TIM_CecTimer, TIM_US_TICKS(TIM_CRX_BLOCK_BIT_SAMPLE_US));
    return eStartBitOk;
  }
  return eStartBitBad;
```

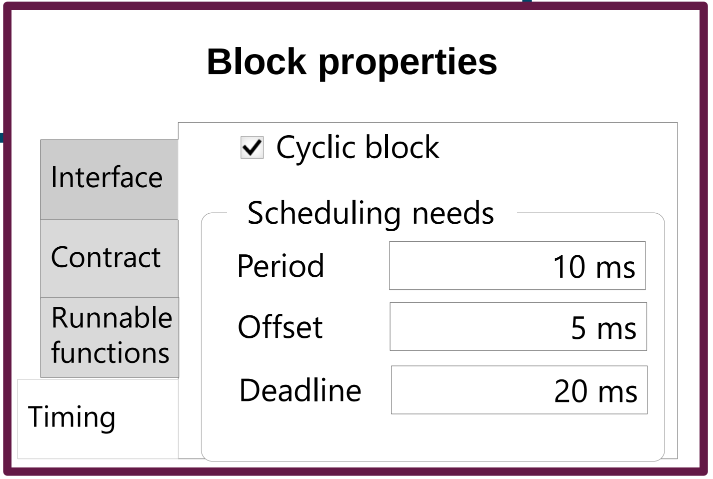
```
switch (_StateValidating)
{
case dsCrxStartLowPolling:
    /* Final start bit -ve edge is earlier than expected +ve edge */
    REG_Write(rCrxStartStatus, REG_StartLenShort);
    break;
```

Architecture driven

- Model based design
- Structuring software in functional units
- Increased readability and reusability

A visual solution to tackle increasing complexity and size of code





Block properties

Interface

Contract

Runnable functions

Timing

☑ Cyclic block

Scheduling needs

Period	10 ms
Offset	5 ms
Deadline	20 ms

Enrichment in form of architectural meta data

- Ensure consistency and completeness from requirements to implementation
- Frontload development of your testing environment early in the design process
- Integrate and test from different sources
- Trace links in the software development.

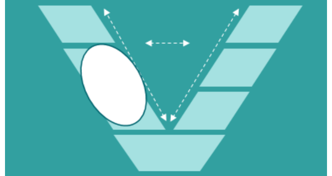
Content

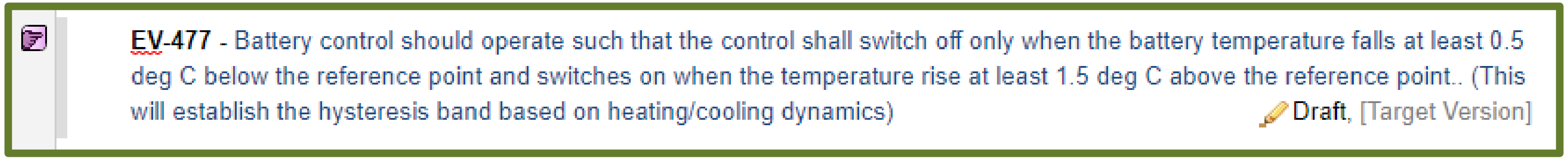
- Introduction
- **Contracts and assume guarantee analysis.**
- Keeping the traces
- Conclusion

SWE.2 Software Architecture Design
Datatypes of the I/O

SIEMENS
Ingenuity for life

SWE.2:
from informal
to formal

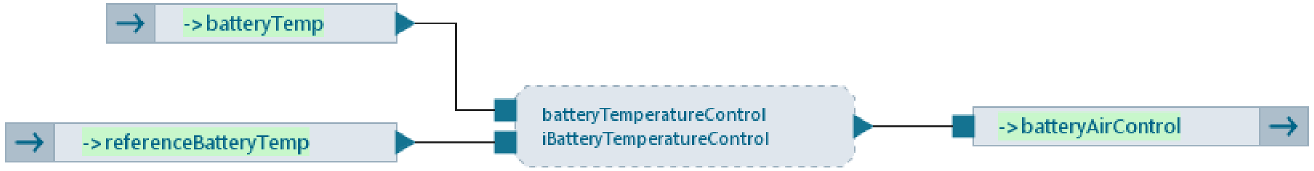




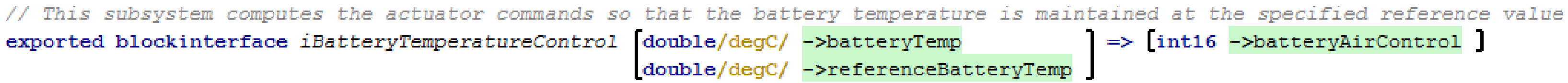
EV-477 - Battery control should operate such that the control shall switch off only when the battery temperature falls at least 0.5 deg C below the reference point and switches on when the temperature rise at least 1.5 deg C above the reference point.. (This will establish the hysteresis band based on heating/cooling dynamics)

Draft, [Target Version]

Architecture

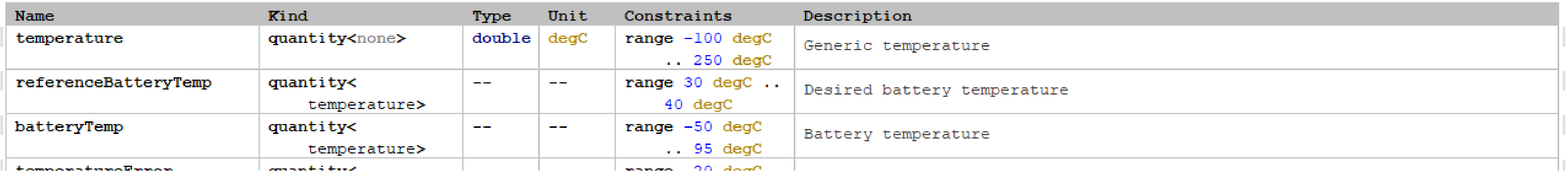


Interface definition



```
// This subsystem computes the actuator commands so that the battery temperature is maintained at the specified reference value
exported blockinterface iBatteryTemperatureControl [double/degC/ ->batteryTemp              ] => [int16 ->batteryAirControl ]
                                                   [double/degC/ ->referenceBatteryTemp     ]
```

Datatype, unit and range definition

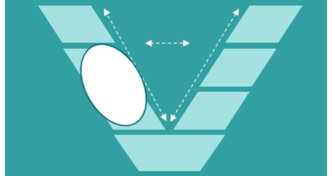


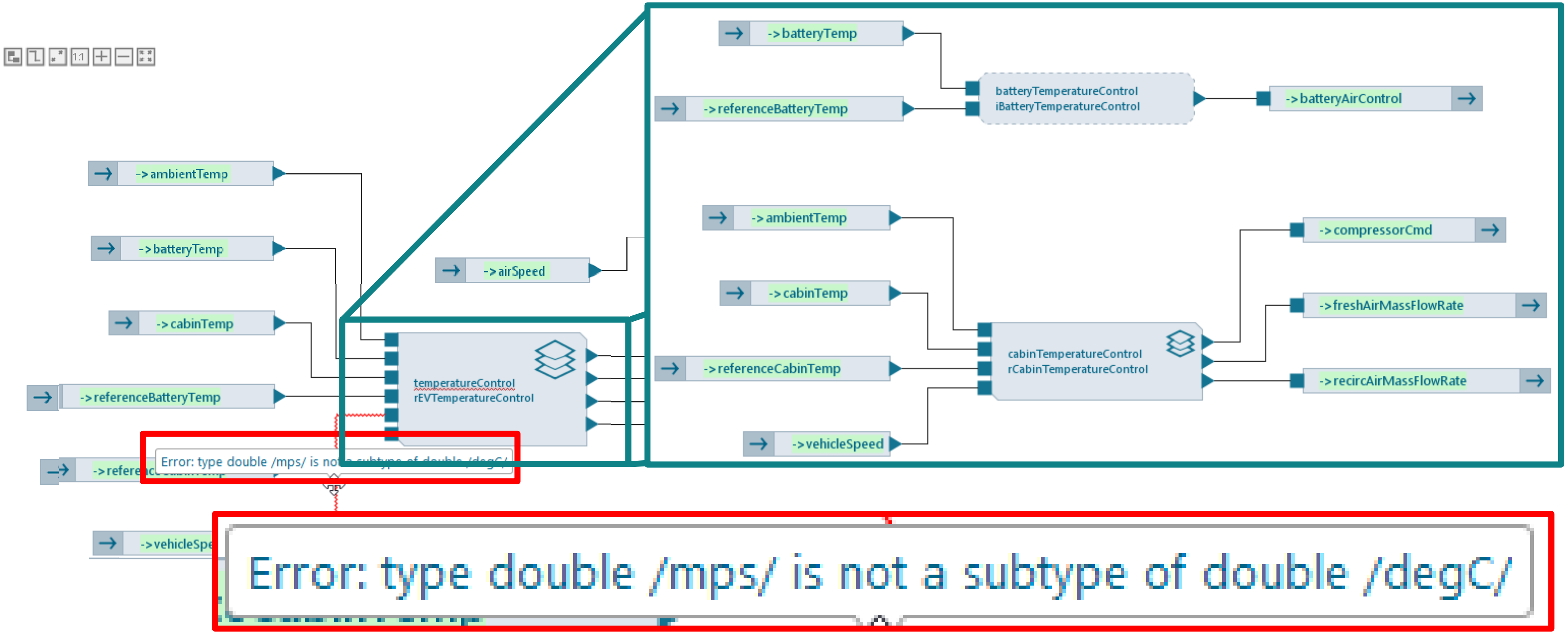
Name	Kind	Type	Unit	Constraints	Description
temperature	quantity<none>	double	degC	range -100 degC .. 250 degC	Generic temperature
referenceBatteryTemp	quantity< temperature>	--	--	range 30 degC .. 40 degC	Desired battery temperature
batteryTemp	quantity< temperature>	--	--	range -50 degC .. 95 degC	Battery temperature
temperatureError	quantity<			range 20 degC	

SWE.2: Software Architecture Design
Building the Architecture

SIEMENS
Ingenuity for life

SWE.2:
Building the architecture





Error: type double /mps/ is not a subtype of double /degC/

Error: type double /mps/ is not a subtype of double /degC/

Contracts in Software Engineering

Providing continuous requirements compliance

SIEMENS
Ingenuity for life

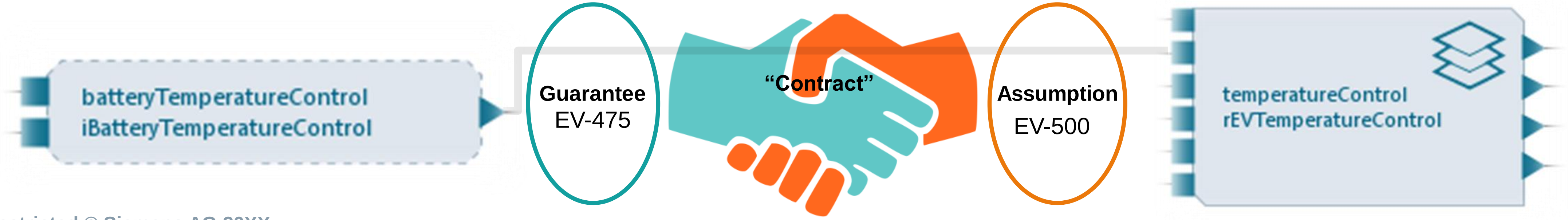
Contract in software development

- A **“Contract”** is an explicit binding **agreement** between 2 or more subsystems, wherein the subsystems **guarantees** they will deliver a certain output such that the other subsystems can **assume** certain inputs.
- Contracts are **formal executable requirements**.
- A “Contract” is split in 2 contracts a **pre- and a post-contract** connected to the subsystem itself to allow MBSE.
- **Assume Guarantee analysis: Consistency check of the software architecture** by checking the assumptions with respect to the guarantee of the connected components.

Allowing debugging of requirements before implementation, preventing mistakes and bugs in later development

Aiding assignment of responsibilities to a precise stakeholder*

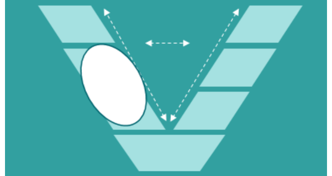
Supporting independent development of the different sub-systems while guaranteeing smooth system integration*



* Albert Benveniste, Benoît Caillaud, Dejan Nickovic, Roberto Passerone, Jean-Baptiste Raclet, et al.. Contracts for Systems Design: Theory. [Research Report] RR-8759, Inria Rennes Bretagne Atlantique; INRIA. 2015, pp.86. 〈hal-01178467〉

SWE.2 Software Architecture Design
Informal requirements to formal requirements

SWE.2:
from informal
to formal



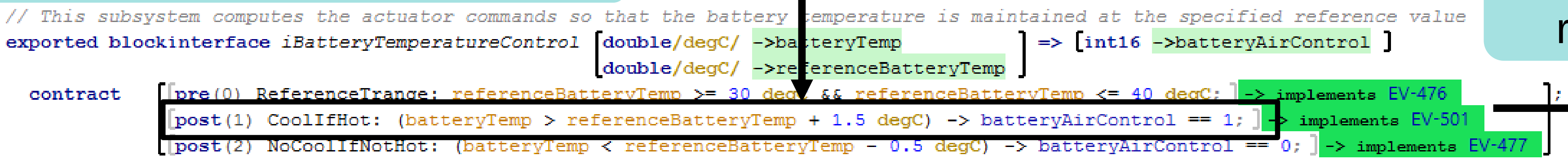
Informal Requirement

EV-501 Battery control should operate such that **the control shall switch on when** the **battery temperature rise at least 1.5 deg C above the reference point**. (This will establish the hysteresis band based on heating/cooling dynamics)

Formal Requirement

(**batteryTemp >= referenceBatteryTemp + 1.5 degC**) **-> batteryAirControl == 1**;

Contract in interface definition



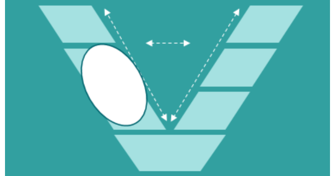
```
// This subsystem computes the actuator commands so that the battery temperature is maintained at the specified reference value
exported blockinterface iBatteryTemperatureControl [double/degC/  ->batteryTemp              ] => [int16 ->batteryAirControl ]
                                                   [double/degC/  ->referenceBatteryTemp     ]
  contract    [[pre(0) ReferenceTrange: referenceBatteryTemp >= 30 degC && referenceBatteryTemp <= 40 degC; ] -> implements EV-476    ];
              [[post(1) CoolIfHot: (batteryTemp > referenceBatteryTemp + 1.5 degC) -> batteryAirControl == 1; ] -> implements EV-501
              [[post(2) NoCoolIfNotHot: (batteryTemp < referenceBatteryTemp - 0.5 degC) -> batteryAirControl == 0; ] -> implements EV-477 ]
```

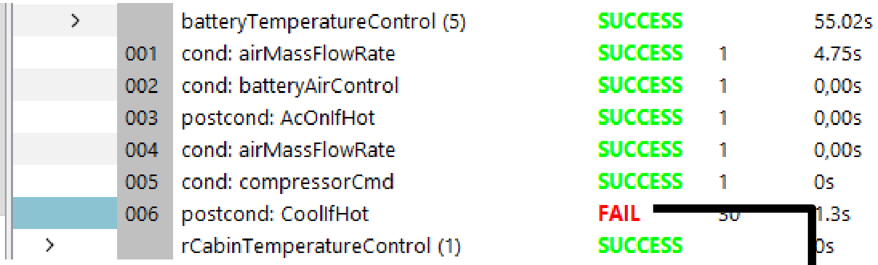
Trace to original requirement

Contract =
Mathematical formulation of the requirement that can be executed and verified.

SWE.2 Software Architecture Design
Formal verification

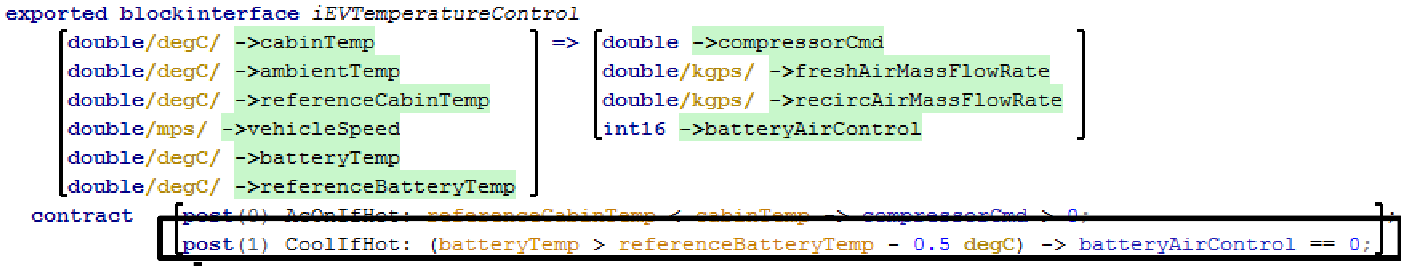
SWE.2:
Checking the consistency of the requirements





>	batteryTemperatureControl (5)	SUCCESS		55.02s
001	cond: airMassFlowRate	SUCCESS	1	4.75s
002	cond: batteryAirControl	SUCCESS	1	0,00s
003	postcond: AcOnIfHot	SUCCESS	1	0,00s
004	cond: airMassFlowRate	SUCCESS	1	0,00s
005	cond: compressorCmd	SUCCESS	1	0s
006	postcond: CoolIfHot	FAIL	30	1.3s
>	rCabinTemperatureControl (1)	SUCCESS		0s

Assume Guarantee analysis will verify the consistency of the software architecture by trying allowed post-contract values in the pre-contract of the connected block.



```
exported blockinterface iEVTemperatureControl
    [double/degC/ ->cabinTemp                 ] => [double ->compressorCmd                ]
    [double/degC/ ->ambientTemp               ]    [double/kgps/ ->freshAirMassFlowRate  ]
    [double/degC/ ->referenceCabinTemp        ]    [double/kgps/ ->recircAirMassFlowRate ]
    [double/mps/ ->vehicleSpeed               ]    [int16 ->batteryAirControl            ]
    [double/degC/ ->batteryTemp               ]
    [double/degC/ ->referenceBatteryTemp      ]
  contract  [post(0) AcOnIfHot: referenceCabinTemp < cabinTemp -> compressorCmd > 0;
            [post(1) CoolIfHot: (batteryTemp > referenceBatteryTemp - 0.5 degC) -> batteryAirControl == 0;
```

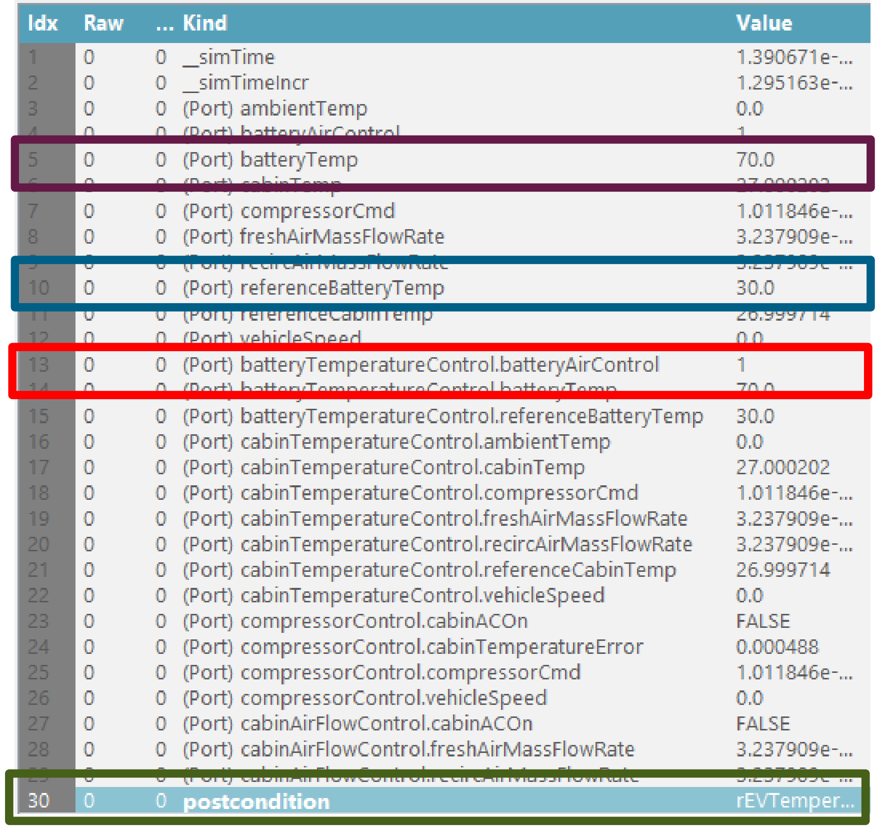
Conflict with another contract of a block in the architecture

Condition causing the conflict:

(batteryTemp > referenceBatteryTemp - 0.5 degC) -> batteryAirControl == 0;

SWE.2 Software Architecture Design
Formal verification

SWE.2:
Analysing
the
Inconsistency



Idx	Raw	...	Kind	Value
1	0	0	__simTime	1.390671e-...
2	0	0	__simTimeIncr	1.295163e-...
3	0	0	(Port) ambientTemp	0.0
4	0	0	(Port) batteryAirControl	1
5	0	0	(Port) batteryTemp	70.0
6	0	0	(Port) cabinTemp	27.000202
7	0	0	(Port) compressorCmd	1.011846e-...
8	0	0	(Port) freshAirMassFlowRate	3.237909e-...
9	0	0	(Port) recircAirMassFlowRate	3.237909e-...
10	0	0	(Port) referenceBatteryTemp	30.0
11	0	0	(Port) referenceCabinTemp	26.999714
12	0	0	(Port) vehicleSpeed	0.0
13	0	0	(Port) batteryTemperatureControl.batteryAirControl	1
14	0	0	(Port) batteryTemperatureControl.batteryTemp	70.0
15	0	0	(Port) batteryTemperatureControl.referenceBatteryTemp	30.0
16	0	0	(Port) cabinTemperatureControl.ambientTemp	0.0
17	0	0	(Port) cabinTemperatureControl.cabinTemp	27.000202
18	0	0	(Port) cabinTemperatureControl.compressorCmd	1.011846e-...
19	0	0	(Port) cabinTemperatureControl.freshAirMassFlowRate	3.237909e-...
20	0	0	(Port) cabinTemperatureControl.recircAirMassFlowRate	3.237909e-...
21	0	0	(Port) cabinTemperatureControl.referenceCabinTemp	26.999714
22	0	0	(Port) cabinTemperatureControl.vehicleSpeed	0.0
23	0	0	(Port) compressorControl.cabinACOn	FALSE
24	0	0	(Port) compressorControl.cabinTemperatureError	0.000488
25	0	0	(Port) compressorControl.compressorCmd	1.011846e-...
26	0	0	(Port) compressorControl.vehicleSpeed	0.0
27	0	0	(Port) cabinAirFlowControl.cabinACOn	FALSE
28	0	0	(Port) cabinAirFlowControl.freshAirMassFlowRate	3.237909e-...
29	0	0	(Port) cabinAirFlowControl.recircAirMassFlowRate	3.237909e-...
30	0	0	**postcondition**	rEVTemper...

Detailed analysis

From EVTemperatureControl SW component:

(batteryTemp > referenceBatteryTemp - 0.5 degC) -> batteryAirControl == 0;

batteryTemp = 70.0

referenceBatteryTemp = 30.0

(70.0 > 30 – 0.5 ? Yes so batteryAirControl == 0

From batteryTemperatureControl SW component:

batteryAirControl == 1;

Follow link: (batteryTemp >= referenceBatteryTemp + 1.5 degC) -> batteryAirControl == 1;

Since 70 > 30 + 1.5

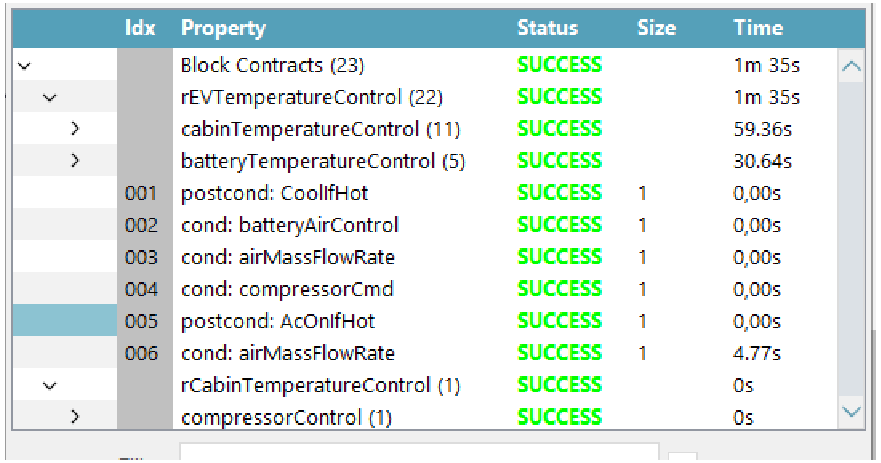
SWE.2 Software Architecture Design
Formal verification

SIEMENS
Ingenuity for life

SWE.2:
Solving the issue

3 possible errors:

- Inconsistent requirements
- Wrong translation into formal requirement
- Inconsistent architecture



	Idx	Property	Status	Size	Time
⌄		Block Contracts (23)	SUCCESS		1m 35s
⌄		rEVTemperatureControl (22)	SUCCESS		1m 35s
›		cabinTemperatureControl (11)	SUCCESS		59.36s
›		batteryTemperatureControl (5)	SUCCESS		30.64s
	001	postcond: CoolIfHot	SUCCESS	1	0,00s
	002	cond: batteryAirControl	SUCCESS	1	0,00s
	003	cond: airMassFlowRate	SUCCESS	1	0,00s
	004	cond: compressorCmd	SUCCESS	1	0,00s
	005	postcond: AcOnIfHot	SUCCESS	1	0,00s
	006	cond: airMassFlowRate	SUCCESS	1	4.77s
⌄		rCabinTemperatureControl (1)	SUCCESS		0s
›		compressorControl (1)	SUCCESS		0s

In EVTemperatureControl SW component:

(batteryTemp ~~>~~**<=** referenceBatteryTemp - 0.5 degC) -> batteryAirControl == 0;

Correction shows consistent software architecture with traceable links to the requirements.

Assume Guarantee Analysis: Frontload testing
Ensure consistency throughout software architecture using verifiable contracts

Content

SIEMENS

Ingenuity for life

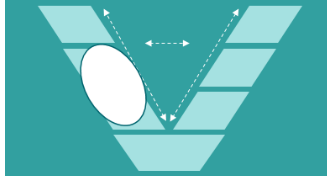
- Introduction
- Contracts and assume guarantee analysis.
- **Keeping the traces**
- Conclusion

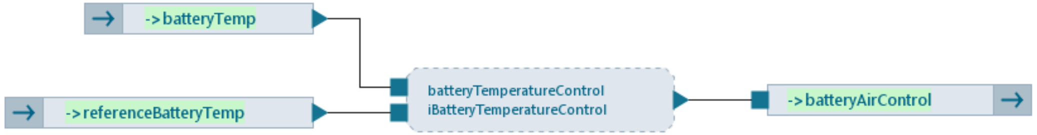
SWE.3 Software Detailed Design & Units Construction

Keeping the trace and the consistency

SIEMENS
Ingenuity for life

SWE.3: keep implementation consistent





Generate implementation shell

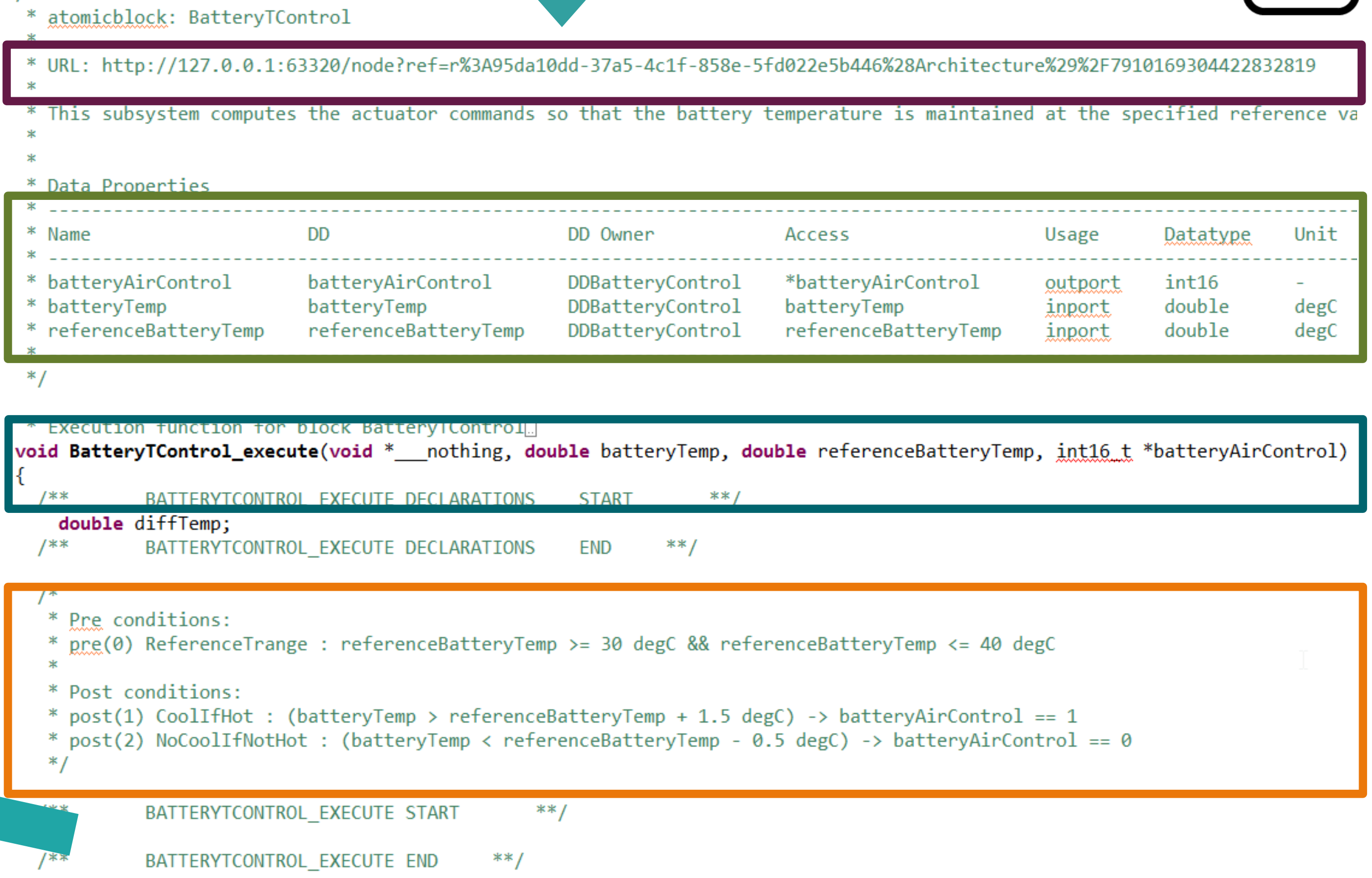
Function definition

Trace to architecture and requirements

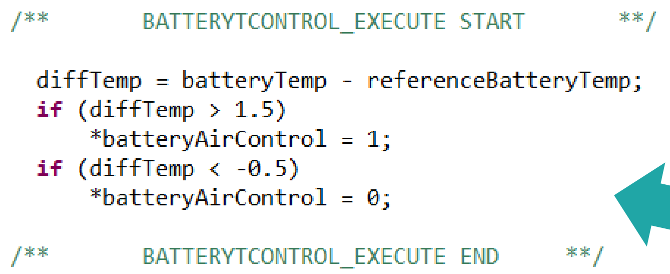
Datatypes and units of the software component I/O

Contracts

Implementation space



```
 * atomicblock: BatteryTControl
 *
 * URL: http://127.0.0.1:63320/node?ref=r%3A95da10dd-37a5-4c1f-858e-5fd022e5b446%28Architecture%29%2F7910169304422832819
 *
 * This subsystem computes the actuator commands so that the battery temperature is maintained at the specified reference va
 *
 *
 * Data Properties
 * ---------------------------------------------------------------------------------------------------------------------------
 * Name                 DD                   DD Owner          Access               Usage     Datatype   Unit
 * ---------------------------------------------------------------------------------------------------------------------------
 * batteryAirControl    batteryAirControl    DDBatteryControl  *batteryAirControl   outport   int16      -
 * batteryTemp          batteryTemp          DDBatteryControl  batteryTemp          inport    double     degC
 * referenceBatteryTemp referenceBatteryTemp DDBatteryControl  referenceBatteryTemp inport    double     degC
 *
 */

 * Execution function for block BatteryTControl
void BatteryTControl_execute(void *___nothing, double batteryTemp, double referenceBatteryTemp, int16_t *batteryAirControl)
{
  /**        BATTERYTCONTROL_EXECUTE DECLARATIONS    START       **/
    double diffTemp;
  /**        BATTERYTCONTROL_EXECUTE DECLARATIONS    END     **/

  /*
   * Pre conditions:
   * pre(0) ReferenceTrange : referenceBatteryTemp >= 30 degC && referenceBatteryTemp <= 40 degC
   *
   * Post conditions:
   * post(1) CoolIfHot : (batteryTemp > referenceBatteryTemp + 1.5 degC) -> batteryAirControl == 1
   * post(2) NoCoolIfNotHot : (batteryTemp < referenceBatteryTemp - 0.5 degC) -> batteryAirControl == 0
   */
  /**        BATTERYTCONTROL_EXECUTE START       **/

  /**        BATTERYTCONTROL_EXECUTE END     **/
```



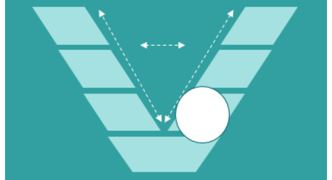
```
/**        BATTERYTCONTROL_EXECUTE START       **/

  diffTemp = batteryTemp - referenceBatteryTemp;
  if (diffTemp > 1.5)
      *batteryAirControl = 1;
  if (diffTemp < -0.5)
      *batteryAirControl = 0;

/**        BATTERYTCONTROL_EXECUTE END     **/
```

Restricted © Siemens

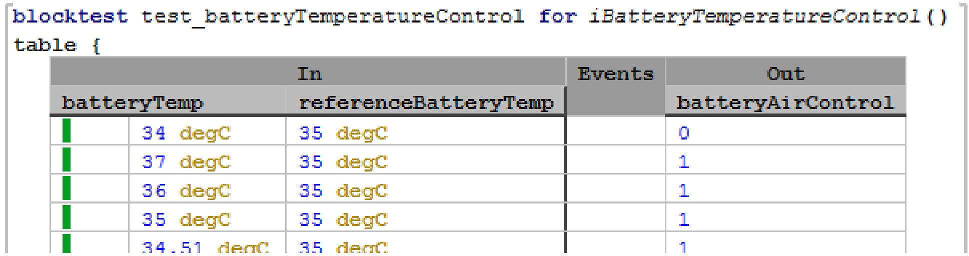
SWE.4 and 5 Software Unit Verifications and open loop integration testing

SIEMENS
Ingenuity for life

SWE.4/5:
Unit testing and integration testing

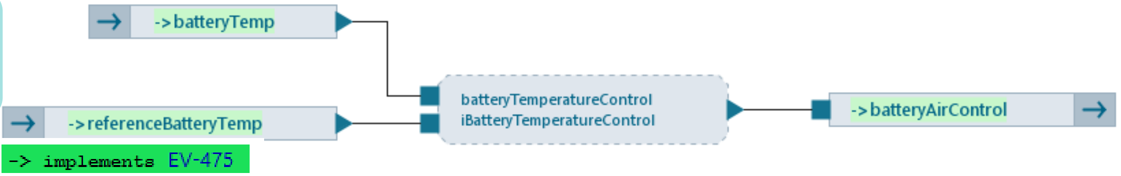


Test case definition based on requirements
Linked to the software architecture



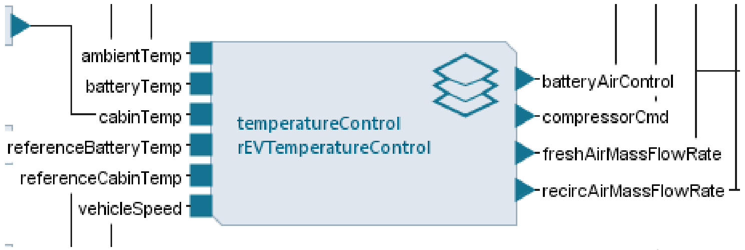
```
blocktest test_batteryTemperatureControl for iBatteryTemperatureControl()
table {
```

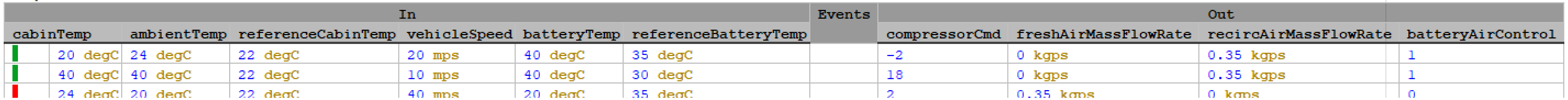
	In		Events	Out
	batteryTemp	referenceBatteryTemp		batteryAirControl
🟩	34 degC	35 degC		0
🟩	37 degC	35 degC		1
🟩	36 degC	35 degC		1
🟩	35 degC	35 degC		1
🟩	34.51 degC	35 degC		1



Run unit test for all implemented software components linked to the architecture

Integration Testing = Unit test of composed software components.





	In						Events	Out			
	cabinTemp	ambientTemp	referenceCabinTemp	vehicleSpeed	batteryTemp	referenceBatteryTemp		compressorCmd	freshAirMassFlowRate	recircAirMassFlowRate	batteryAirControl
🟩	20 degC	24 degC	22 degC	20 mps	40 degC	35 degC		-2	0 kgps	0.35 kgps	1
🟩	40 degC	40 degC	22 degC	10 mps	40 degC	30 degC		18	0 kgps	0.35 kgps	1
🟥	24 degC	20 degC	22 degC	40 mps	20 degC	35 degC		2	0.35 kgps	0 kgps	0

Using contracts will reduce the number of errors typically found at this stage and will reduce the typical glue code that has to be made to connect everything.

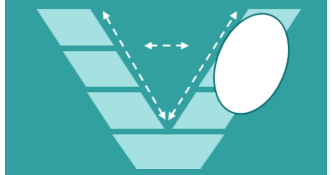
SWE.6 Software Qualification Tests Closed loop SIL validation

SIEMENS

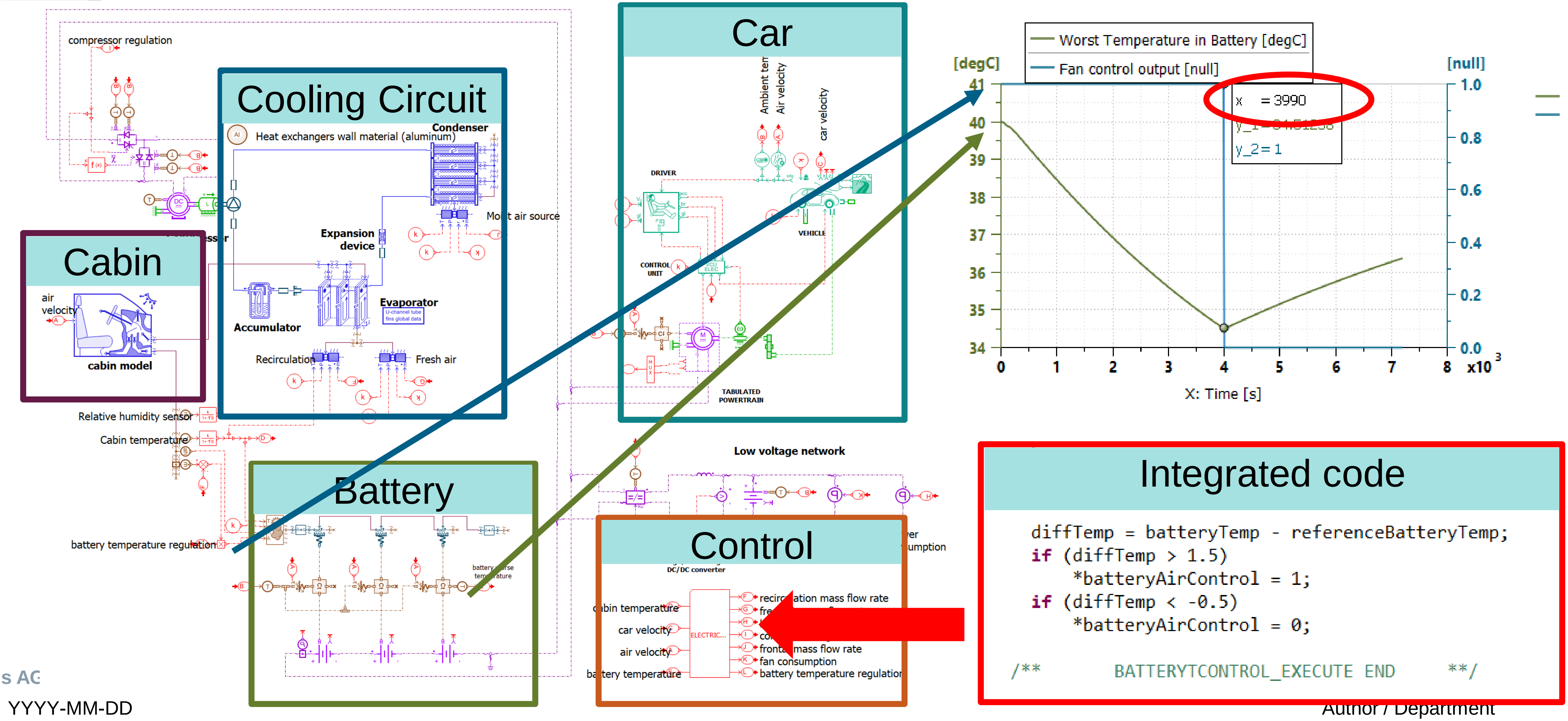
Ingenuity for life

SWE.6:

System testing



EV-488 - Battery temperature shall be regulated within the desired band within 600 seconds when the vehicle is baking in hot sun at an ambient temperature of 40 deg C for at least two hours and the battery is hot due to ambient temperature



Integrated code

```
diffTemp = batteryTemp - referenceBatteryTemp;
if (diffTemp > 1.5)
    *batteryAirControl = 1;
if (diffTemp < -0.5)
    *batteryAirControl = 0;

/**       BATTERYCONTROL_EXECUTE END        **/
```

Content

- Introduction
- Contracts and assume guarantee analysis.
- Keeping the traces
- **Conclusion**

Frontloaded workflow

SIEMENS
Ingenuity for life

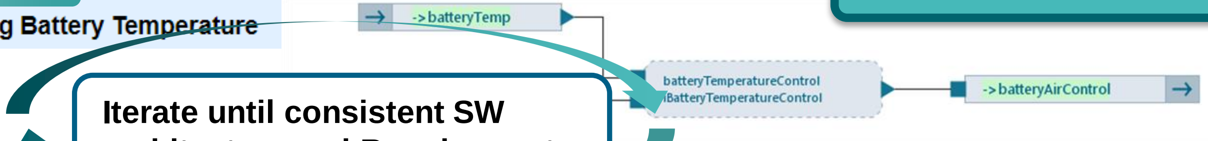
SWE.1: Requirements

1) Define requirements

EV-476 - Regulating Battery Temperature

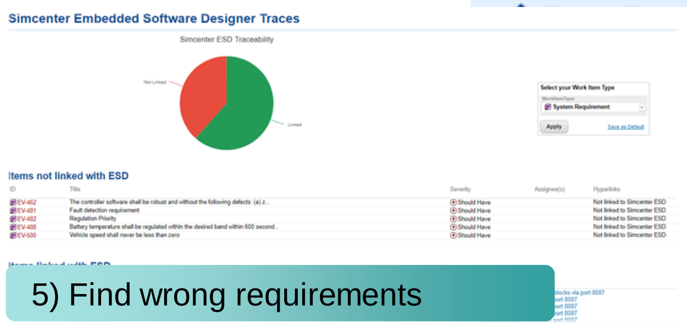
2) Build SW architecture in ESD

SWE.2: Software Architecture Design



Iterate until consistent SW architecture and Requirements

Functional contracts
Timing contracts



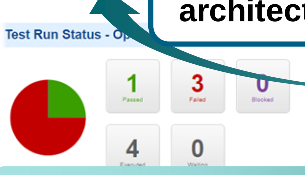
Simcenter Embedded Software Designer Traces

Simcenter ESD Traceability

Items not linked with ESD

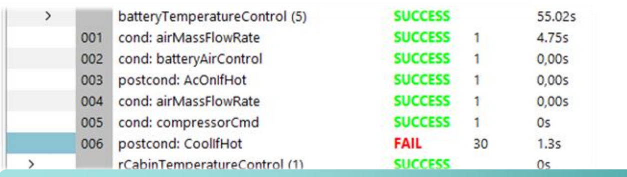
ID	Title	Severity	Assignee(s)	Hyperlinks
EV-462	The controller software shall be robust and without the following defects: (a) z...	Should Have		Not linked to Simcenter ESD
EV-481	Fault detection requirement	Should Have		Not linked to Simcenter ESD
EV-482	Regulation Priority	Should Have		Not linked to Simcenter ESD
EV-488	Battery temperature shall be regulated within the desired band within 600 second...	Should Have		Not linked to Simcenter ESD
EV-500	Vehicle speed shall never be less than zero	Should Have		Not linked to Simcenter ESD

Items linked with ESD



Test Run Status - Op

1	3	0
Passed	Failed	Blocked
4	0	
Executed	Waiting	



›		batteryTemperatureControl (5)	SUCCESS		55.02s
	001	cond: airMassFlowRate	SUCCESS	1	4.75s
	002	cond: batteryAirControl	SUCCESS	1	0,00s
	003	postcond: AcOnIfHot	SUCCESS	1	0,00s
	004	cond: airMassFlowRate	SUCCESS	1	0,00s
	005	cond: compressorCmd	SUCCESS	1	0s
	006	postcond: CoolIfHot	FAIL	30	1.3s
›		rCabinTemperatureControl (1)	SUCCESS		0s

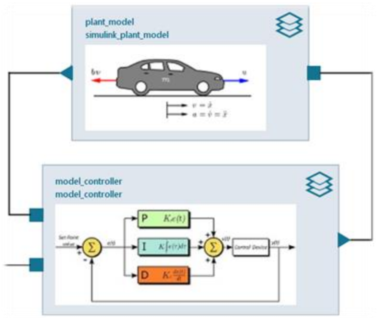
5) Find wrong requirements

4) Orchestrate Analyses in test framework

3) Analyze consistency of architecture and Requirements

SWE.6 system Validation

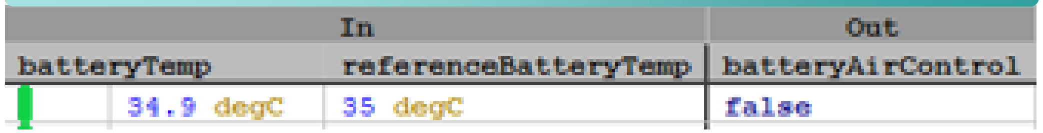
Export code for closed loop validation in system simulation tool



SWE.5 Integration testing

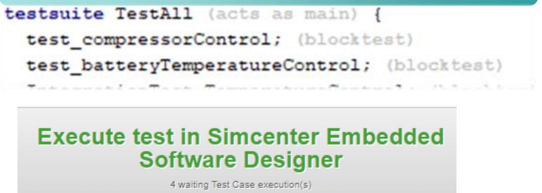
SWE.4: Unit testing

Write unit and integration test case



In		Out
batteryTemp	referenceBatteryTemp	batteryAirControl
34.9 degC	35 degC	false

Run test suits

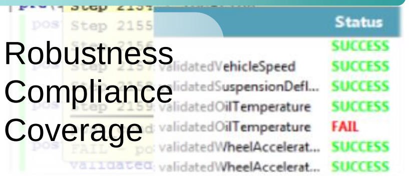


```
testsuite TestAll (acts as main) {
  test_compressorControl; (blocktest)
  test_batteryTemperatureControl; (blocktest)
```

Execute test in Simcenter Embedded Software Designer

4 waiting Test Case execution(s)

Analyses code on bugs



	Status
	SUCCESS
validatedVehicleSpeed	SUCCESS
validatedSuspensionDefl...	SUCCESS
validatedOilTemperature	SUCCESS
validatedOilTemperature	FAIL
validatedWheelAccelerat...	SUCCESS
validatedWheelAccelerat...	SUCCESS

Robustness
Compliance
Coverage

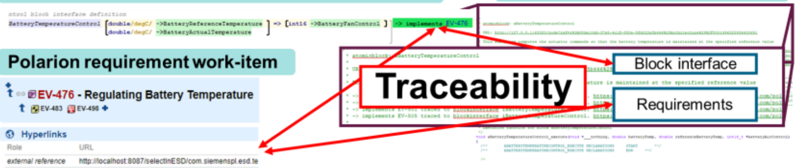
Orchestrate in test framework

SWE.3: Implementation

Extract Implementation shells from architecture

ESD assets (Block interfaces/ test cases / analysis)

Generated implementation shell



```
ntrol block interface definition
BatteryTemperatureControl [double/degC/ ->BatteryReferenceTemperature ] => [int16 ->BatteryFanControl ] -> implements EV-476
                          [double/degC/ ->BatteryActualTemperature    ]
```

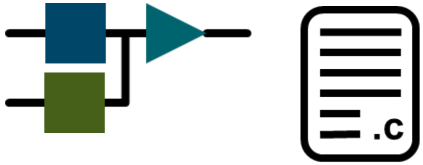
Polarion requirement work-item

EV-476 - Regulating Battery Temperature
EV-483 EV-498

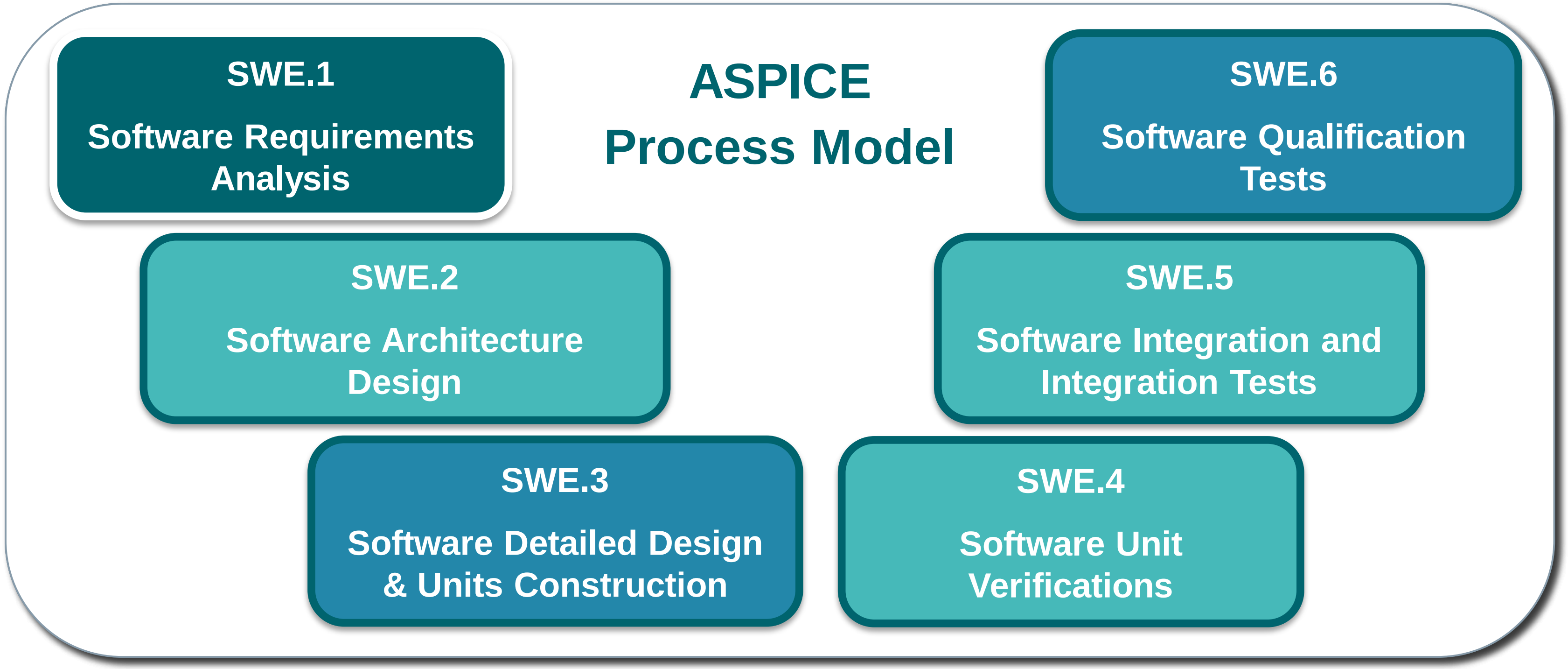
Hyperlinks

Role	URL
external reference	http://localhost:8087/selectInESD/com.siemenspl.esd.te

Implement in desired environment



Model based software architecture design by contract
Solution scope of Simcenter Embedded Software Designer



ASPICE Process Model

SWE.1 Software Requirements Analysis

SWE.2 Software Architecture Design

SWE.3 Software Detailed Design & Units Construction

SWE.4 Software Unit Verifications

SWE.5 Software Integration and Integration Tests

SWE.6 Software Qualification Tests

- **Upfront analysis and Verification** of the architecture consistency and interfaces
- **Easy integration** after implementation
- **Automate** unit and integration testing

- **Ensure consistency** throughout the design process
- **Distribute implementation** over different tools and internal and external suppliers.
- **Validate** performance with **SIL** testing

- Ensure **bidirectional traceability** by connecting to an ALM-tool
- **Automated test result reporting** and **communication** between all the stakeholders

Solution scope

Extended Support

Integration ALM-tool