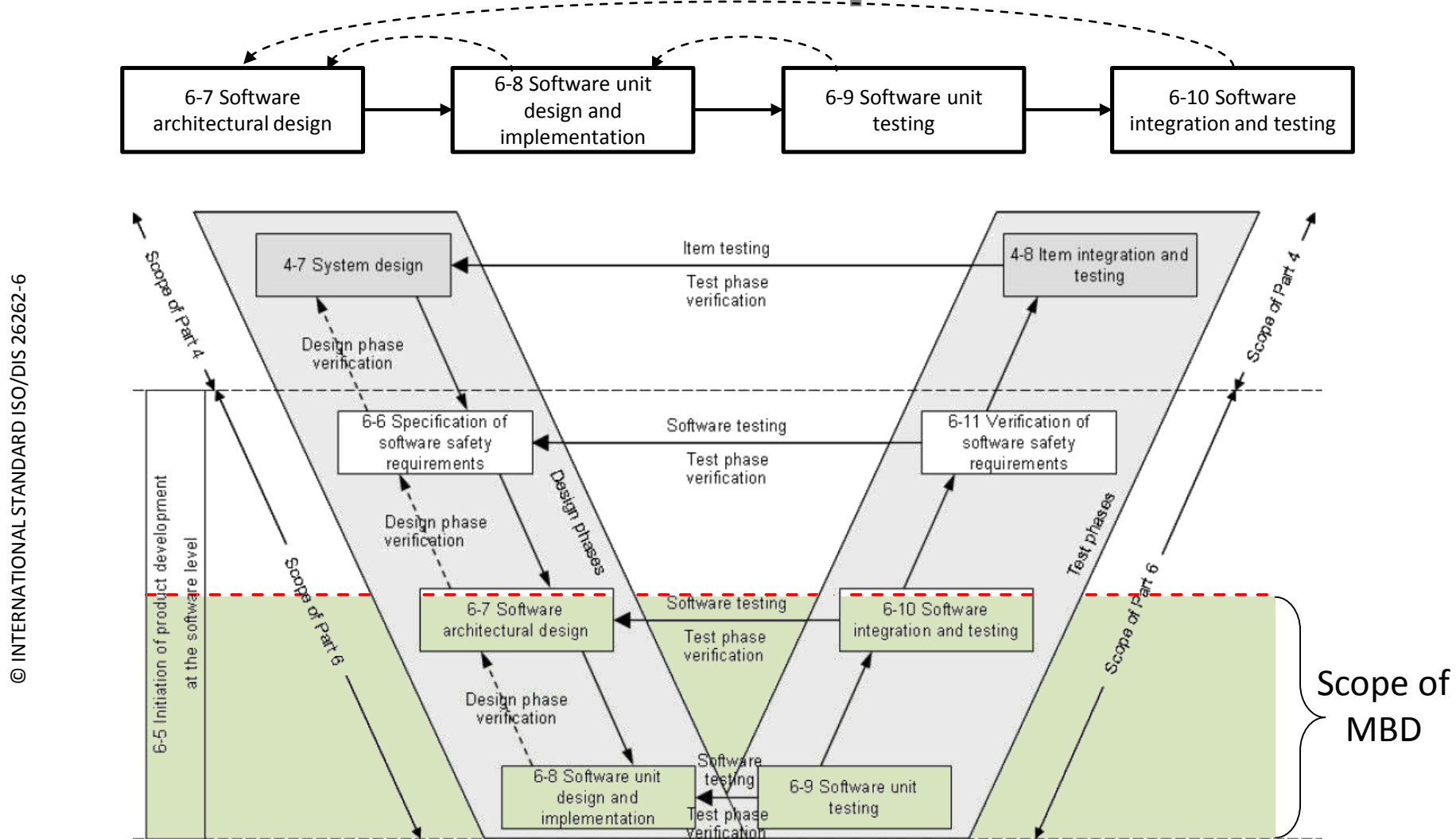


# Test Driven Development

with Simulink® & Stateflow®

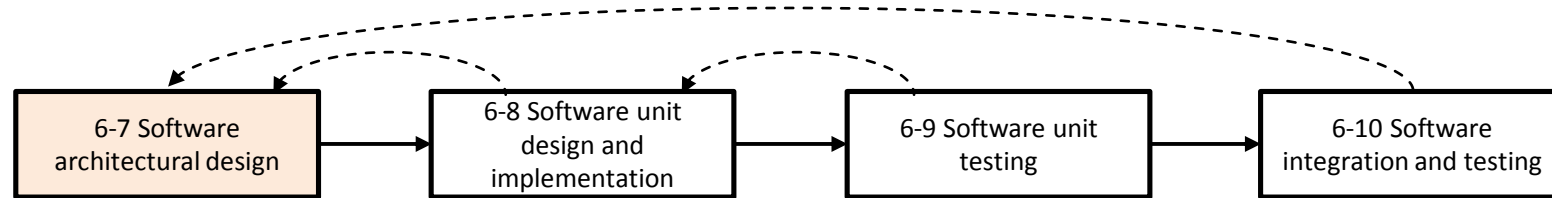
# Software Development Model



# Agenda

- Software architectural design
- Software unit design and implementation
- Software unit testing
- Software integration and testing

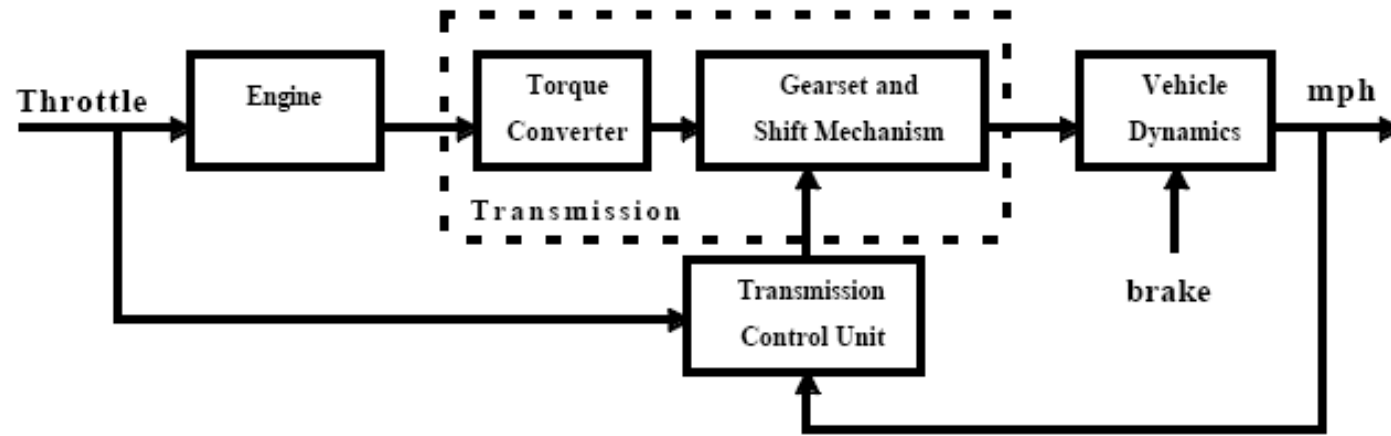
# Software Architectural Design



... develop and verify a software architectural design that realizes the software requirements

# Traditional Architecture

- Usually documented as text + drawings:



- Limited verification ability
- Limited reusability

# Architectural Notations

- ISO 26262-6 §7:

Table 2 — Notations for software architectural design

Methods		ASIL			
		A	B	C	D
1a	Informal notations	++	++	+	+
1b	Semi-formal notations	+	++	++	++
1c	Formal notations	+	+	+	+

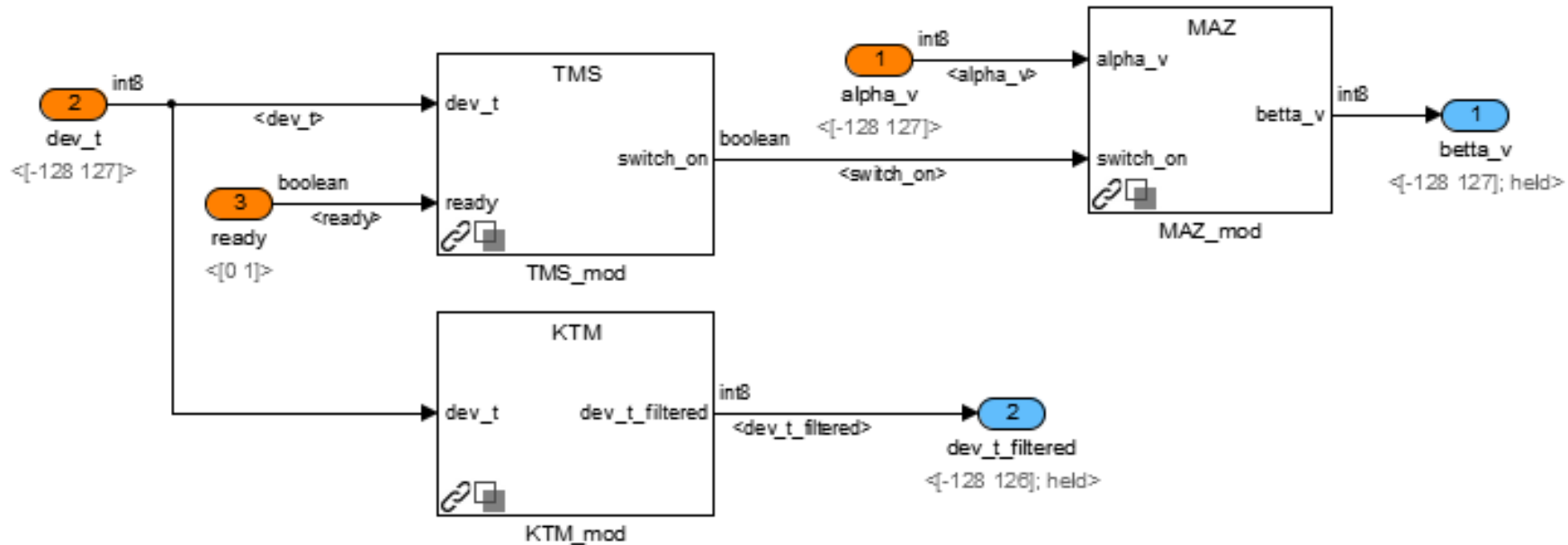
- ISO 26262-6 Annex B (informative):

Modelling is usually carried out with commercial off-the-shelf modelling and simulation software tools. They support the development and definition of system/software elements, and their **connections** and **interfaces** by **semi-formal graphical models**. These models employ **editable, hierarchical block diagrams** (e.g. **control diagrams**) and extended state transition diagrams (e.g. **state charts**)...

...Hierarchically structured modularity is used in order to **control complexity**. A model consists of function blocks with **well-defined inputs and outputs**. Function blocks are **connected** within the block diagram by directed edges **between their interfaces**, which describe **signal flows**.

# Model-Based Architecture

- Semi-formal notation: File > Print



- Ability to verify compliance of interfaces

# Data & Control Flow Analysis

- ISO 26262-6 §7:

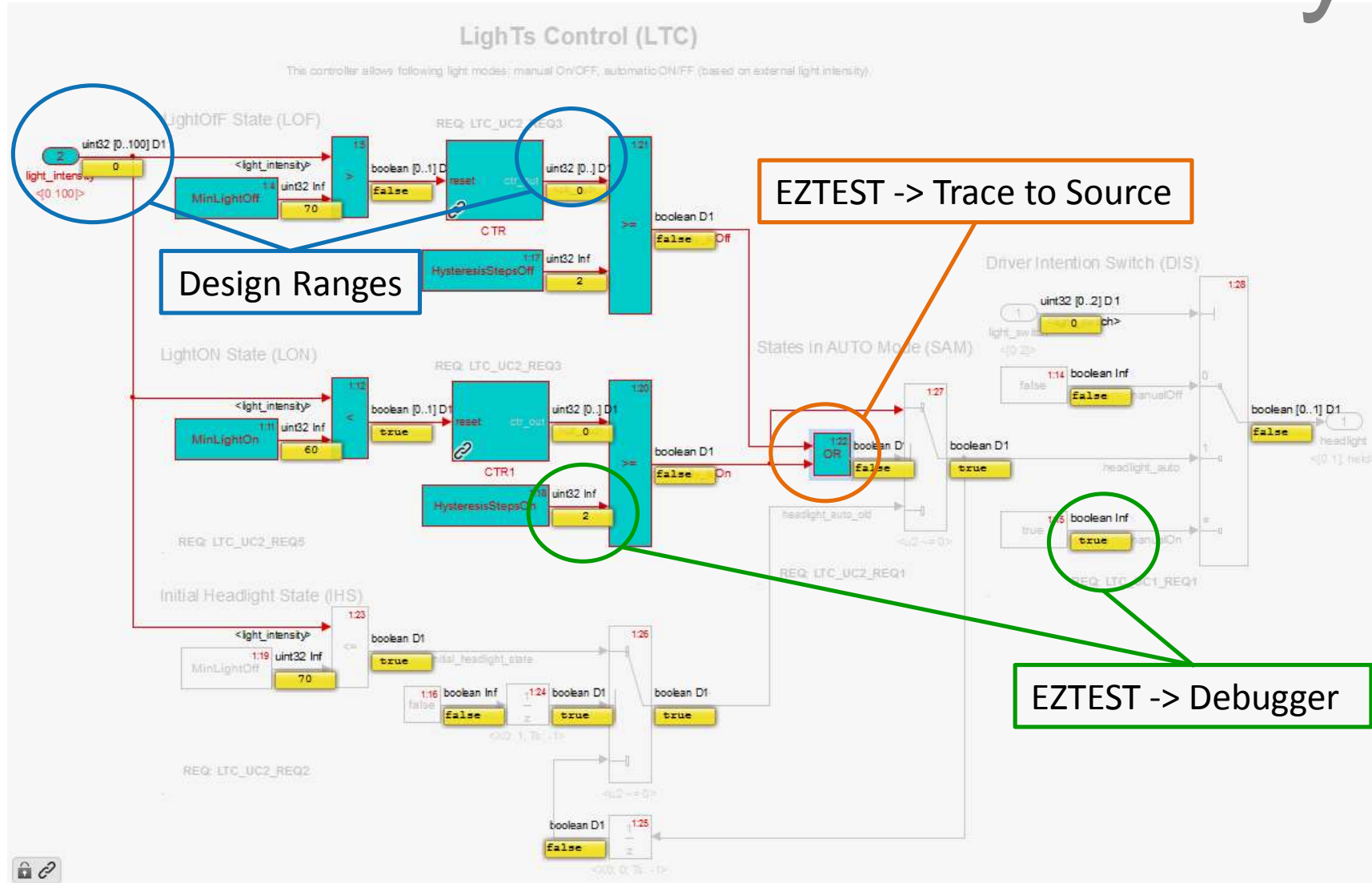
Table 4 — Mechanisms for error detection at the software architectural level

Methods		ASIL			
		A	B	C	D
1a	Range checks of input and output data	++	++	++	++
1b	Plausibility check <sup>a</sup>	+	+	+	++
1c	Detection of data errors <sup>b</sup>	+	+	+	+
1d	External monitoring facility <sup>c</sup>	o	+	+	++
1e	Control flow monitoring	o	+	++	++
1f	Diverse software design	o	o	+	++

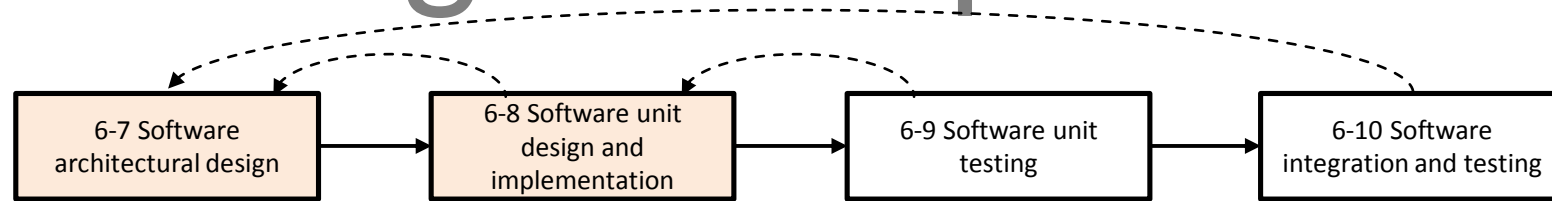
- Assert signals and check ranges
- Trace signals to source and destination
- Coverage & step-by-step debugger (later)



# Data & Control Flow Analysis



# Unit Design & Implementation



... software units are specified and implemented in accordance with the software architectural design and the associated software requirements followed by static model verification.

# Requirements-Based Design

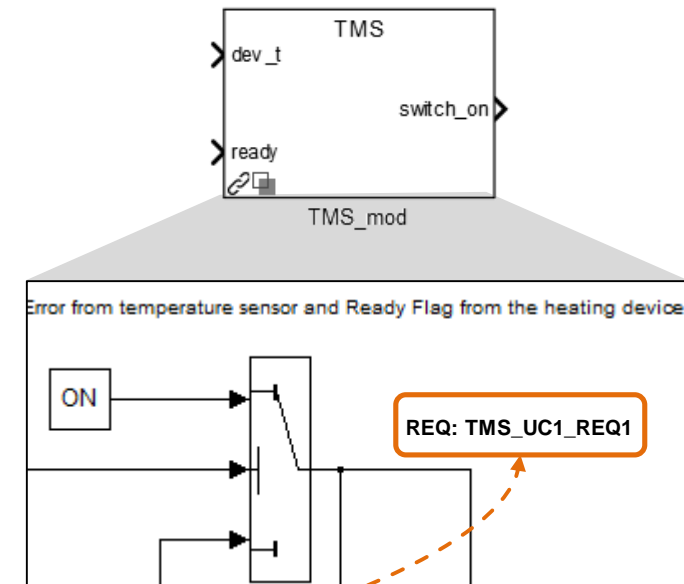
## Unit Specification for Thermostat (TMS) of a Heating Device

TMS unit turns the heating device on or off based on temperature deviation measured by temperature sensor and ready signal from the heating device.

### Interface:

Name	Type	Data Type	Dimension	Min	Max
dev_t	I	int8	1	-128	127
ready	I	boolean	1	0	1
switch_on	O	boolean	1	0	1
OFF	P	boolean	1	0	0
OFF_threshold	P	int8	1	-20	0
ON	P	boolean	1	1	1
ON_threshold	P	int8	1	0	20

Use Case ID	Description in natural language
TMS_UC1	<p>The heating device shall be switched ON, if the temperature deviation (dev_t) is bigger than or equal to ON_threshold and the ready signal is TRUE.</p> <p>The heating device should be switched OFF, if the temperature deviation (dev_t) is smaller than or equal to OFF_threshold or the ready signal is FALSE.</p> <p>In other cases the old state of the controller output should be preserved.</p>
Requirement ID	Description in mathematical notation
TMS_UC1_REQ1	<pre>IF   (dev_t &gt;= ON_threshold) AND (ready == TRUE) THEN   switch_on(k) = ON ELSE IF   dev_t &lt;= OFF_threshold THEN   switch_on(k) = OFF ELSE   switch_on(k) = switch_on(k-1) END</pre>

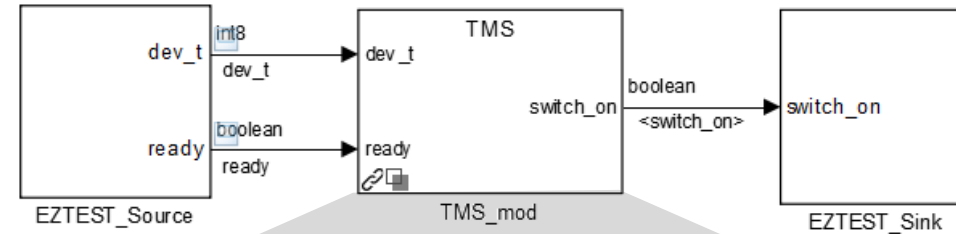


trace

Executable specifications help understanding the requirements

# Test-Driven Development

## Test Harness for Thermostat System



## Verification Report for TMS\_EZ

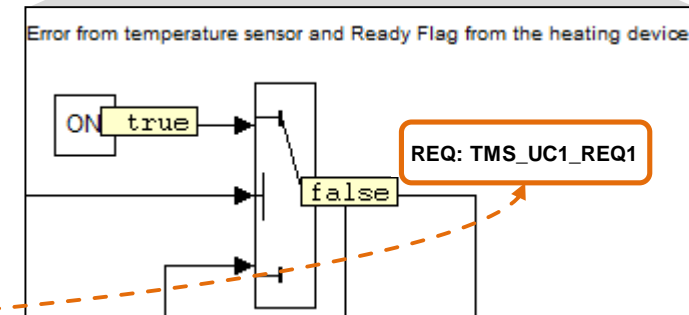
☐ Summary

☒ Interface

Name	Type	Data Type	Dimension	Min	Max	Description
dev_t	I	int8	1	-128	127	
ready	I	boolean	1	0	1	
switch_on	O	boolean	1	0	1	
ON	C	boolean	1	1	1	
OFF	C	boolean	1	0	0	
ON_threshold	P	int8	1	0	20	
OFF_threshold	P	int8	1	-20	0	

☒ TMS\_UC1\_REQ1\_1

t	dev_t	ready	switch_on_ref	switch_on	ON	OFF	ON_threshold	OFF_threshold	REQ: TMS_UC1_REQ1_1
0	-4	1	0	0	1	0	2	-3	Go below OFF_threshold
0.04	-2			0					
0.06	4		1	1					Go above ON_threshold
0.08	1		?	1					
0.2	2			1					
0.22	3			1					
0.24	4	0	0	0					Switch off based on 'ready' flag
0.26	5	1	1	1					
0.28	-128		0	0					Go below OFF_threshold
0.3	-4			0					
0.32	127		1	1					Go above ON_threshold
0.34	3			1					
0.36				1					



## Test-Driven Development:

- Natural Language
- Executable Specification
- Requirements-Based Test

# Software Unit Verification

Table 9 — Methods for the verification of software unit design and implementation

Methods		ASIL			
		A	B	C	D
1a	Walk-through <sup>a</sup>	++	+	o	o
1b	Inspection <sup>a</sup>	+	++	++	++
1c	Semi-formal verification	+	+	++	++
1d	Formal verification	o	o	+	+
1e	Control flow analysis <sup>b,c</sup>	+	+	++	++
1f	Data flow analysis <sup>b,c</sup>	+	+	++	++
1g	Static code analysis	+	++	++	++
1h	Semantic code analysis <sup>d</sup>	+	+	+	+

“...inspections quickly brought all the developers up to the level of the best developers”

- Tackett and Van Doren 1999

# Formal Inspection Advantages

- Joint team effort and collaboration
- Know-how and experience exchange
- Readable & well-documented models
- Bugs are found earlier and faster
- Efficient development process

# Static Analysis Automation

The image displays the EZCHECK static analysis tool interface, which is used for automating static analysis of Simulink models. The main window shows the results of the analysis for a model named "TMS/TMS\_var/TerMoStat".

**Before:** The initial model, labeled "Thermostat", is shown in a red box. It is a simple block diagram with an input "error" and an output "ctrl". The model contains a "Relational Operator" block and a "Switch" block.

**Thermostat (TMS):** The model after static analysis, shown in a green box. It is a more complex state machine model. The description states: "TMS turns the heating device ON or OFF based on temperature Set-Point Error from temperature sensor and Ready Flag from the heating device." The model includes blocks for "ON\_threshold", "OFF\_threshold", "ON\_or\_OLD", and "OFF\_or\_OLD". It also features a "ctrl" output and a "switch\_on" output.

**EZCHECK V3.1.0 Results:** The tool has identified 15 errors (ev\_0002 to ev\_0151) across various categories, including Safety Modeling Guidelines, Interface Specification, and Model Documentation. The results are listed in the "EZCHECK" pane.

**Result Viewer:** The "Result Viewer" pane displays the test run details for the "TMS/TMS\_var/TerMoStat" model. The results show a successful analysis with 32 passes, 0 failures, 0 warnings, and 0 not-run tests.

Test Run		Test Object	
User	TechDirector	System	<a href="#">TMS/TMS_var/TerMoStat</a>
Date	28-Jun-2016	User	TechDirector
Start	22:17:07	Last Saved	2016-06-28 22:11:24
End	22:17:10	Model	TMS_lib
Test Environment		Model Version	
Windows	7 Professional (6.1) 64-bit	Type	Simulink Library
MATLAB	8.4 (R2014b) 64-bit	Sample Time	inherited
EZCHECK	v3.1.0	Block Count	11

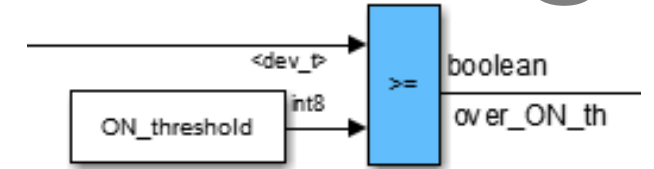
**Thermostat (TMS) Test Results:**

Test Run	Test Object
Pass: 32	Fail: 0
Warning: 0	Not Run: 0

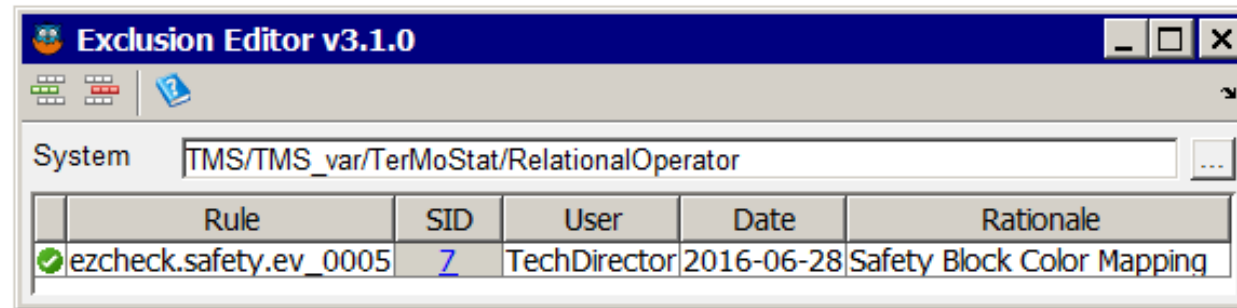
The interface also includes a "Modify" button and a "Run Selected" button. The copyright notice at the bottom indicates the software is © Copyright TechCoaching GmbH 2011-2016.

# Intrusive Exclusion Handling

- Integrated Exclusion-Editor (Kontext-Menu)
  - Rule + SID
  - User + Date
  - Rationale



✓ ev\_0005: Simulink Subset Usage



- Review-Support & Reporting (Links to Model)

## Check Consistency of Block Parameters

Identify blocks which do not comply with the reference library parameters.

Context-Based Exclusion Documentation for Simplified Reviews

✓ Passed

The following blocks have unsupported parameters:

Block Name	Type	Parameter	Actual value	Supported values	Excluded	User	Date	Rationale
<a href="#">RelationalOperator</a>	RelationalOperator	AttributesFormatString	<%<BackgroundColor>>		<a href="#">Yes</a>	TechDirector	2016-06-28	Safety Block Color Mapping
		BackgroundColor	lightBlue	white				



# Tool Help

- EZCHECK Workflow
- Modeling Guidelines
- EZCHECK API

EZCHECK users become productive within 10 minutes.

The screenshot displays the 'Supplemental Software' application window. The left pane shows a tree view of the 'EZCHECK' workflow, with 'ev\_0009 Model Documentation' selected. The right pane shows the 'Model Documentation' page, which includes sections for ID, Title, Priority, Scope, MATLAB, Static Check, Prerequisites, Description, and Simulink Documentation. The 'Simulink Documentation' section contains a table with a row for 'Thermostat (TMS)'.

**Supplemental Software**

File Edit View Go Favorites Window Help

Search

Contents Search Results

- EZCHECK
  - Getting Started
  - User's Guide
  - Reference
  - Safety Modeling Guidelines
    - ev\_0001 Unit Folder Structure
    - ev\_0002 Constant Block Usage and Appearance
    - ev\_0003 Interface Specification
    - ev\_0004 Interface Consistency
    - ev\_0005 Simulink Subset Usage
    - ev\_0006 Identifiers
    - ev\_0007 Generic Functionality
    - ev\_0008 Proper Labeling of Signal Lines
    - ev\_0009 Model Documentation
    - ev\_0010 Restricted Usage of Data Types
    - ev\_0011 Division by Zero
    - ev\_0012 Naming Conventions
    - ev\_0013 Gain Block Usage
    - ev\_0081 Unconnected Ports and Signal Lines
    - ev\_0110 Calculations in Block Settings
    - ev\_0123 Stateflow Port Names
    - ev\_0125 Scope of Signals and Variables
    - ev\_0127 MATLAB Commands in Stateflow
    - ev\_0132 Transitions in Flowcharts
    - ev\_0137 States in State Machines
    - ev\_0151 Transition Action Patterns
    - ev\_0230 Stateflow Transitions
    - ev\_0281 Naming of Trigger and Enable Ports
    - ev\_0301 Controller Model
    - ev\_0302 Model Configuration Settings
    - ev\_0331 Structure Layer
    - ev\_0401 Warnings During Model Update
    - ev\_0501 Format of Entries in a State Block
    - ev\_0511 Setting Return Value from Graphics
    - ev\_0521 Use of Return Value from Graphics
    - ev\_0531 Placement of the Default Transition
    - ev\_1001 Bitwise Stateflow Operators
    - ev\_1004 Simulink Model Appearance
    - ev\_1005 Modifications in Basic Blocks
    - ev\_1011 Pointers in Stateflow
    - ev\_1012 Event Broadcasts
- Blocks
  - Functions
  - Examples
  - Demos
- Release Notes

**Model Documentation**

**ID: Title**  
ev\_0009: Model Documentation

**Priority**  
Mandatory

**Scope**  
MAAB: hyl\_0112  
MISRA: modified AC GMG 009

**MATLAB**  
All Versions

**Static Check**  
Yes

**Prerequisites**  
No

**Description**

**Simulink Documentation**  
The following information shall be provided as free text annotations in the graphical workspace for every model, subsystem and library to aid understanding:

- Title on each page (Format: Arial, **bold**, 20 pt)
- Outline description (Format: Arial, standard, 10 pt)

The information shall be placed in upper area of the model to be easily identified.

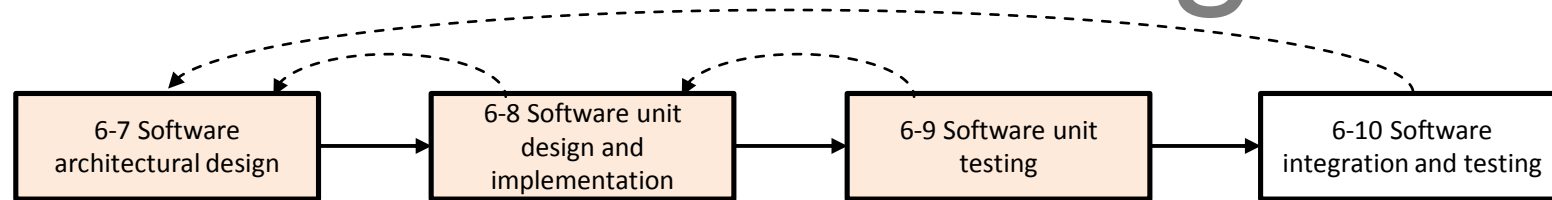
**Correct:**

Thermostat (TMS)
TMS turns the heating device ON or OFF based on the temperature Set-Point Error from temperature sensor and Ready Flag from the heating device.

# EZCHECK: Key Features

- Compliance Checking for safety-related guidelines
- Automated guideline-based model correction (Fix)
- Programmable scripting interface for check automation
- Intrusive exclusion handling for unit-based development
- Kontext-based exclusion documentation for review
- Automated report generation and relocation

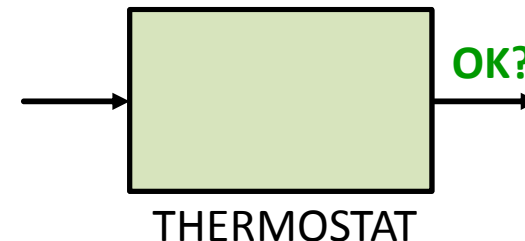
# Software Unit Testing



- ...demonstrate that the software units
- fulfill the software unit specifications
  - do not contain undesired functionality

# Testing Terminology

- Confusing terminology:  
Unit, Function, Module, Routine, Procedure,  
Component, Block, Part, Construct, Entity
- Unit is an atomic subsystem in Simulink
- Unit test should be:
  - Dynamic (executes test object)
  - Functional (requirements-based)
  - Proving expected functionality (ref == out)
  - Systematic with sufficient structural coverage



# Software Unit Testing

Table 10 — Methods for software unit testing

Methods		ASIL			
		A	B	C	D
1a	Requirements-based test <sup>a</sup>	++	++	++	++
1b	Interface test	++	++	++	++
1c	Fault injection test <sup>b</sup>	+	+	+	++
1d	Resource usage test <sup>c</sup>	+	+	+	++
1e	Back-to-back comparison test between model and code, if applicable <sup>d</sup>	+	+	++	++

Table 11 — Methods for deriving test cases for software unit testing

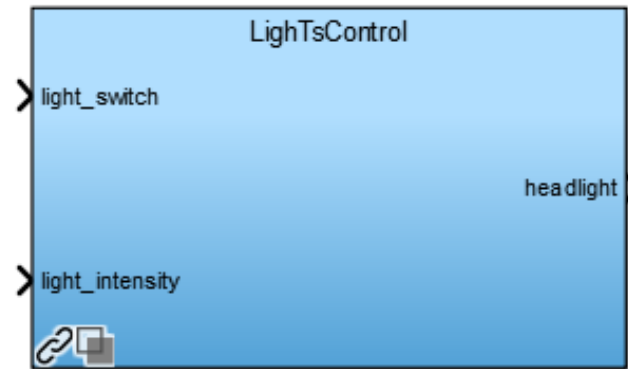
Methods		ASIL			
		A	B	C	D
1a	Analysis of requirements	++	++	++	++
1b	Generation and analysis of equivalence classes <sup>a</sup>	+	++	++	++
1c	Analysis of boundary values <sup>b</sup>	+	++	++	++
1d	Error guessing <sup>c</sup>	+	+	+	+

<sup>a</sup> Equivalence classes can be identified based on the division of inputs and outputs, such that a representative test value can be selected for each class.

<sup>b</sup> This method applies to interfaces, values approaching and crossing the boundaries and out of range values.

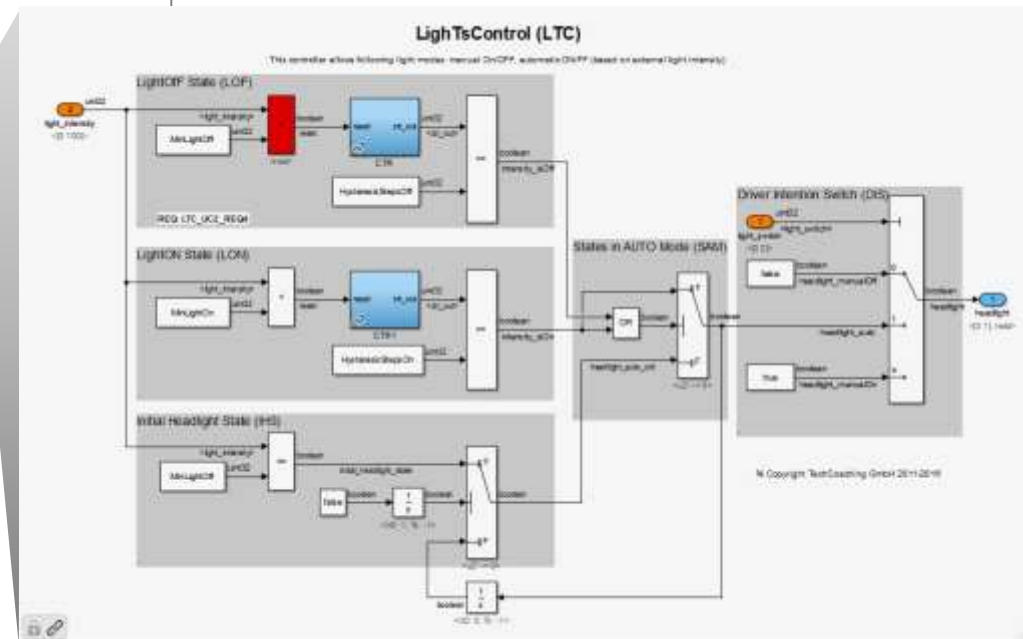
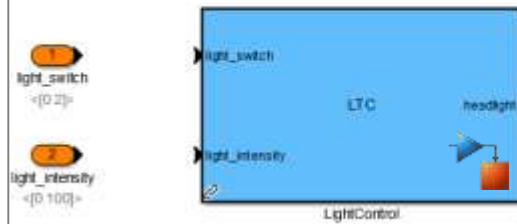
<sup>c</sup> Error guessing tests can be based on data collected through a “lessons learned” process and expert judgment.

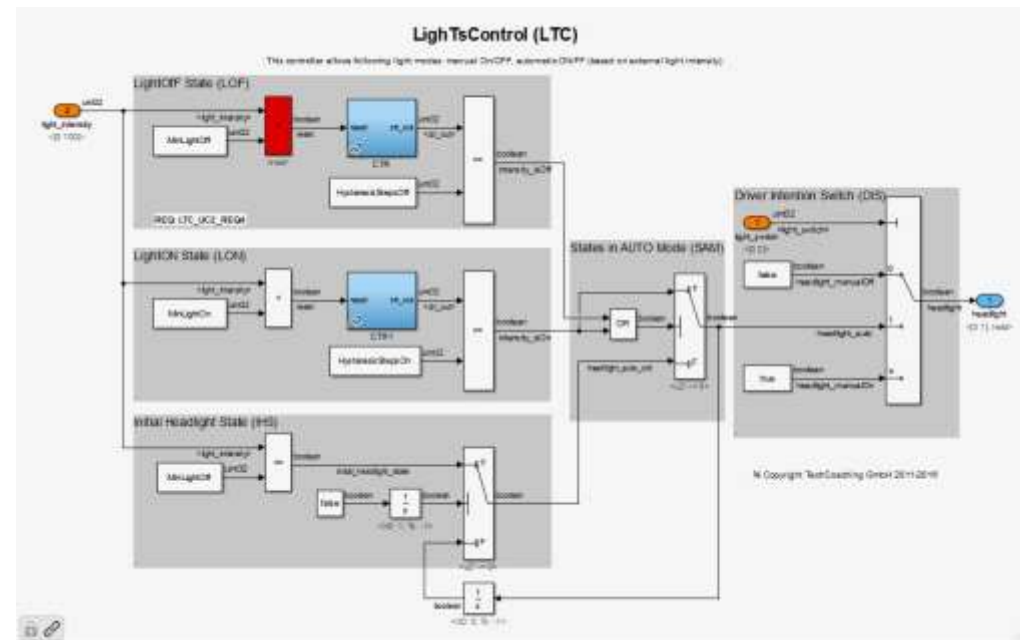
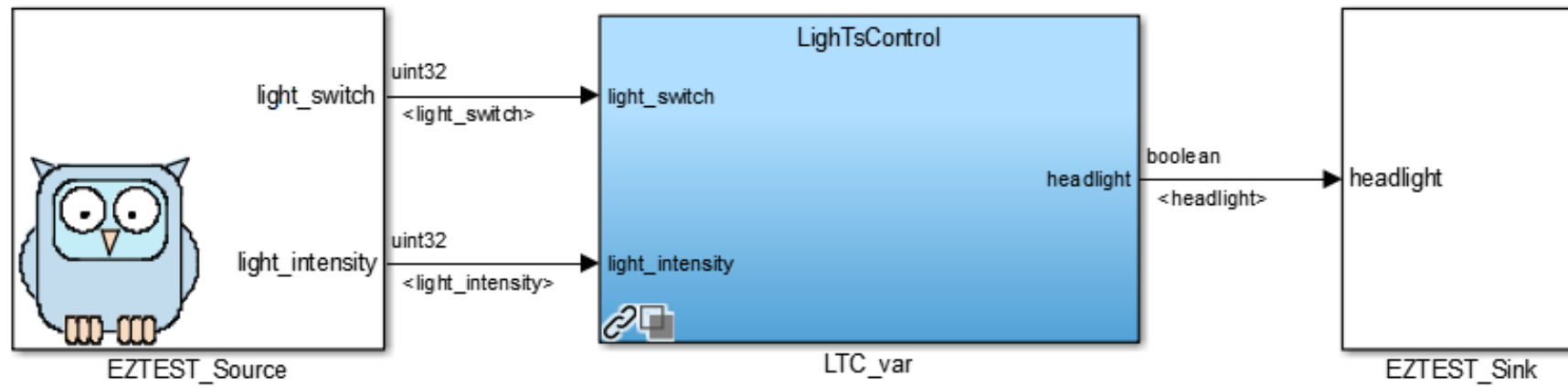
# Model Verification



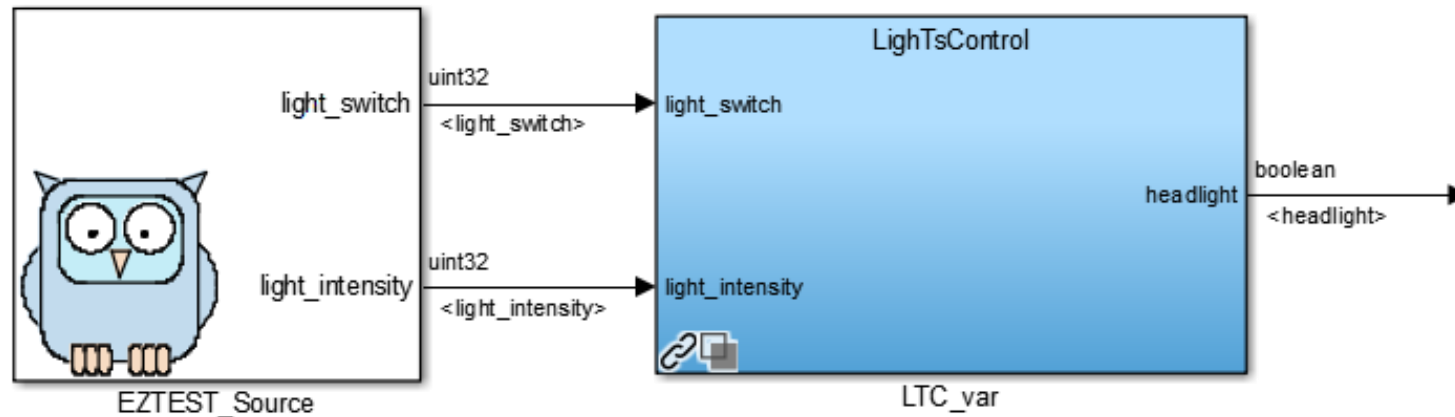
## Variants for LTC

Allow switching between different test modes









## Structural Coverage Report for LTC\_var

### Table of Contents

- [1. Analysis Information](#)
- [2. Tests](#)
- [3. Summary](#)
- [4. Details](#)

### Analysis Information

#### Model Information

Model version	1.256
Author	TechDirector
Last saved	Thu May 05 17:16:25 2016

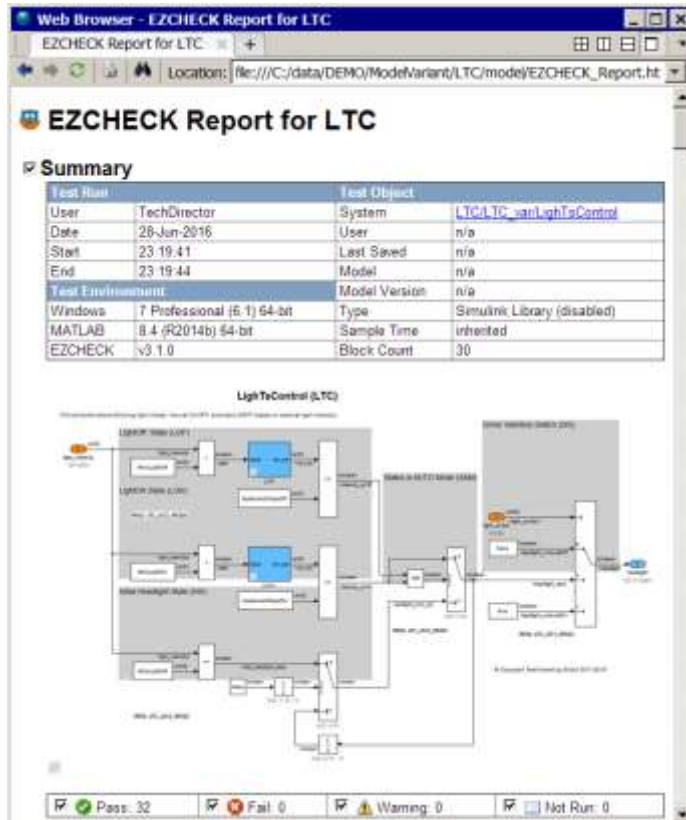
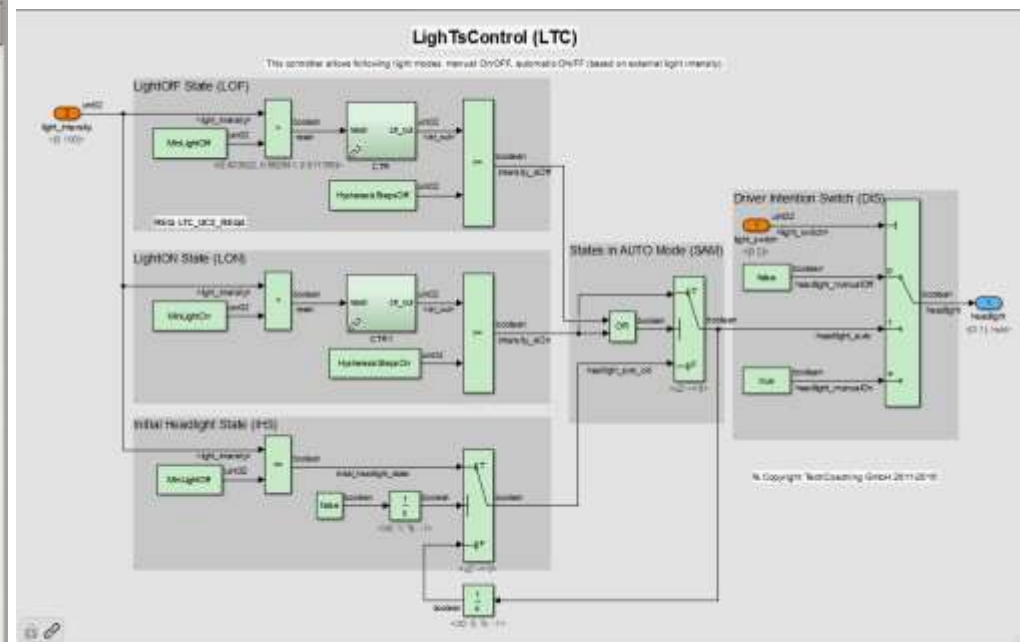
#### Simulation Optimization Options

Inline parameters	on
Block reduction	forced off
Conditional branch optimization	on

#### Coverage Options

Analyzed model	LTC/LTC_var
Logic block short circuiting	off

### Tests



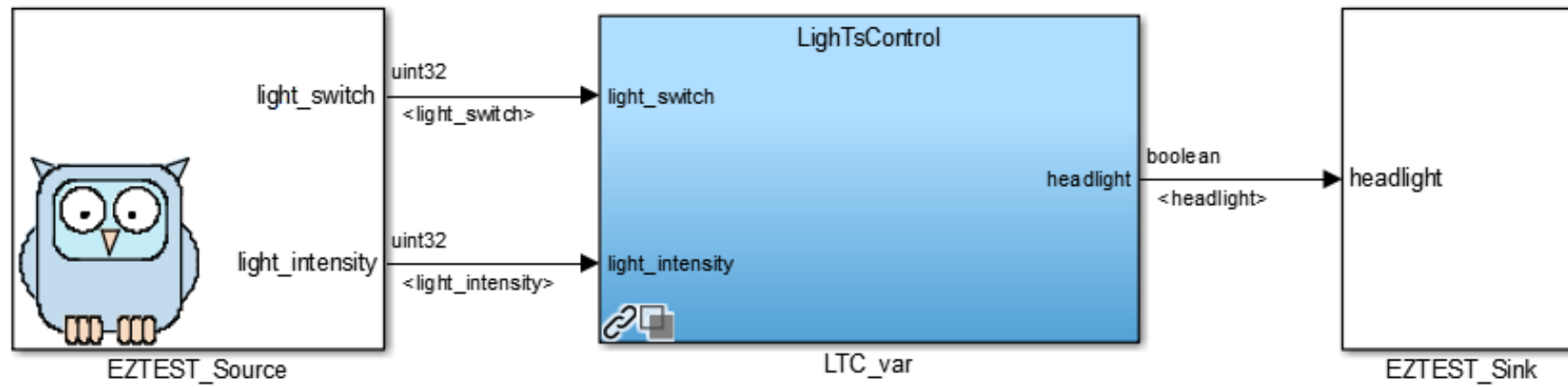
# Regression Tests

- Repetition of all tests after unit modification
- Assure absence of negative side effects
- Can be done manually or automatically
- Basis for back-to-back tests
- Document test data and results

# EZTEST: Key Features

- Enables requirements-based and back-to-back testing
- Easy generation of test harness with unit interface
- Easy test case generation, configuration and execution
- Pass-fail criteria based on reference & tolerance values
- Automatic assessments and graphical representation
- Step-by-step debugger with access to internal model data
- Integrated static model analysis and reporting (SLVV)
- Integrated structural test coverage and reporting (SLVV)
- Generation of safety compliant verification report
- Import/export of test data for MDF, MAT, XML, etc.
- Technical documentation with examples and demos
- Safety manual with tool classification by TÜV-SÜD

# Code Generation



```
LightTsControl.c
26 /* Model step function */
27 void STEP_LighTsControl(const uint32_T light_switch, const u
28 light_intensity, boolean_T *headlight)
29 {
30     boolean_T rtb_intensity_isOn;
31
32     /* Outputs for Atomic SubSystem: '<Root>/LightTsControl' */
33     /* RelationalOperator: '<S1>/RelationalOperator1' incorporates
34
35     rtb_intensity_isOn = (boolean_T)((int32_T)(LightTsControl_Un
36     HysteresisStepsOn));
37
38     /* Switch: '<S1>/Switch1' incorporates:
39
40     if ((LightTsControl_UnitDelay1 >= HysteresisStepsOff) || ((
41     rtb_intensity_isOn)) {
42         LightTsControl_UnitDelay = rtb_intensity_isOn;
43     } else {
44         if ((int32_T)LightTsControl_UnitDelay3) {
45             /* Switch: '<S1>/Switch' incorporates:
46
47             LightTsControl_UnitDelay = (light_intensity <= MinLight
48         }
49     }
50 }
51
52 /* End of Switch: '<S1>/Switch1' */
53
54 /* MultiplexSwitch: '<S1>/MultiplexSwitch' incorporates:
55
56 switch (light_switch) {
57     case 0:
58         /* Output: '<Root>/headlight' incorporates:
59
60         *headlight = (uint8_T>false;
61     break;
```

# Back-to-Back Comparison Tests

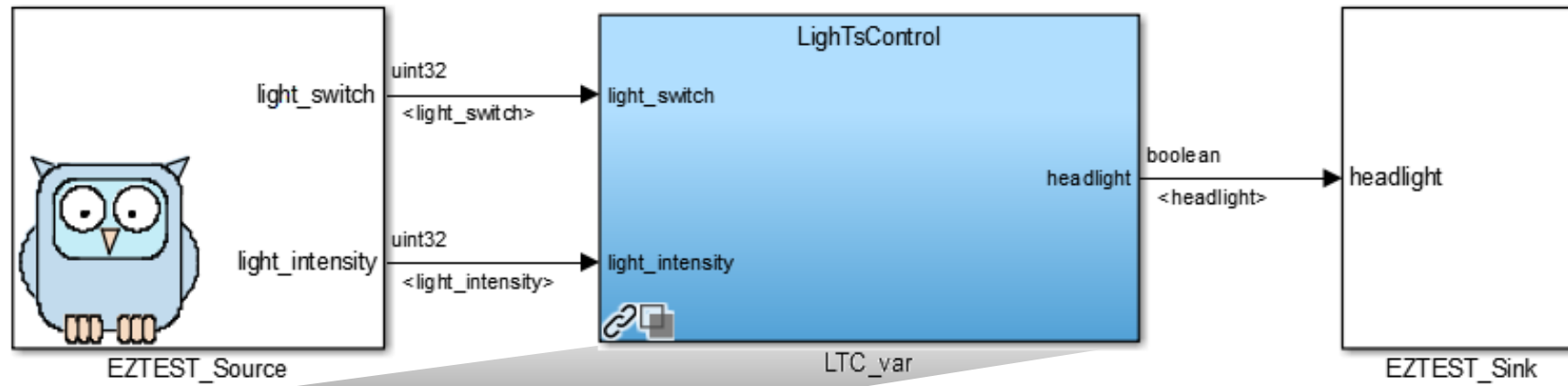
Table 10 — Methods for software unit testing

Methods		ASIL			
		A	B	C	D
1a	Requirements-based test <sup>a</sup>	++	++	++	++
1b	Interface test	++	++	++	++
1c	Fault injection test <sup>b</sup>	+	+	+	++
1d	Resource usage test <sup>c</sup>	+	+	+	++
1e	Back-to-back comparison test between model and code, if applicable <sup>d</sup>	+	+	++	++

© INTERNATIONAL STANDARD ISO/DIS 26262-6

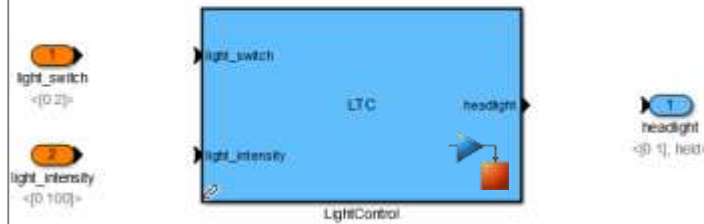
- **Software in the loop**
  - A test level where the real software (C sources) is used and tested in a simulated environment (host compiler)
- **Processor in the loop**
  - A test level where the real software (C sources) is used and tested on target hardware (target compiler)
- **Reuse existing regression tests**

# Code Integration



### Variants for LTC

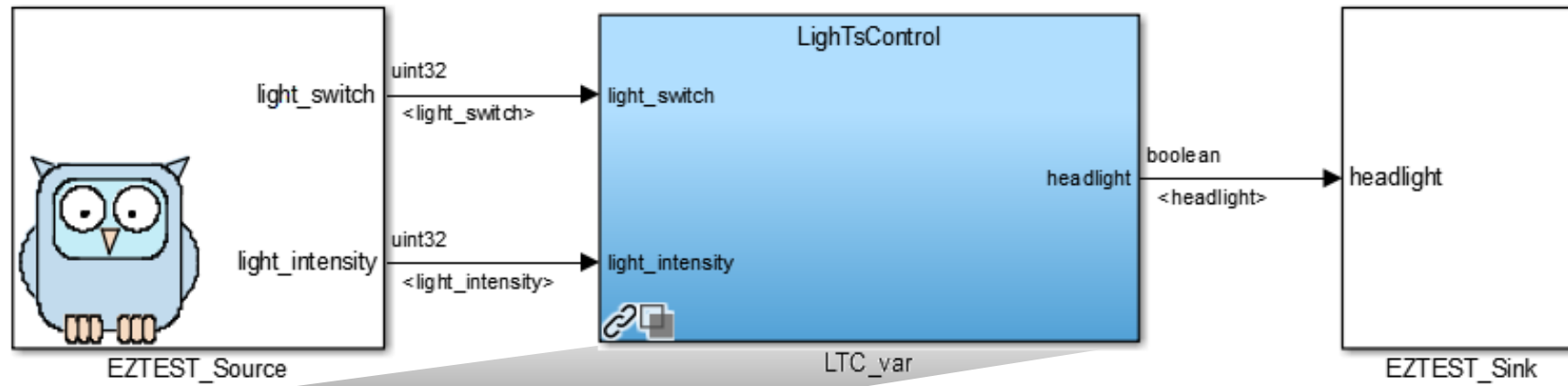
Allow switching between different test modes



```

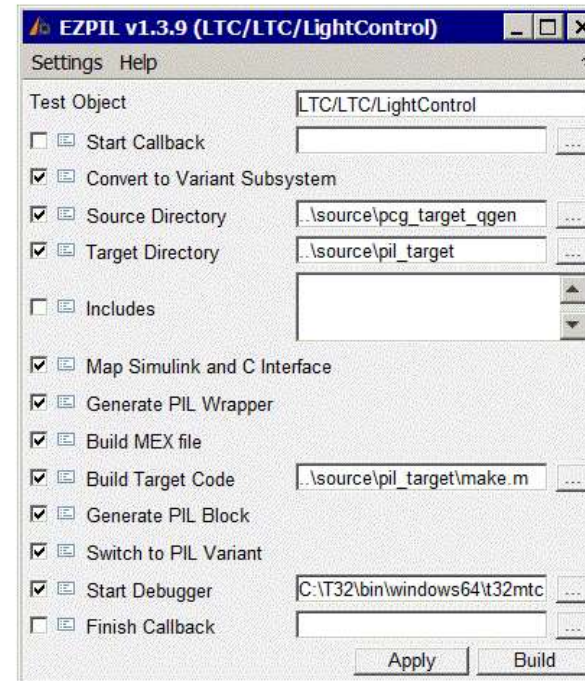
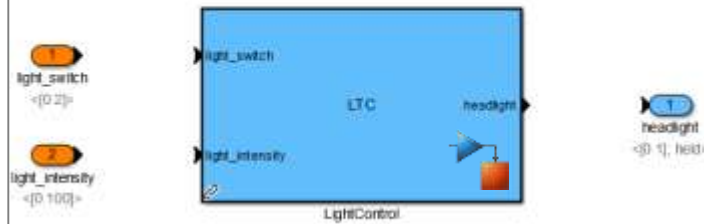
25  /* Model step function */
26  void STEP_LightTsControl(const uint32_T light_switch, const u
27  light_intensity, boolean_T *headlight)
28  {
29  {
30  boolean_T rtb_intensity_isOn;
31
32  /* Outputs for Atomic SubSystem: '<Root>/LightTsControl' */
33  /* RelationalOperator: '<S1>/RelationalOperator1' incorporates
34  rtb_intensity_isOn = (boolean_T)((int32_T)(LightTsControl_U
35  HysteresisStepsOn));
36
37  /* Switch: '<S1>/Switch1' incorporates:
38  if ((LightTsControl_UnitDelay1 >= HysteresisStepsOff) || ((
39  rtb_intensity_isOn)) {
40  LightTsControl_UnitDelay = rtb_intensity_isOn;
41  } else {
42  if ((int32_T)LightTsControl_UnitDelay3) {
43  /* Switch: '<S1>/Switch' incorporates:
44  LightTsControl_UnitDelay = (light_intensity <= MinLight
45  }
46  }
47
48  /* End of Switch: '<S1>/Switch1' */
49
50  /* MultiPortSwitch: '<S1>/MultiportSwitch' incorporates:
51  switch (light_switch) {
52  case 0:
53  /* Output: '<Root>/headlight' incorporates:
54  *headlight = (uint8_T)false;
55  break;
  
```

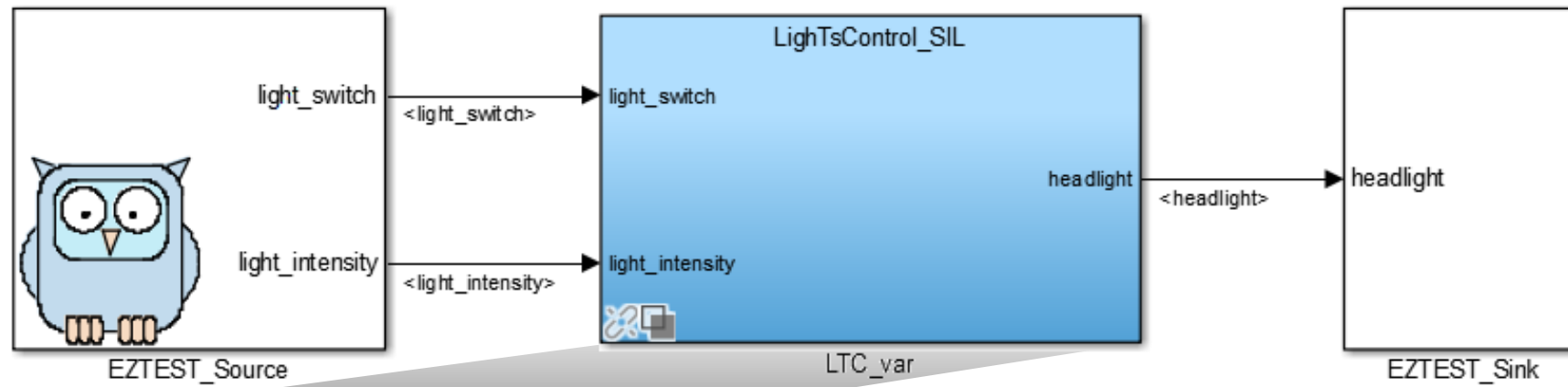




### Variants for LTC

Allow switching between different test modes





### Variants for LTC

Allow switching between different test modes

1  
light\_switch  
<[0 2]>

2  
light\_intensity  
<[0 100]>



### EZPIL Variant

Automatically generated test mode

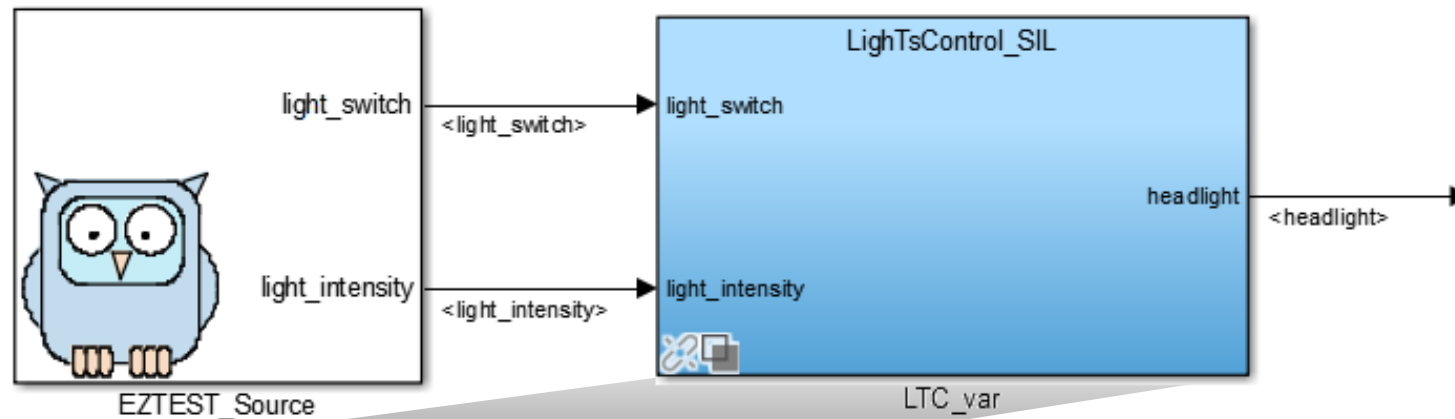
3  
light\_switch  
<[0 2]>

2  
light\_intensity  
<[0 100]>



1  
headlight  
<[0 1], held>

# Code Verification



**EZTEST v3.7.3 (LTC/LTC\_var)**

Test Specification: **LT\_LTC\_EZtest.xlsx**

Test Cases: **Driver Intention Switch**

Verification & Validation:

- ☐ Static Analysis Report
- ☒ Structural Coverage Report
- ☒ Decision
- ☒ MCDC
- ☒ Boundary Values Analysis
- ☐ Verification Report
- ☐ Debug
- ☐ Real-Time

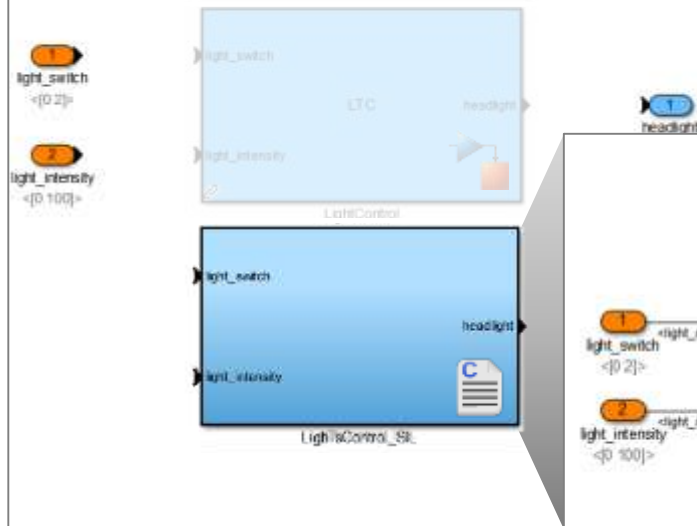
Running test 1 of 8

	A	B	C	D	E	F	G	H	I	J
		light_switch	light_intensity	headlight_ref	headlight	MinLightOff	MinLightOn	HysteresisStepsOff	HysteresisStepsOn	
1										
2	0	0	0	0	70	60	2	2	2	REQ: LTC_UC1_REQ1
3	0.02									There are three switch positions to ex
4	0.04									- Headlight is OFF;
5	0.06									- Headlight is AUTO (ON if low light int
6	0.08									- Headlight is ON.
7	0.1									
8	0.12	2		1						ON
9	0.14									
10	0.16									
11	0.18	0		0						OFF
12	0.2									
13	0.22									
14	0.24	1		1						AUTO
15	0.26									
16	0.28									
17	500									
18										
19										
20										
21										
22										
23										
24										
25										
26										
27										

Driver Intention Switch

### Variants for LTC

Allow switching between different test modes

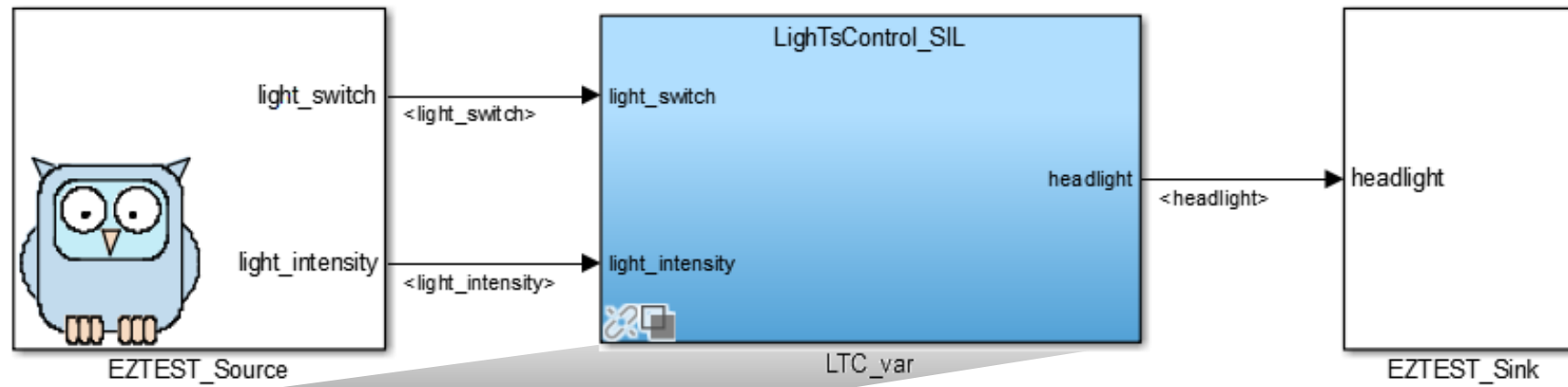


### EZPIL Variant

Automatically generated test mode

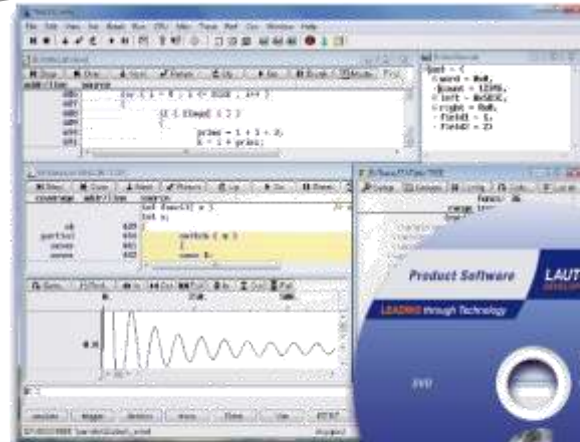
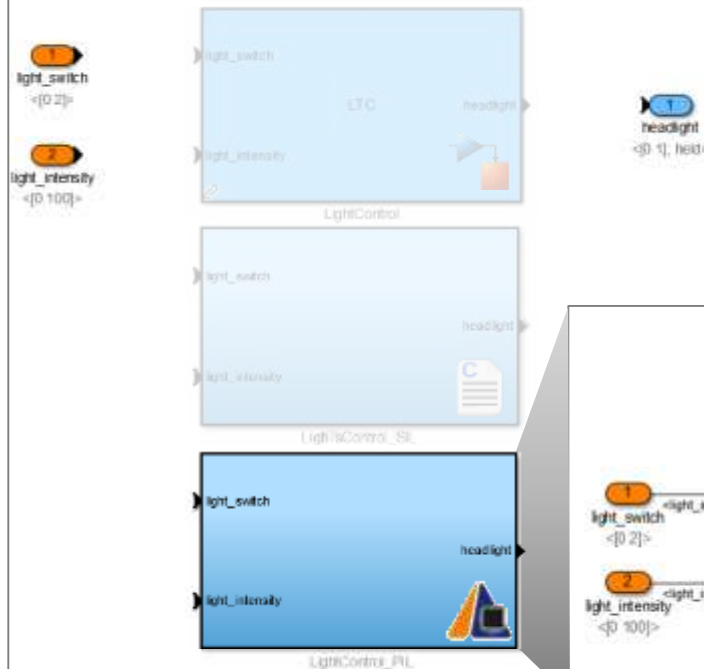


# Target Emulation



### Variants for LTC

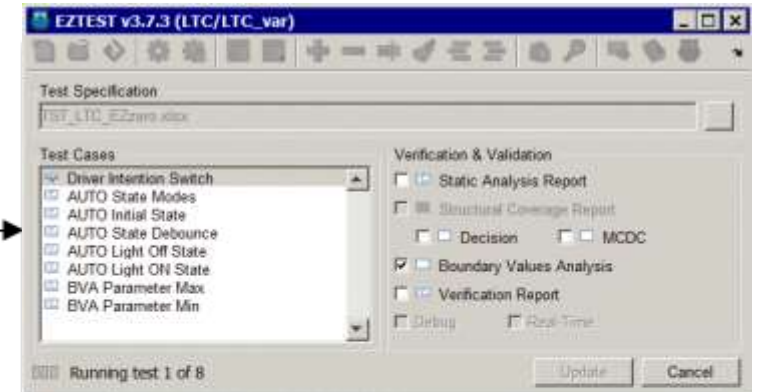
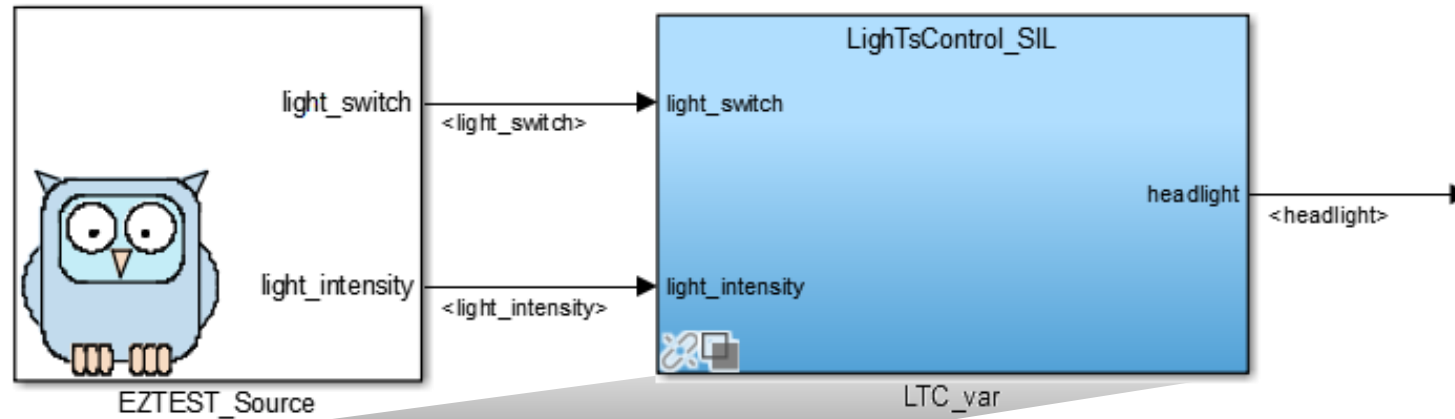
Allow switching between different test modes



### EZPIL Variant

Automatically generated test mode





	A	B	C	D	E	F	G	H	I	
		light_switch	light_intensity	headlight_ref	headlight	MinLightOff	MinLightOn	HysteresisStepsOff	HysteresisStepsOn	
1										
2	0	0	0	0	70	60	2	2	REQ: LTC_UC1_REQ1	
3	0.02									There are three switch positions to ex
4	0.04									- Headlight is OFF;
5	0.06									- Headlight is AUTO (ON if low light int
6	0.08									- Headlight is ON.
7	0.1									
8	0.12	2		1						ON
9	0.14									
10	0.16									
11	0.18	0		0						OFF
12	0.2									
13	0.22									
14	0.24	1		1						AUTO
15	0.26									
16	0.28									
17	500									
18										
19										
20										
21										
22										
23										
24										
25										
26										
27										

Driver Intention Switch

## Variants for LTC

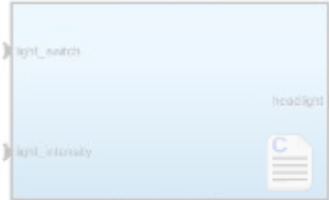
Allow switching between different test modes

light\_switch  
<[0 2]>

light\_intensity  
<[0 100]>



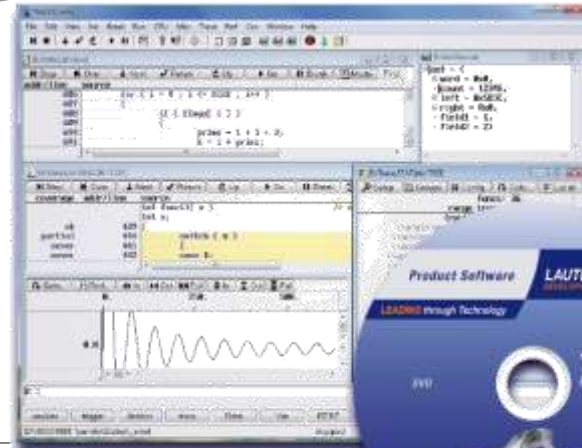
LightControl



LightControl\_SIL

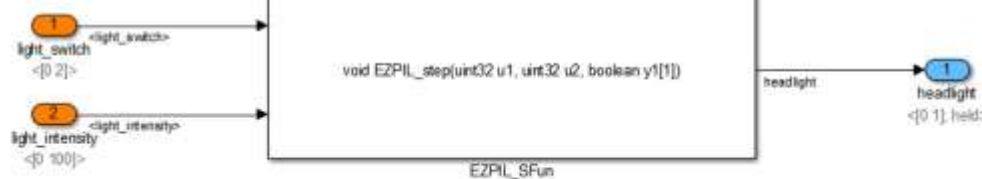


LightControl\_PIL



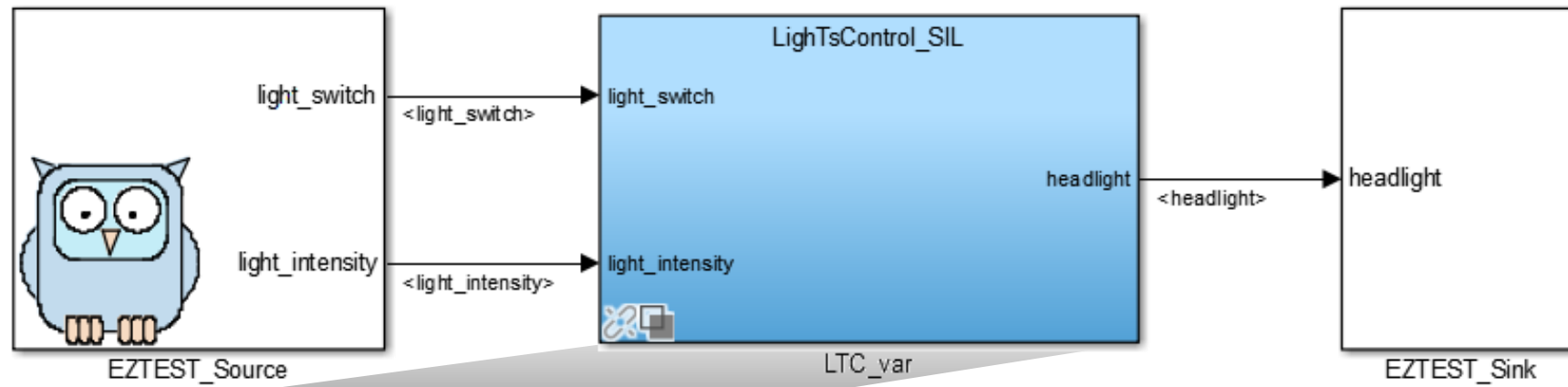
## EZPIL Variant

Automatically generated test mode



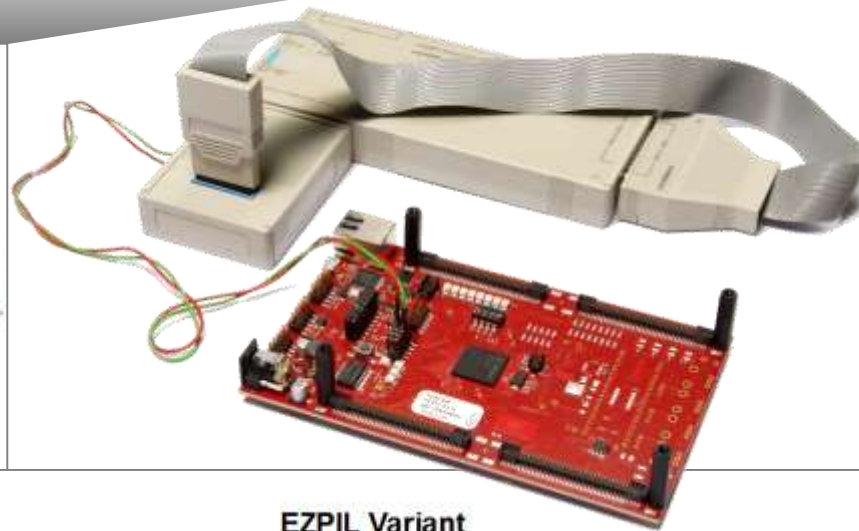
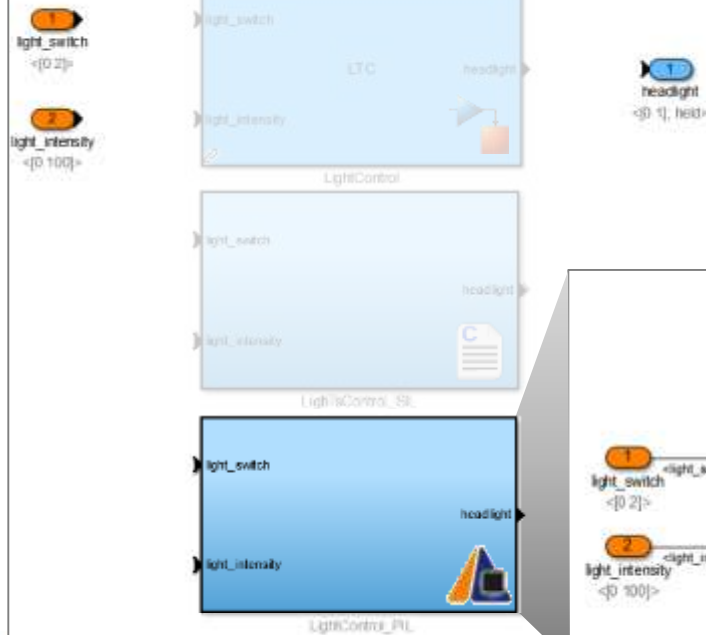
# On Target Verification





### Variants for LTC

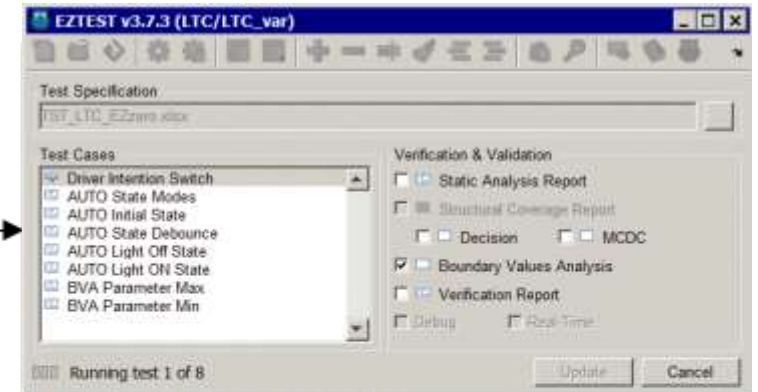
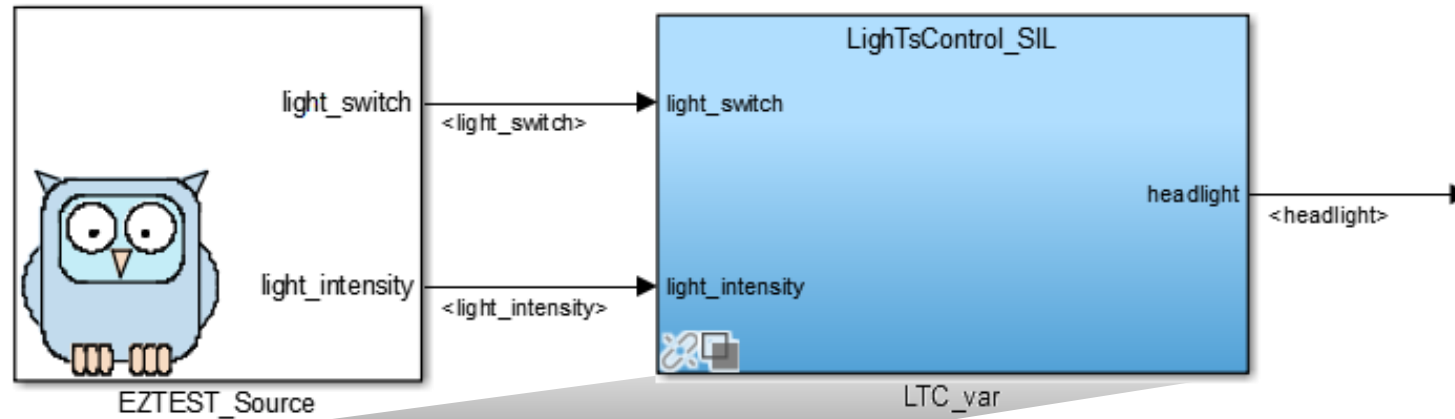
Allow switching between different test modes



### EZPIL Variant

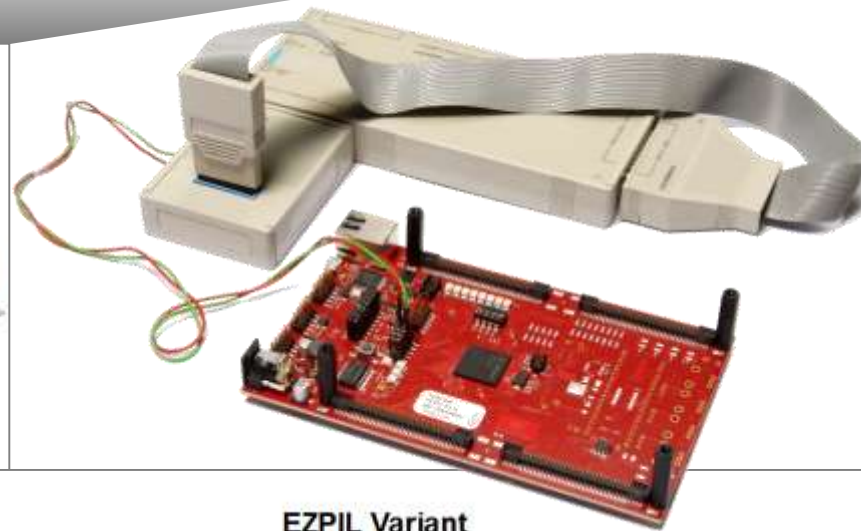
Automatically generated test mode





	A	B	C	D	E	F	G	H	I	
		light_switch	light_intensity	headlight_ref	headlight	MinLightOff	MinLightOn	HysteresisStepsOff	HysteresisStepsOn	
1										
2	0	0	0	0	70	60	2	2	REQ: LTC_UC1_REQ1	
3	0.02								There are three switch positions to ex	
4	0.04								- Headlight is OFF;	
5	0.06								- Headlight is AUTO (ON if low light int	
6	0.08								- Headlight is ON.	
7	0.1									
8	0.12	2		1					ON	
9	0.14									
10	0.16									
11	0.18	0		0					OFF	
12	0.2									
13	0.22									
14	0.24	1		1					AUTO	
15	0.26									
16	0.28									
17	500									
18										
19										
20										
21										
22										
23										
24										
25										
26										
27										

Driver Intention Switch



## Variants for LTC

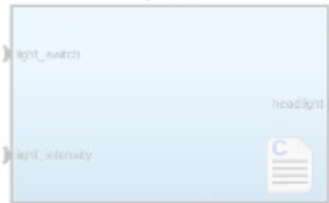
Allow switching between different test modes

1 light\_switch  
<[0 2]>

2 light\_intensity  
<[0 100]>



LightControl



LightControl\_SIL



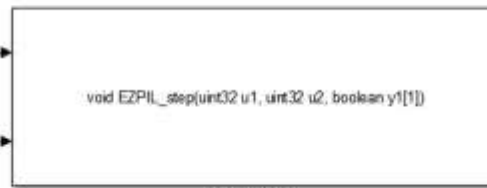
LightControl\_PIL

## EZPIL Variant

Automatically generated test mode

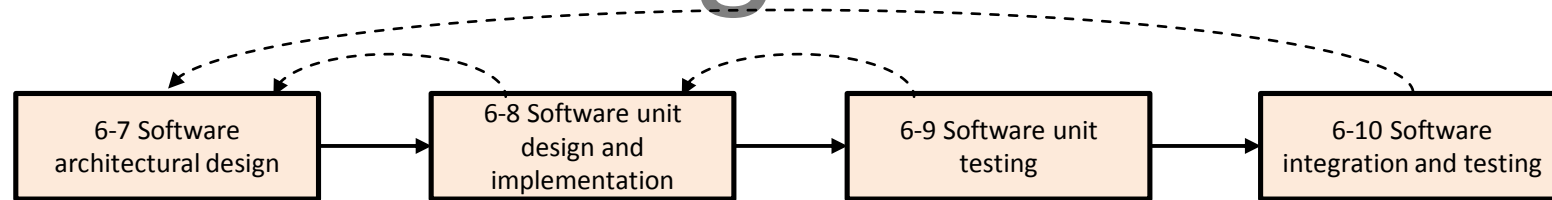
1 light\_switch  
<[0 2]>

2 light\_intensity  
<[0 100]>



EZPIL\_SFun

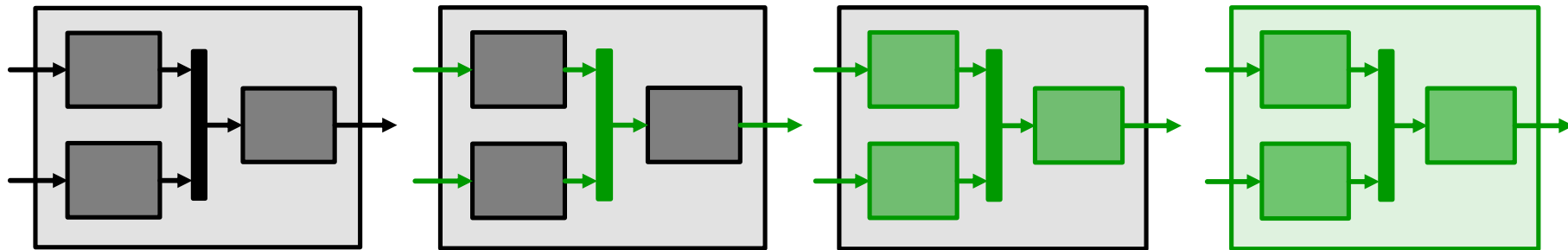
# Software Integration & Testing



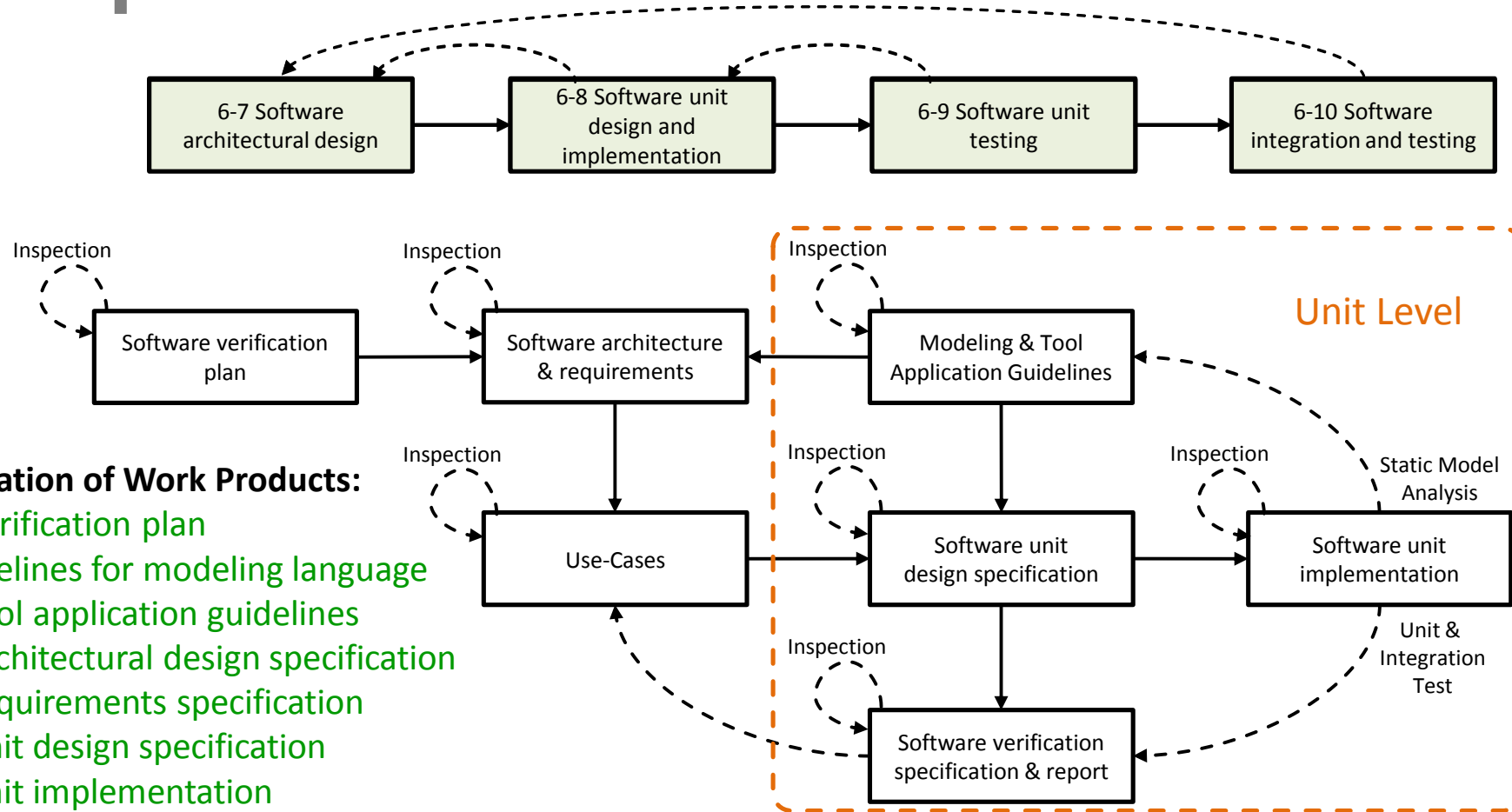
... performed to expose faults in the interfaces and in the interaction between integrated components

# Software Integration Strategy

- Guaranty well-defined model architecture
- Guaranty top-down interface compliance
- Guaranty unit quality prior to integration
- Incremental high-quality unit integration
- Reuse functional and interface tests



# Inspection of Work Products

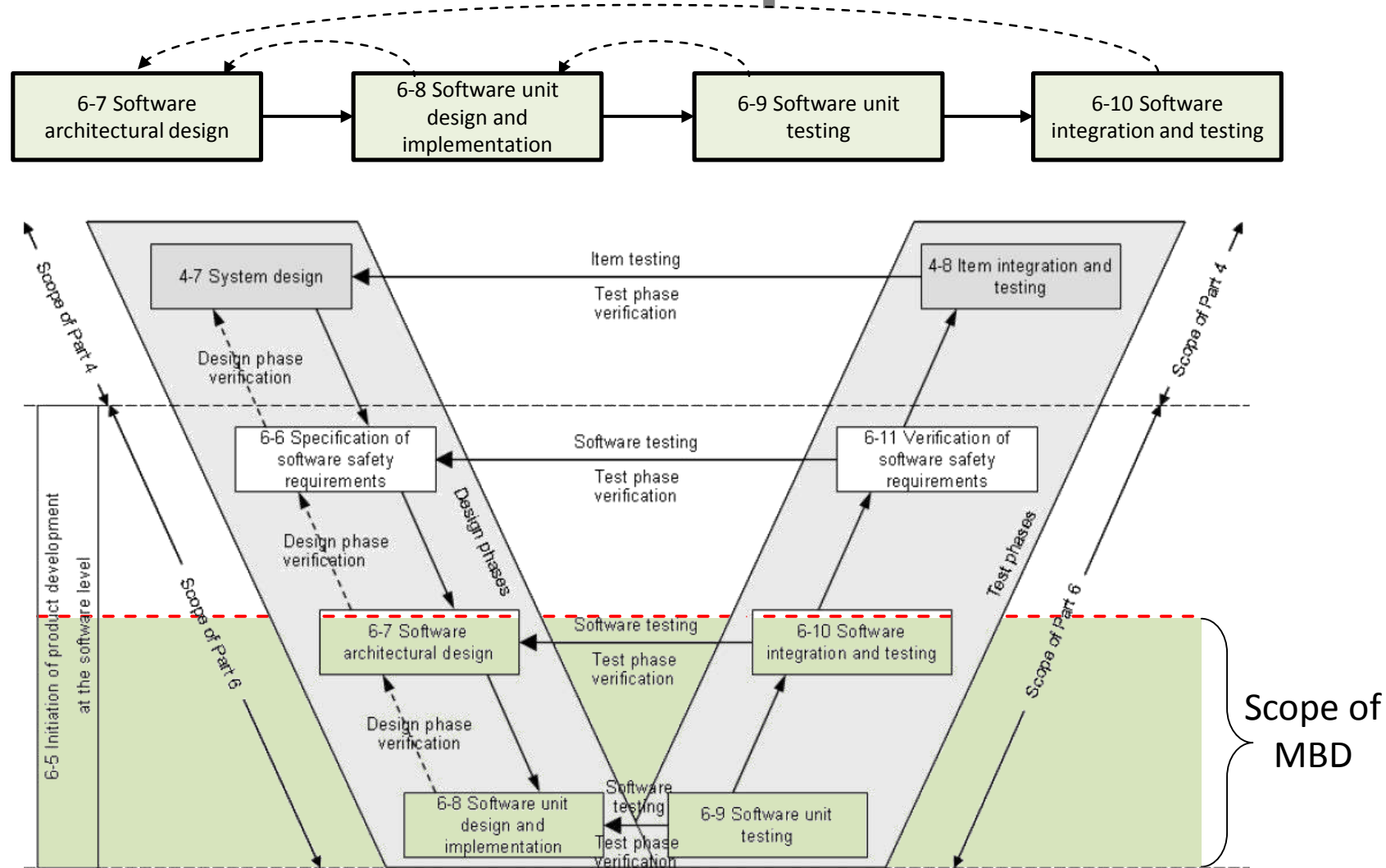


## Final Confirmation of Work Products:

- Software verification plan
- Design guidelines for modeling language
- Software tool application guidelines
- Software architectural design specification
- Software requirements specification
- Software unit design specification
- Software unit implementation
- **Software verification specification**
- **Software verification report**

~80% of development time is spent on unit-level verification.

# Software Development Model



# Summary

- Software architectural design
- Software unit design and implementation
- Software unit testing
- Software integration and testing