Increasing Design Confidence

Model and Code Verification

The Cost of Failure…

Ariane 5

$7,500,000,000

Rocket & payload lost

The Cost of Failure…

USS Yorktown

0 Knots

Top speed

The Cost of Failure…

Therac-25

6 Casualties

due to radiation overdose

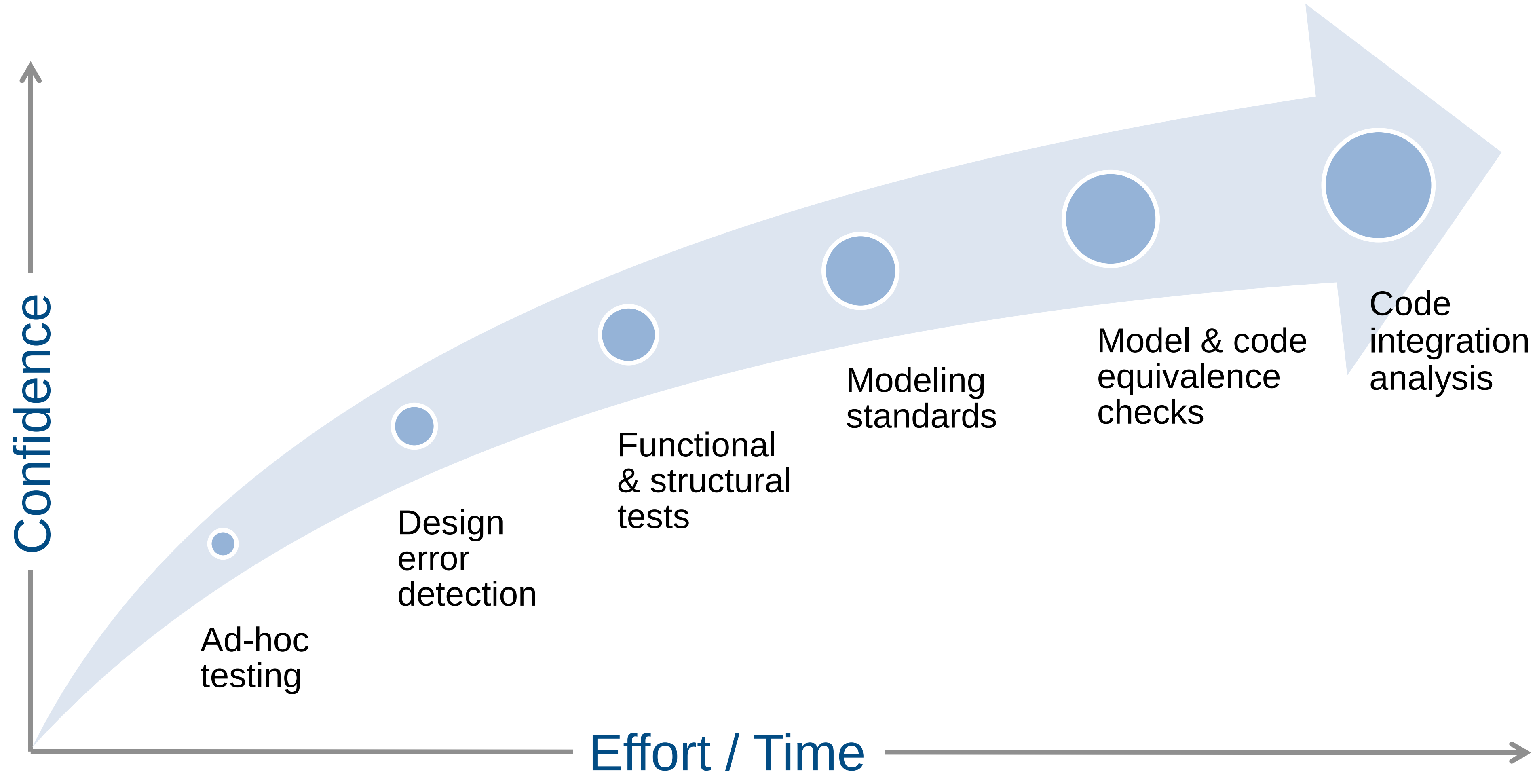
Motivation

It is easier and less expensive to fix design errors early in the process when they happen.

Model-Based Design enables:

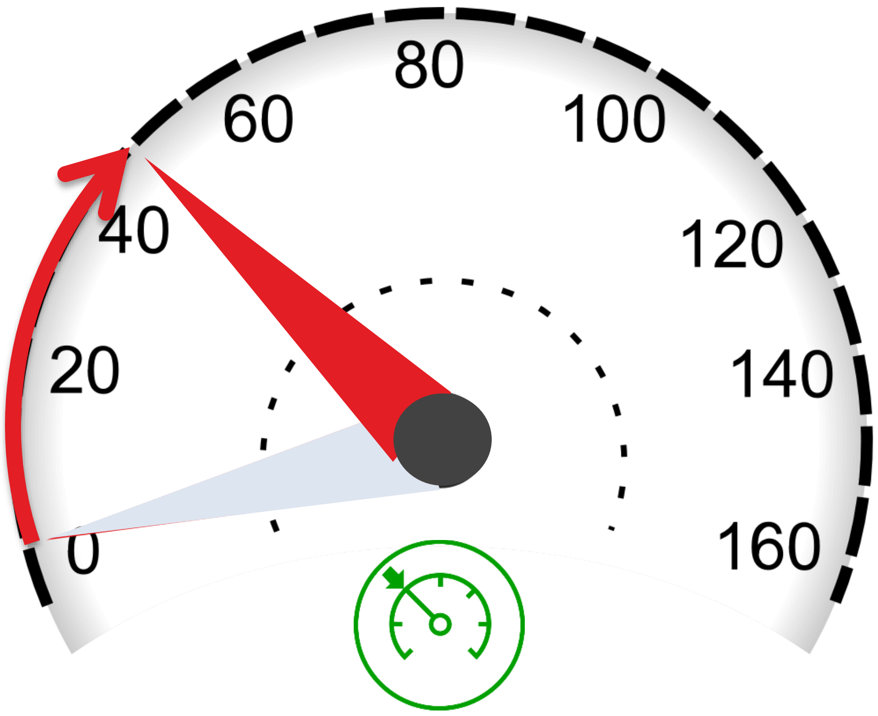
1. *Early testing to increase confidence in your design*
2. *Delivery of higher quality software throughout the workflow*

Gaining Confidence in our Design



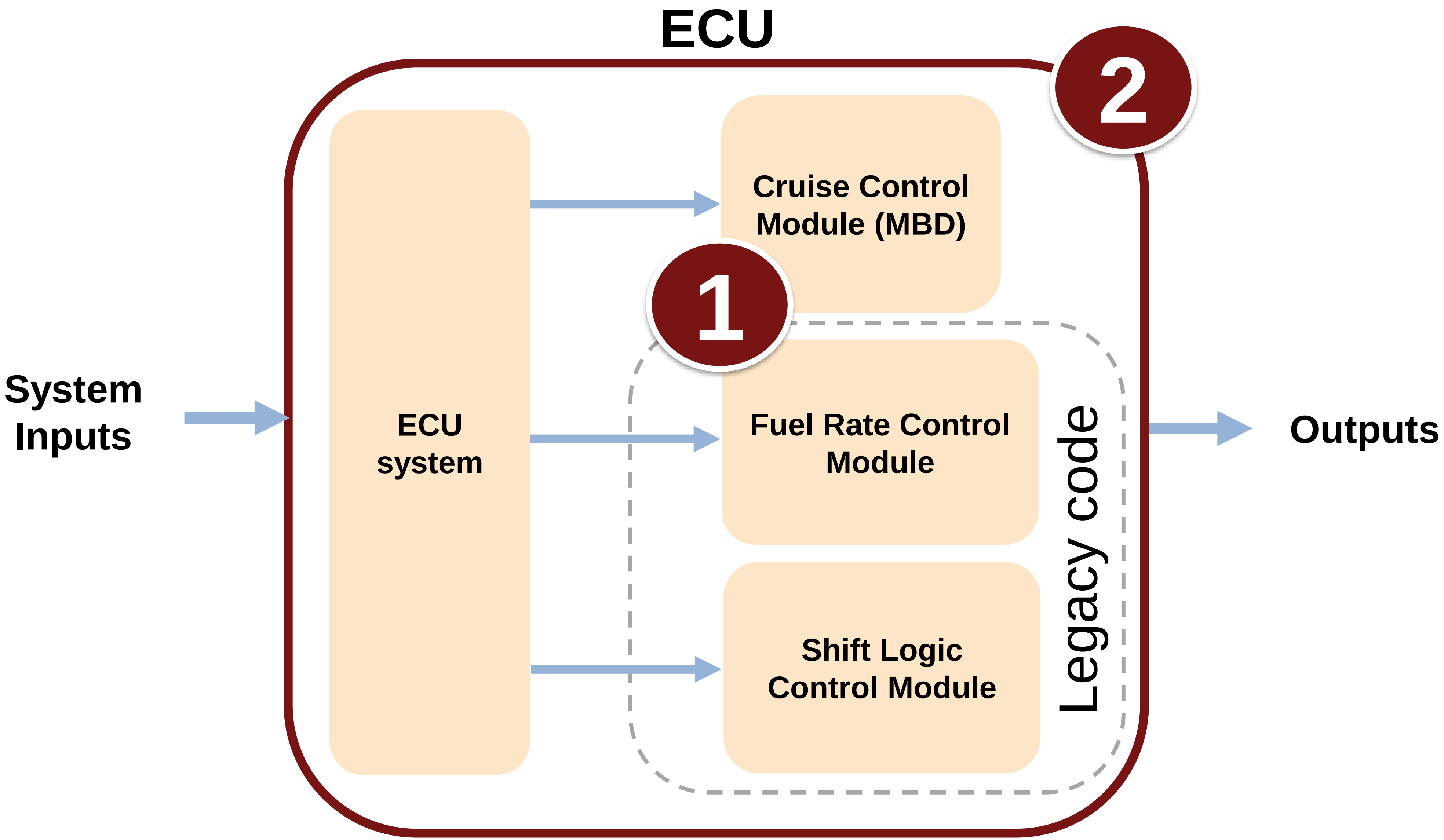
Application: Cruise Control

Control speed according to setpoint

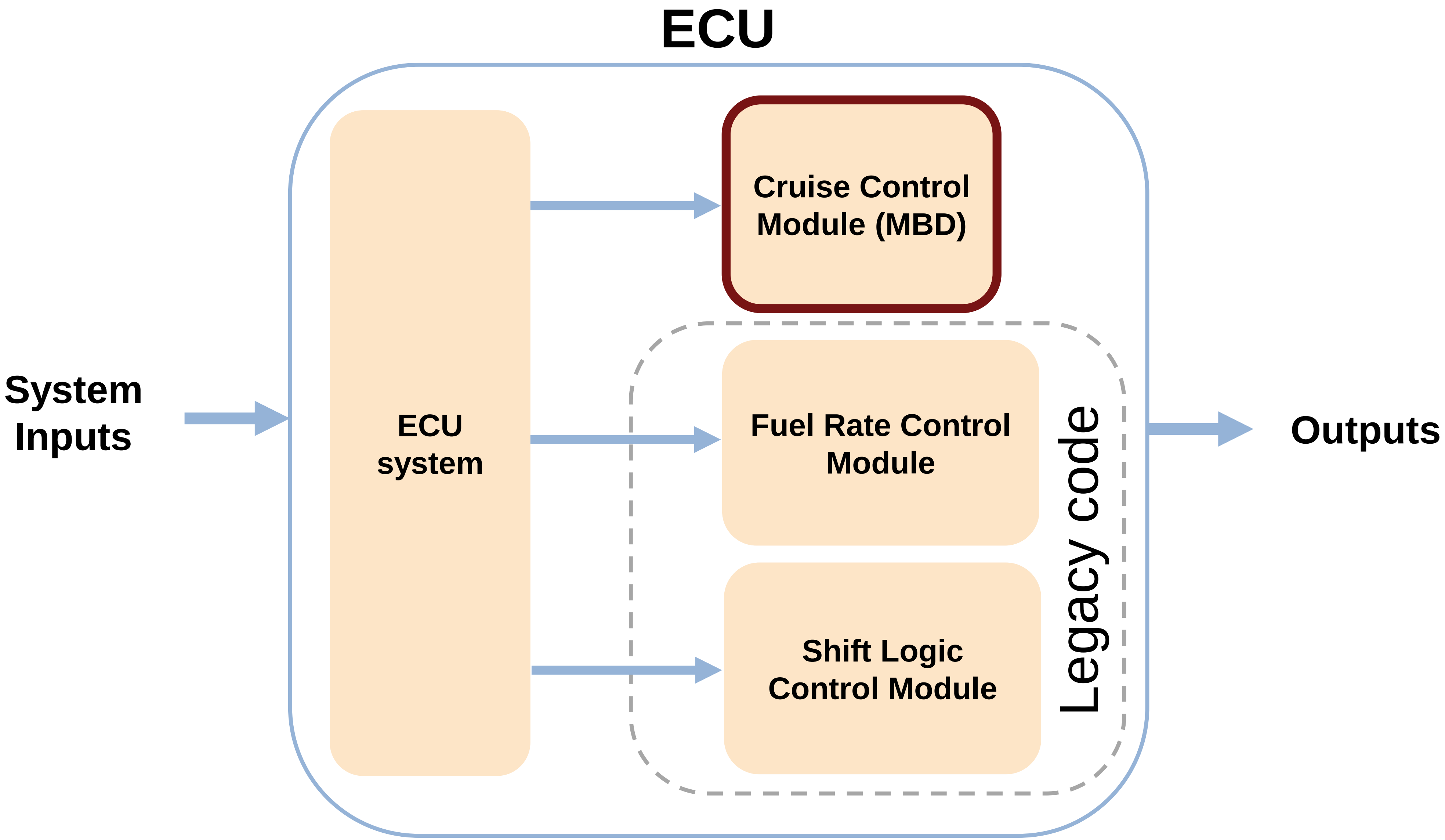


50 km/h

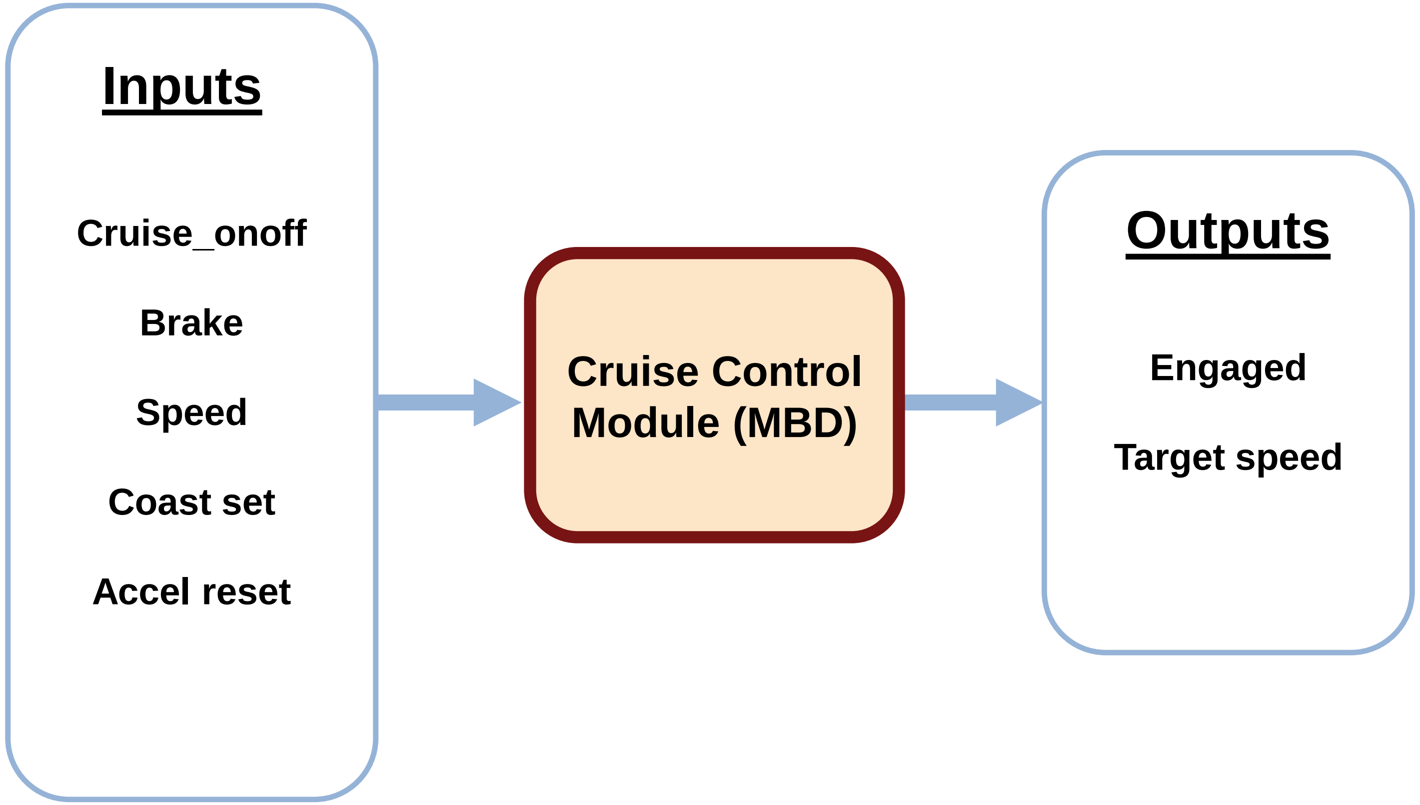
Application: Cruise Control



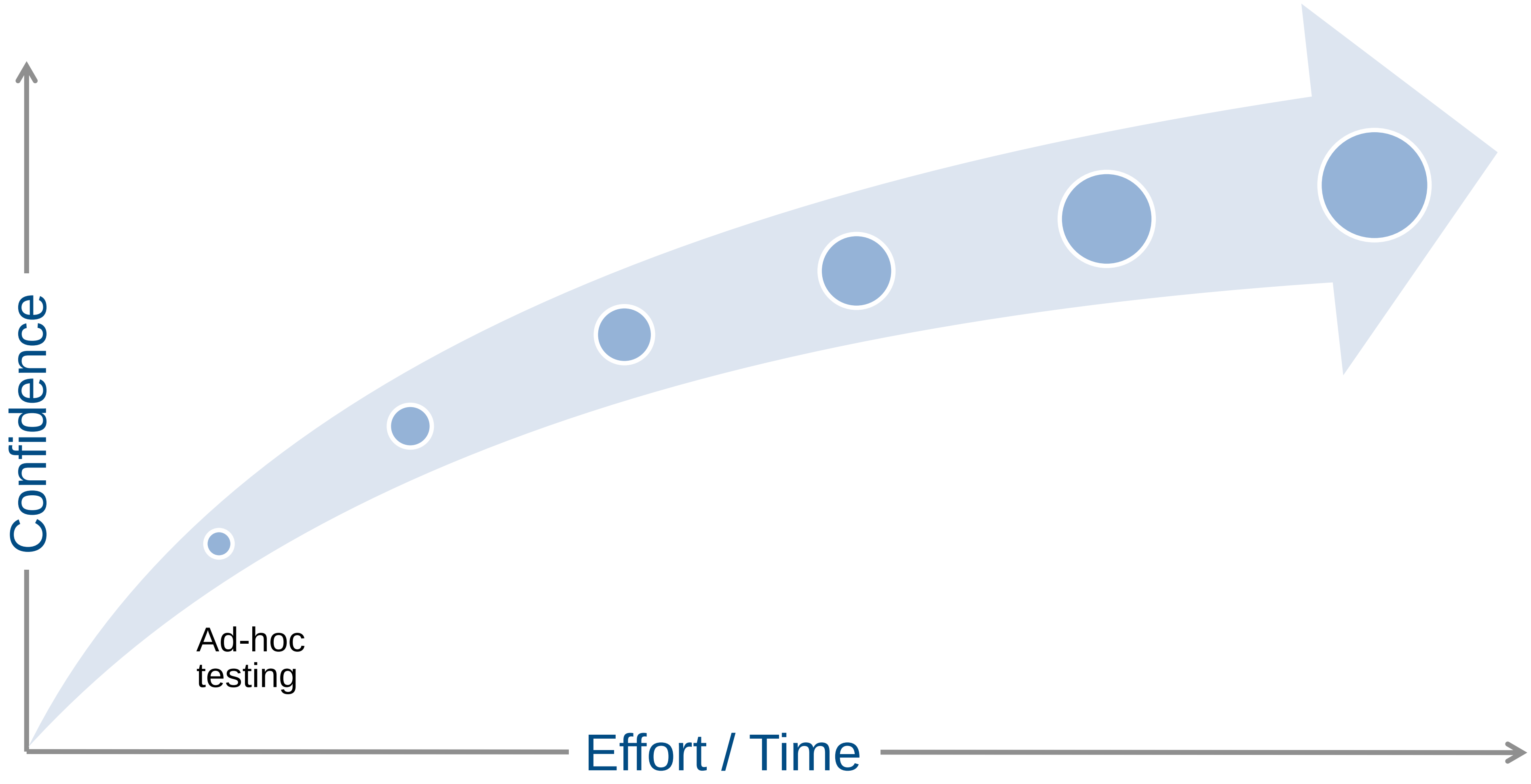
Application: Cruise Control



Application: Cruise Control

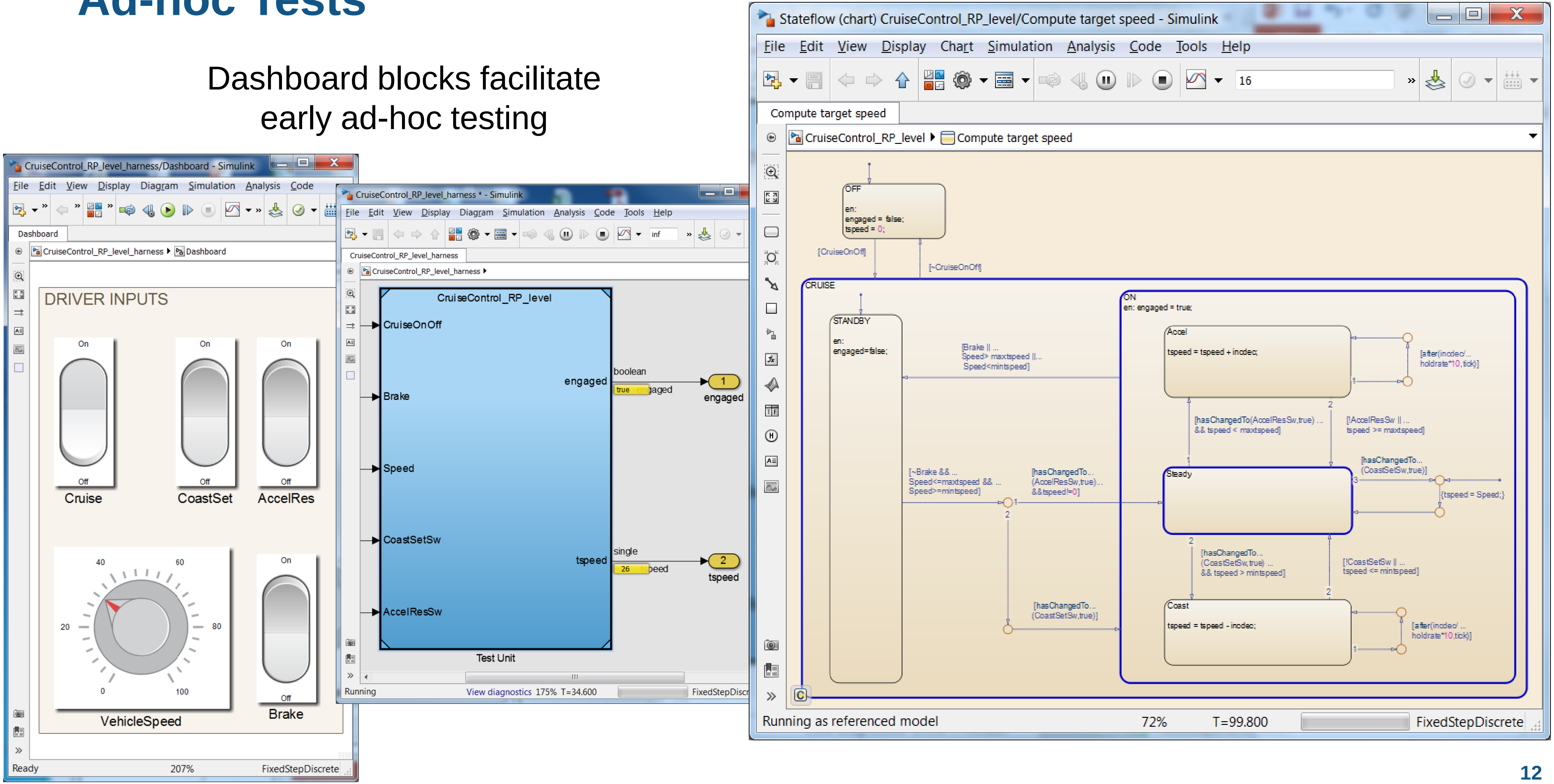


Gaining Confidence in our Design

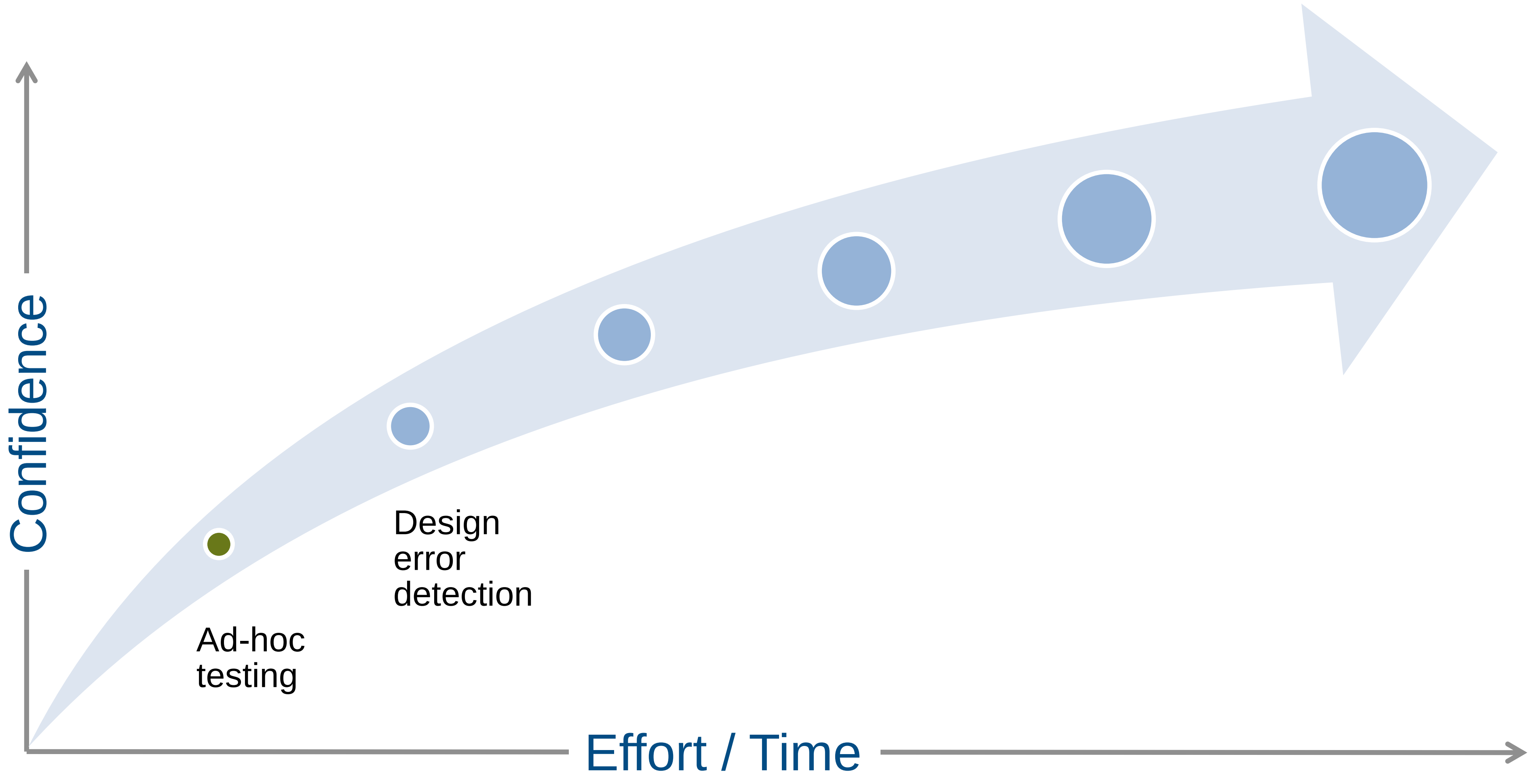


Ad-hoc Tests

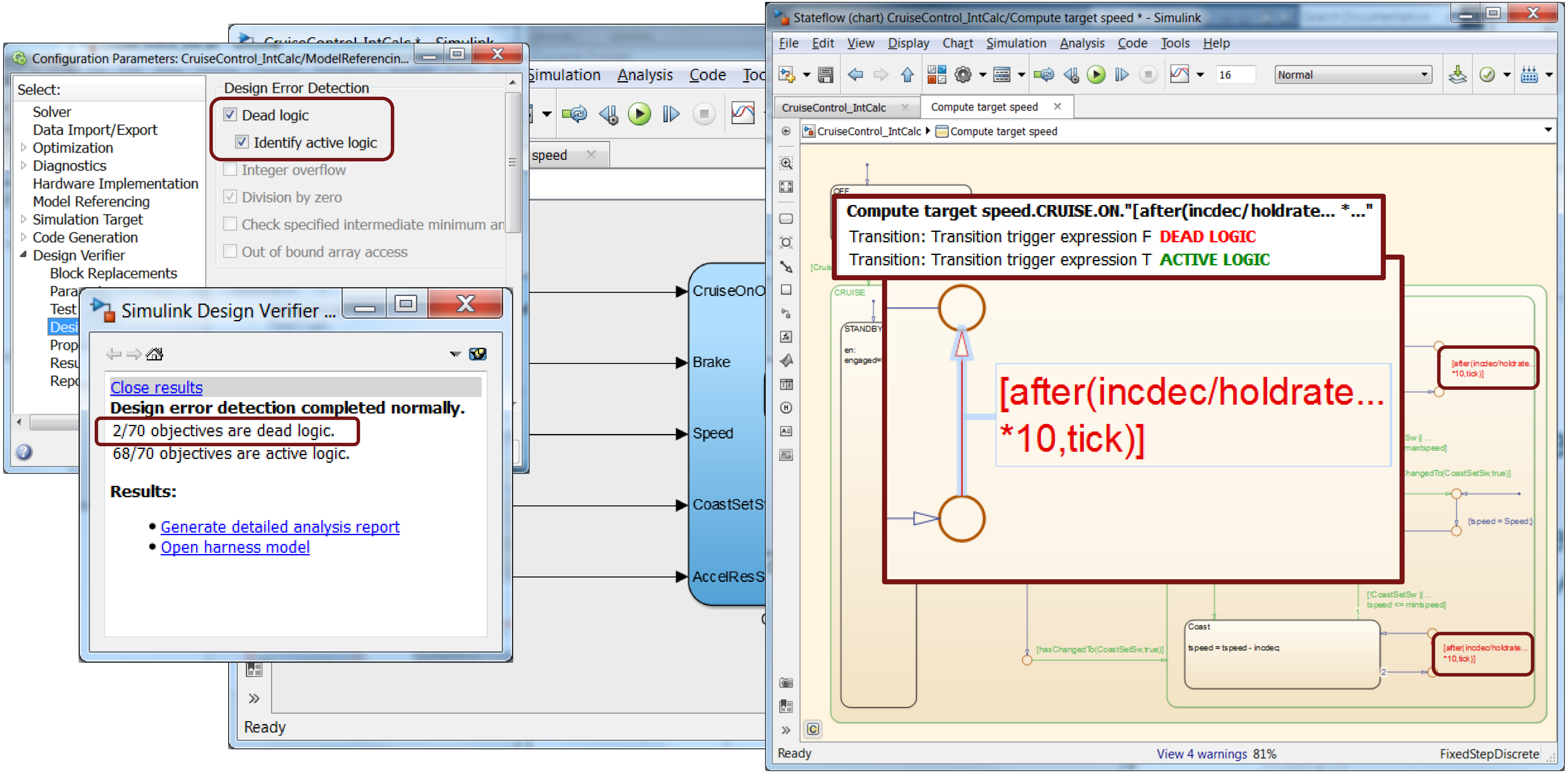
Dashboard blocks facilitate
early ad-hoc testing



Gaining Confidence in our Design



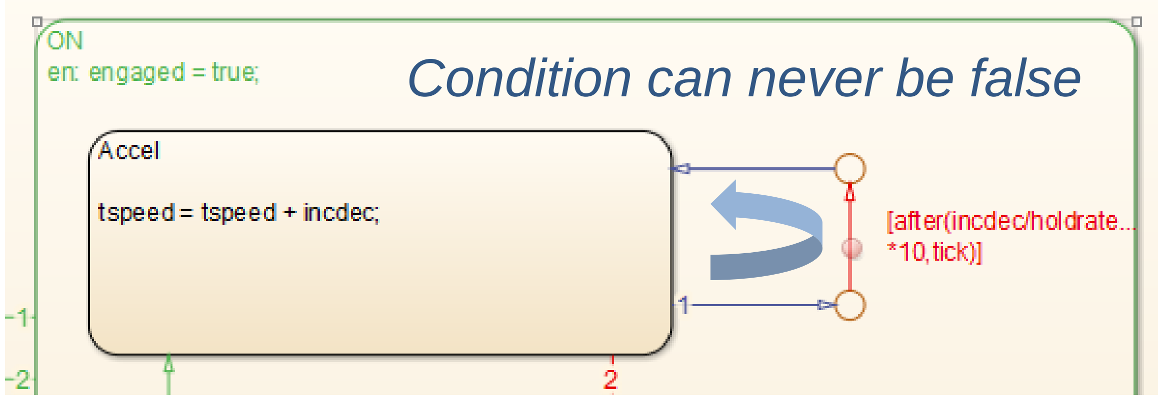
Finding Design Errors: Dead Logic



Finding Unintended Behavior

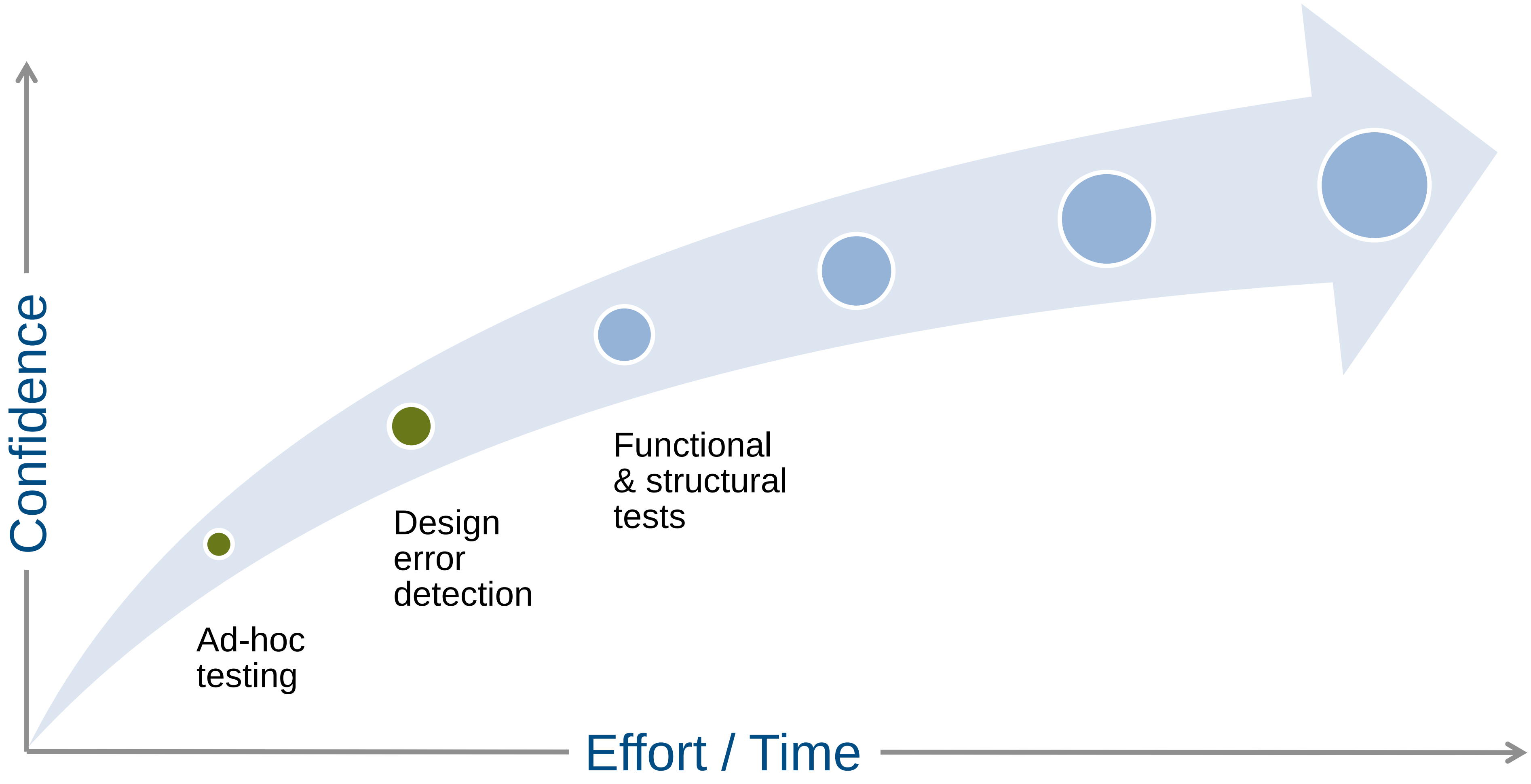
Command Window

```
debug>> incdec
incdec =
     1

debug>> holdrate
holdrate =
     5

debug>> class(incdec)
ans =
uint8

debug>> class(holdrate)
ans =
uint8

debug>> incdec/holdrate*10
ans =
     0

debug>> 10*incdec/holdrate
ans =
     2

fx debug>>
```



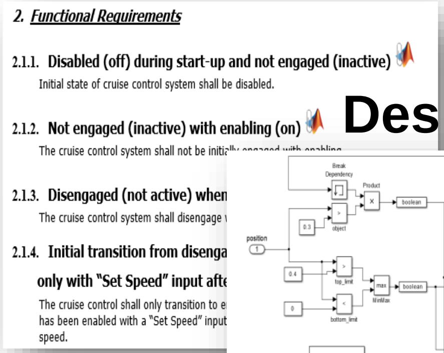
- Dead logic due to “uint8” operation on incdec/holdrate*10
- Fix change the order of operation 10*incdec/holdrate

Gaining Confidence in our Design

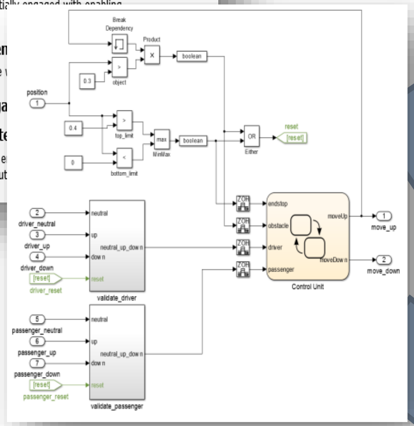


Simulation Testing Workflow

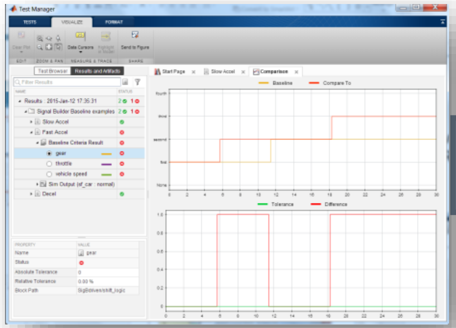
Requirements



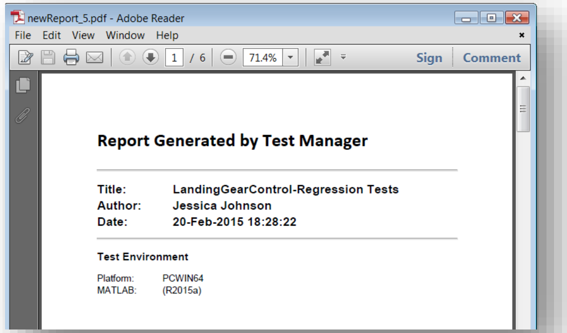
Design



Did we meet requirements?



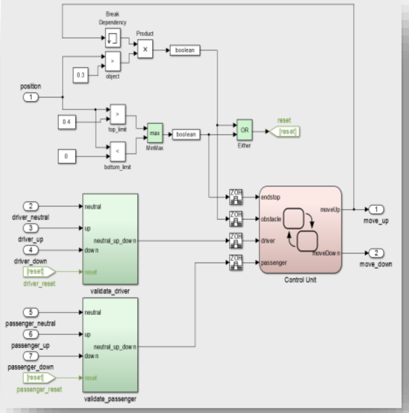
Review functional behavior



Functional

Structural

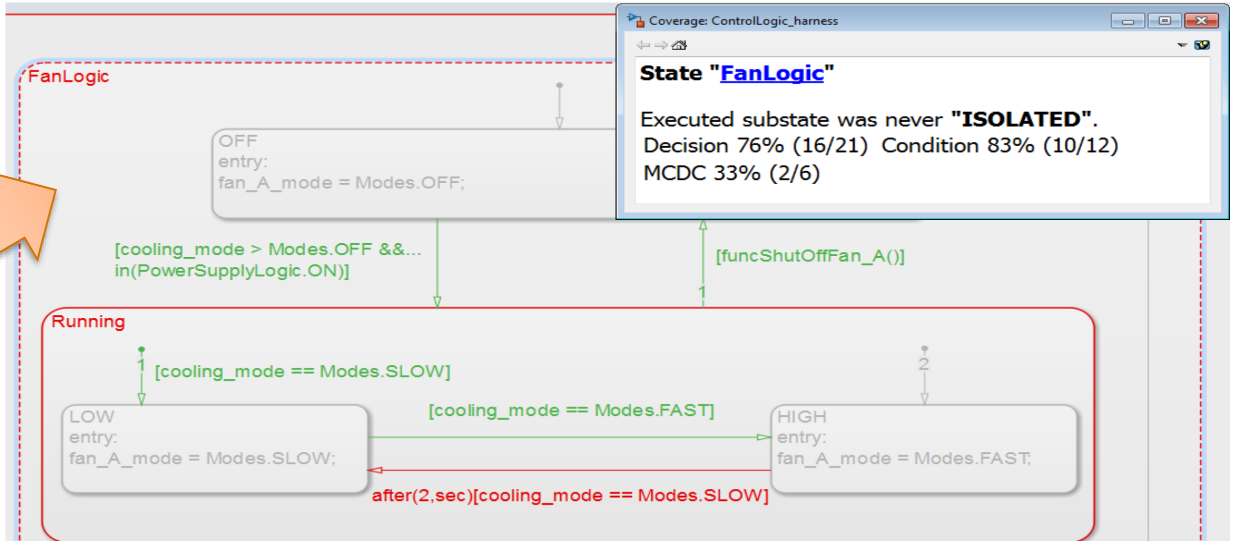
Did we completely test our model?



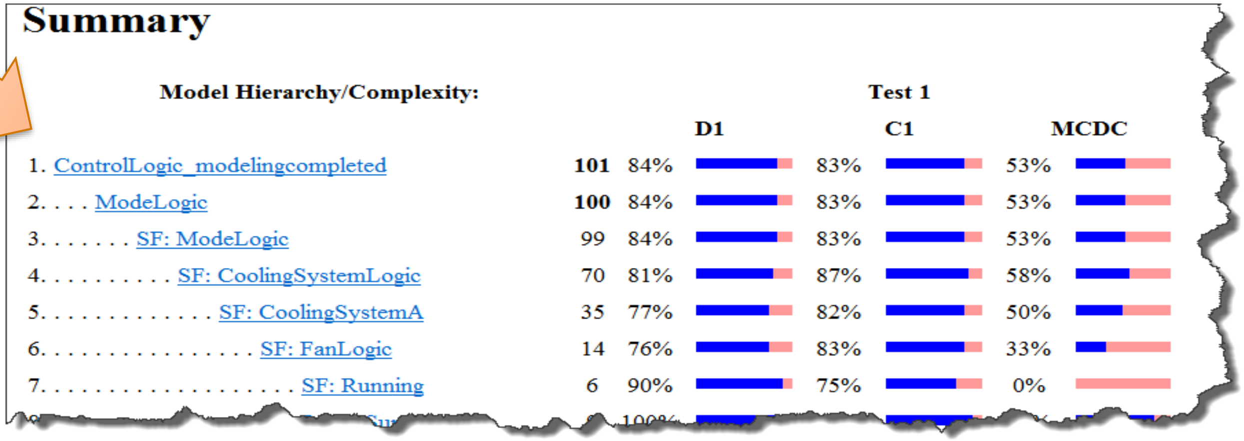
Structural coverage report

		D1	C1	MCDC
1. slvnvdemo_powerwindow_controller	58	100%	80%	60%
2. . . . control	57	100%	80%	60%
3. . . . . . SF: control	56	100%	80%	60%
4. . . . . SF: safe	52	100%	75%	50%
5. . . . . SF: driverDown	7	100%	75%	50%
6. . . . . SF: driverNeutral	26	100%	75%	50%
7. . . . . SF: passengerDown	7	100%	75%	50%
8. . . . . SF: passengerUp	7	100%	75%	50%
9. . . . . SF: driverUp	7	100%	75%	50%

Did We Completely Test our Model?



Model Coverage Analysis



Summary

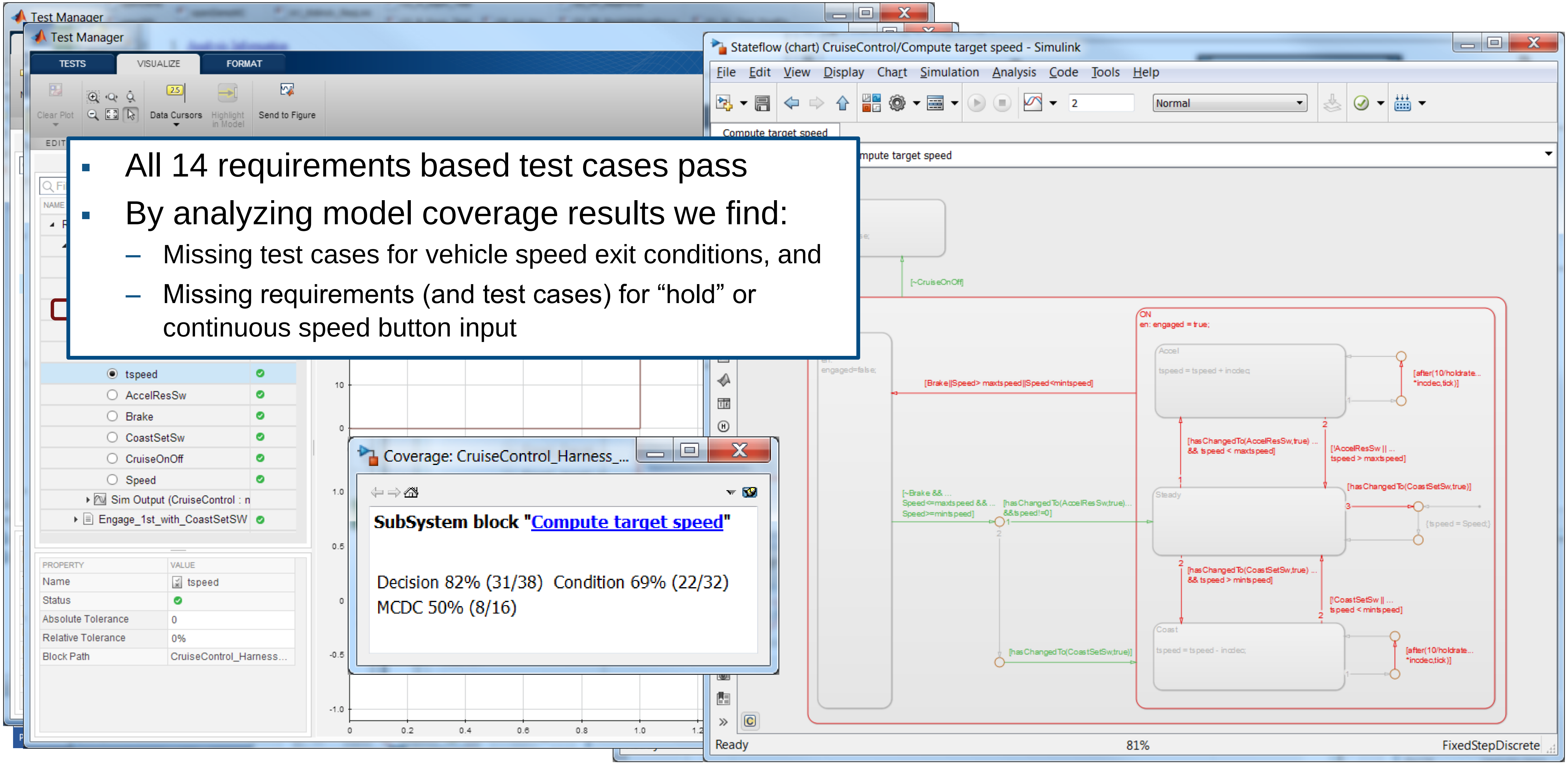
Model Hierarchy/Complexity:		Test 1 D1	C1	MCDC
1. ControlLogic_modelingcompleted	101	84%	83%	53%
2. . . . ModeLogic	100	84%	83%	53%
3. . . . . . . SF: ModeLogic	99	84%	83%	53%
4. . . . . SF: CoolingSystemLogic	70	81%	87%	58%
5. . . . . SF: CoolingSystemA	35	77%	82%	50%
6. . . . . SF: FanLogic	14	76%	83%	33%
7. . . . . SF: Running	6	90%	75%	0%

Potential causes of less than 100% coverage:

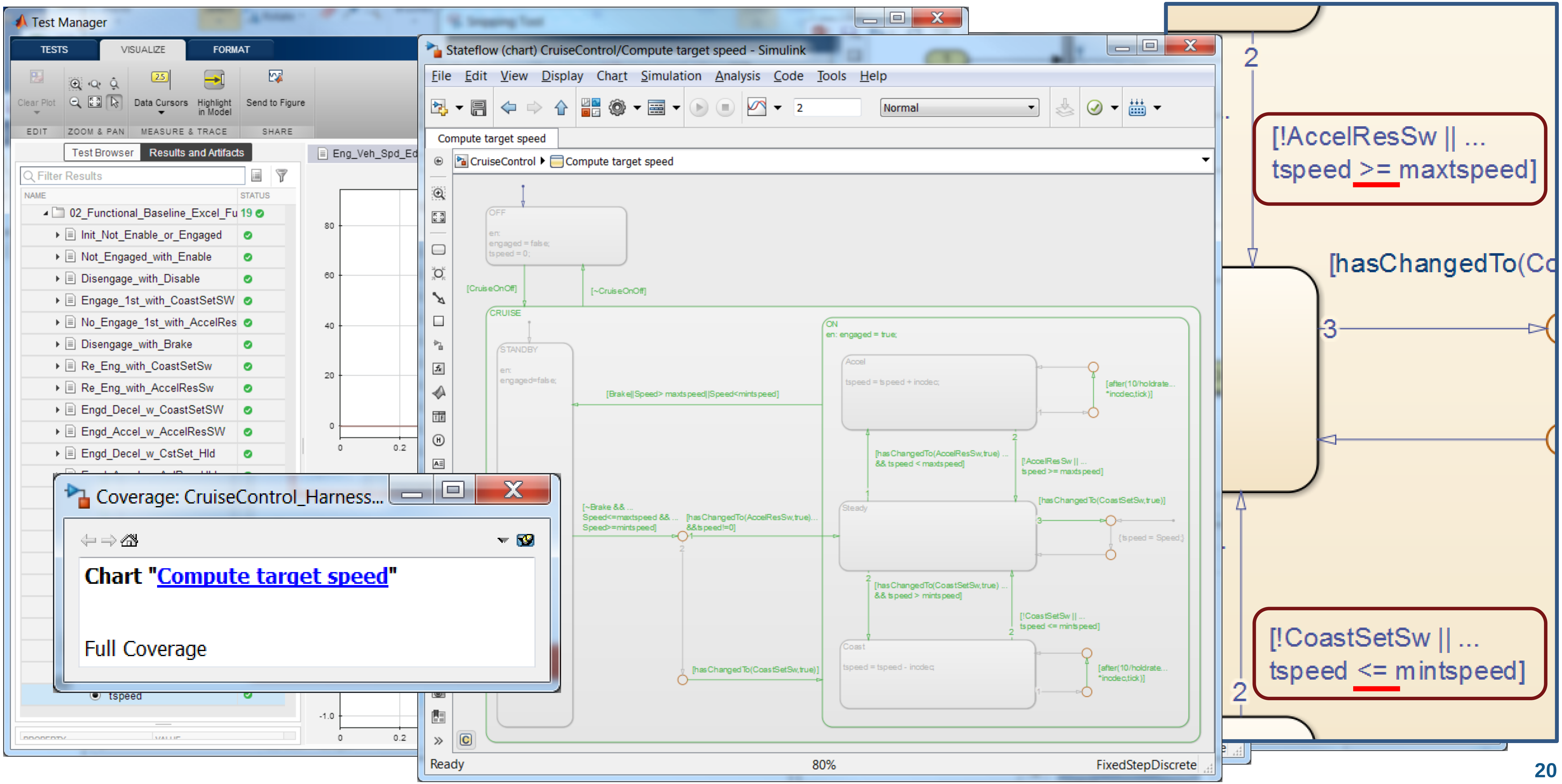
- Missing requirements
- Over-specified design
- Design errors
- Missing tests

Requirements Based Functional Testing with Coverage Analysis

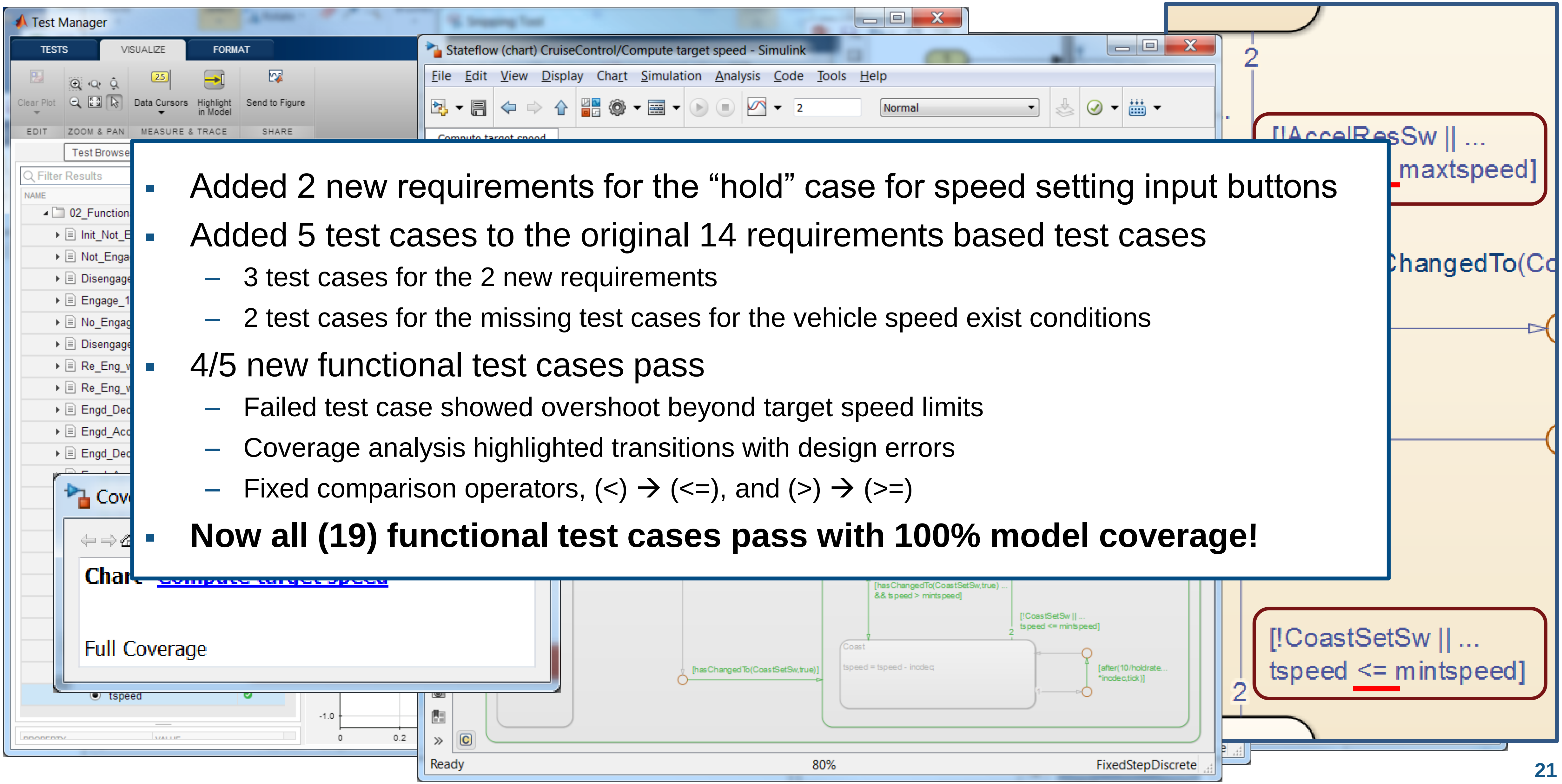
- All 14 requirements based test cases pass
- By analyzing model coverage results we find:
  - Missing test cases for vehicle speed exit conditions, and
  - Missing requirements (and test cases) for “hold” or continuous speed button input



Functional Testing with Added Requirements & Test Cases

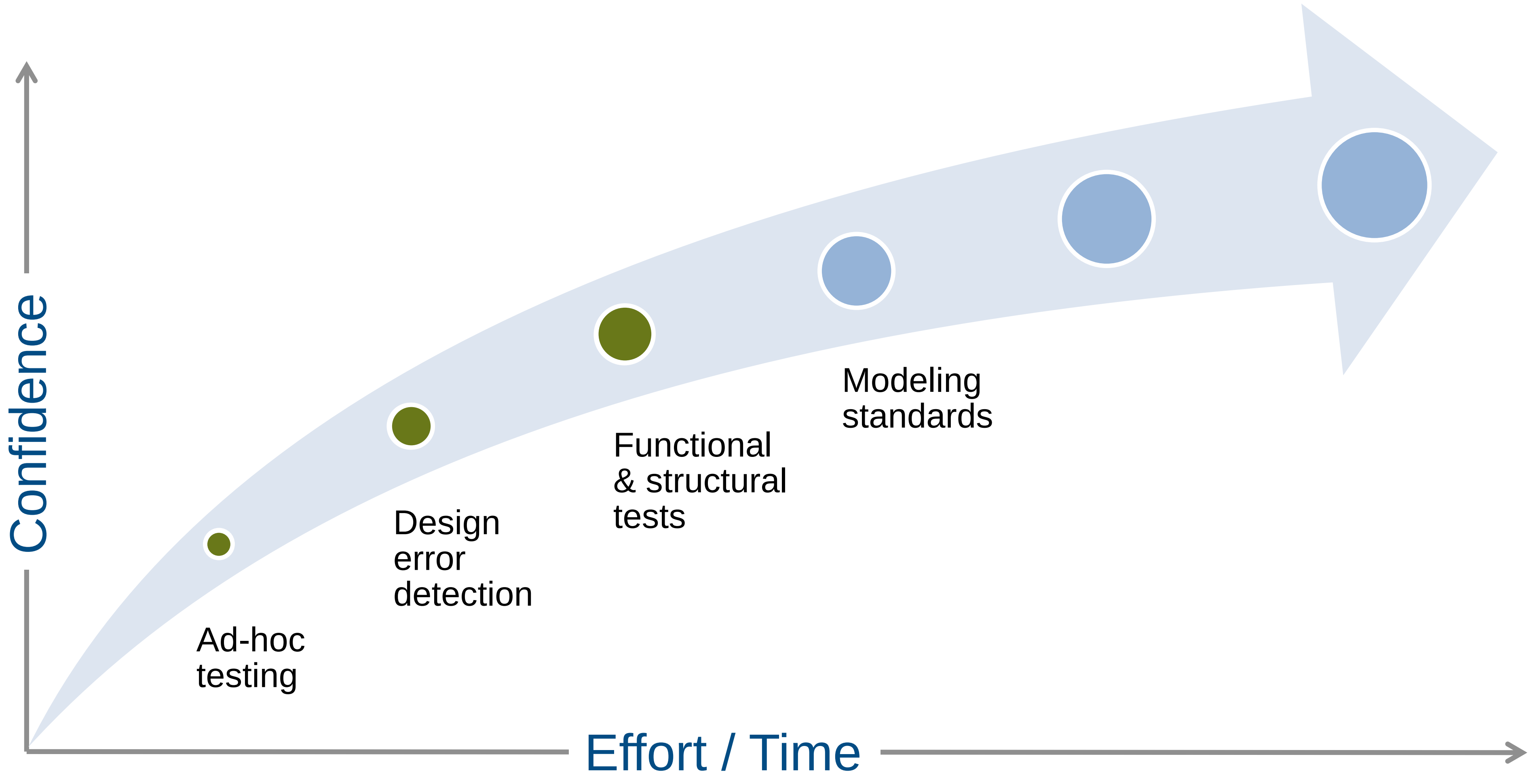


Functional Testing with Added Requirements & Test Cases

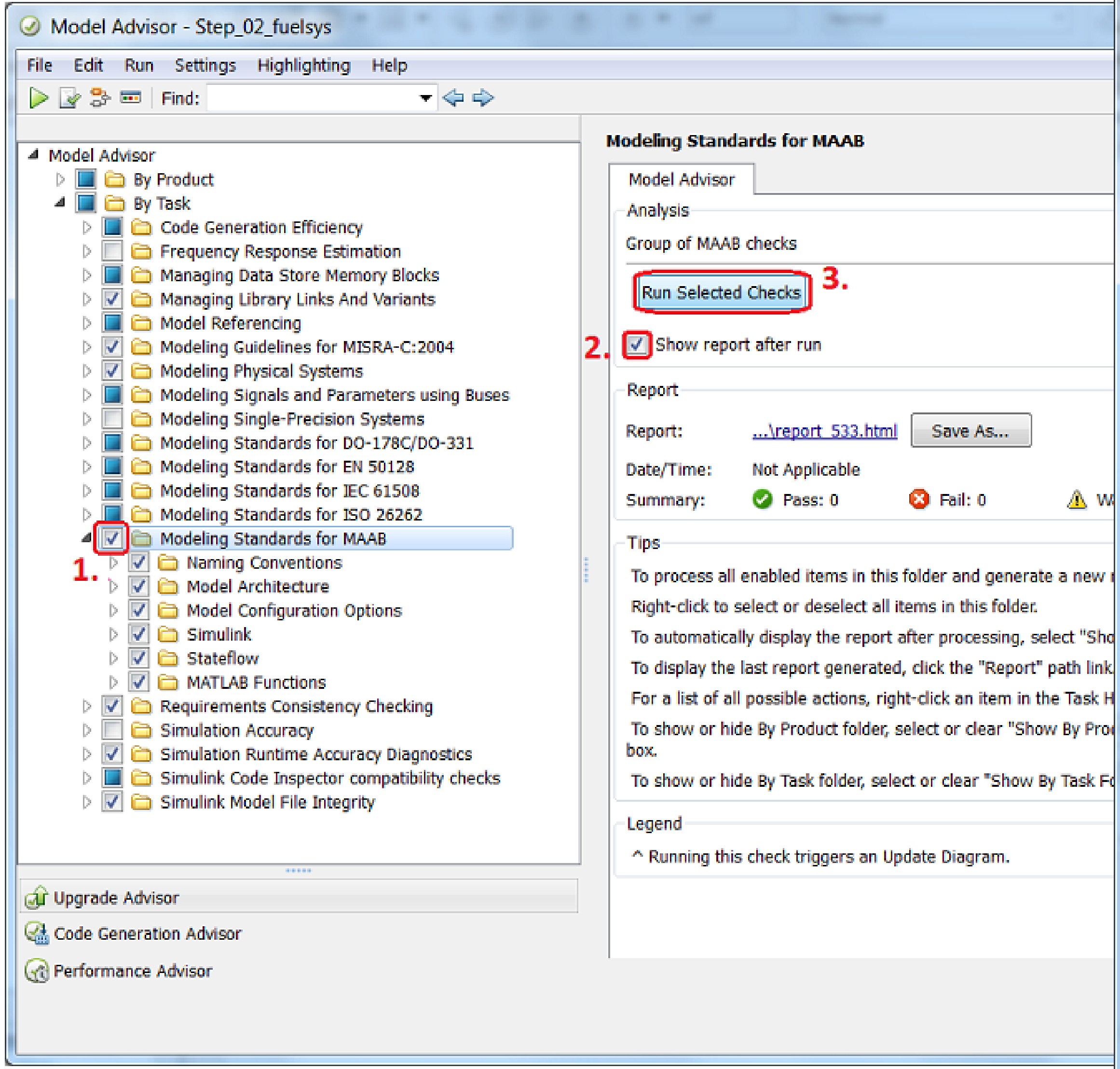


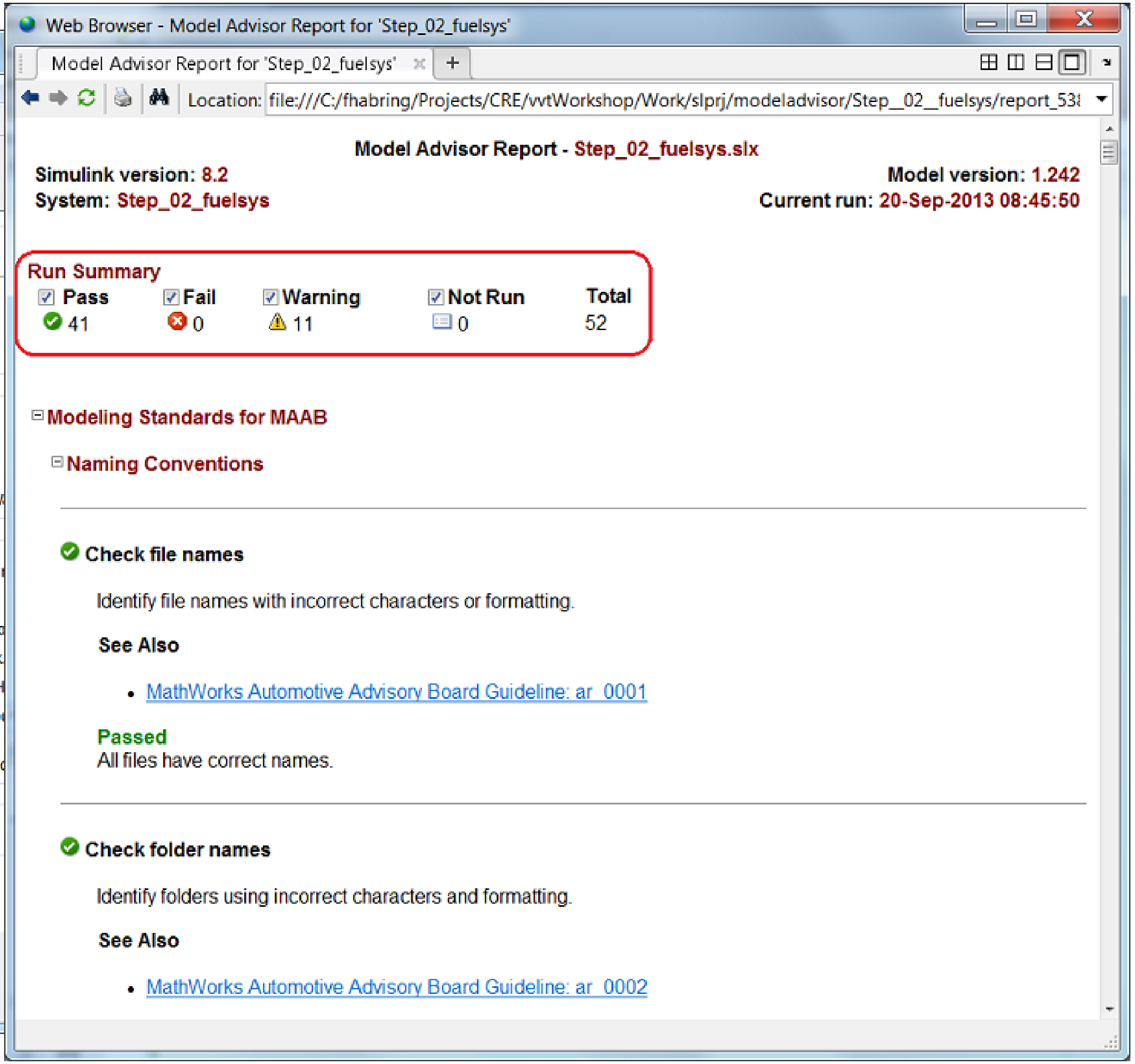
- Added 2 new requirements for the “hold” case for speed setting input buttons
- Added 5 test cases to the original 14 requirements based test cases
  - 3 test cases for the 2 new requirements
  - 2 test cases for the missing test cases for the vehicle speed exist conditions
- 4/5 new functional test cases pass
  - Failed test case showed overshoot beyond target speed limits
  - Coverage analysis highlighted transitions with design errors
  - Fixed comparison operators, (<) → (<=), and (>) → (>=)
- **Now all (19) functional test cases pass with 100% model coverage!**

Gaining Confidence in our Design

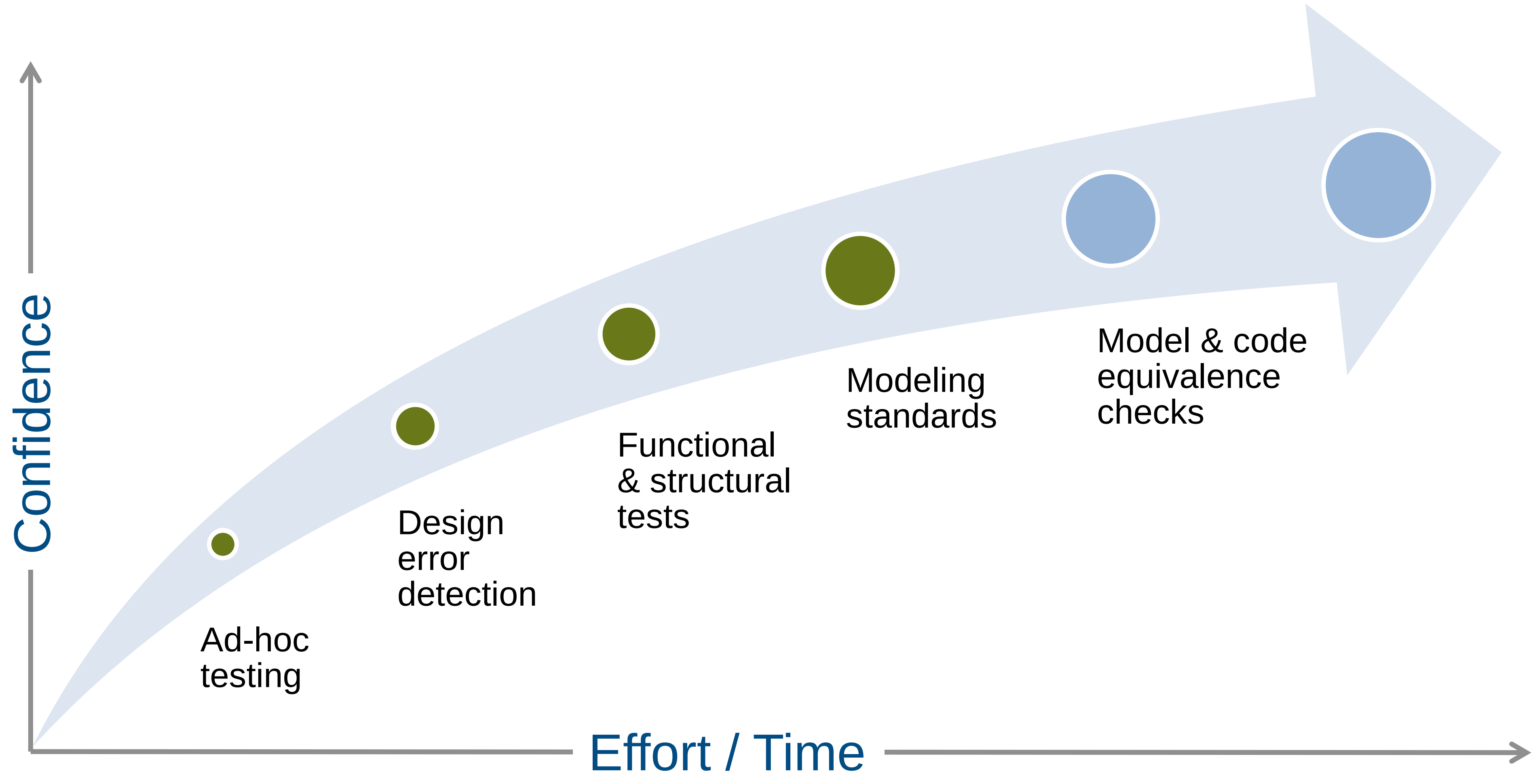


Model Advisor – Model Standards Checking

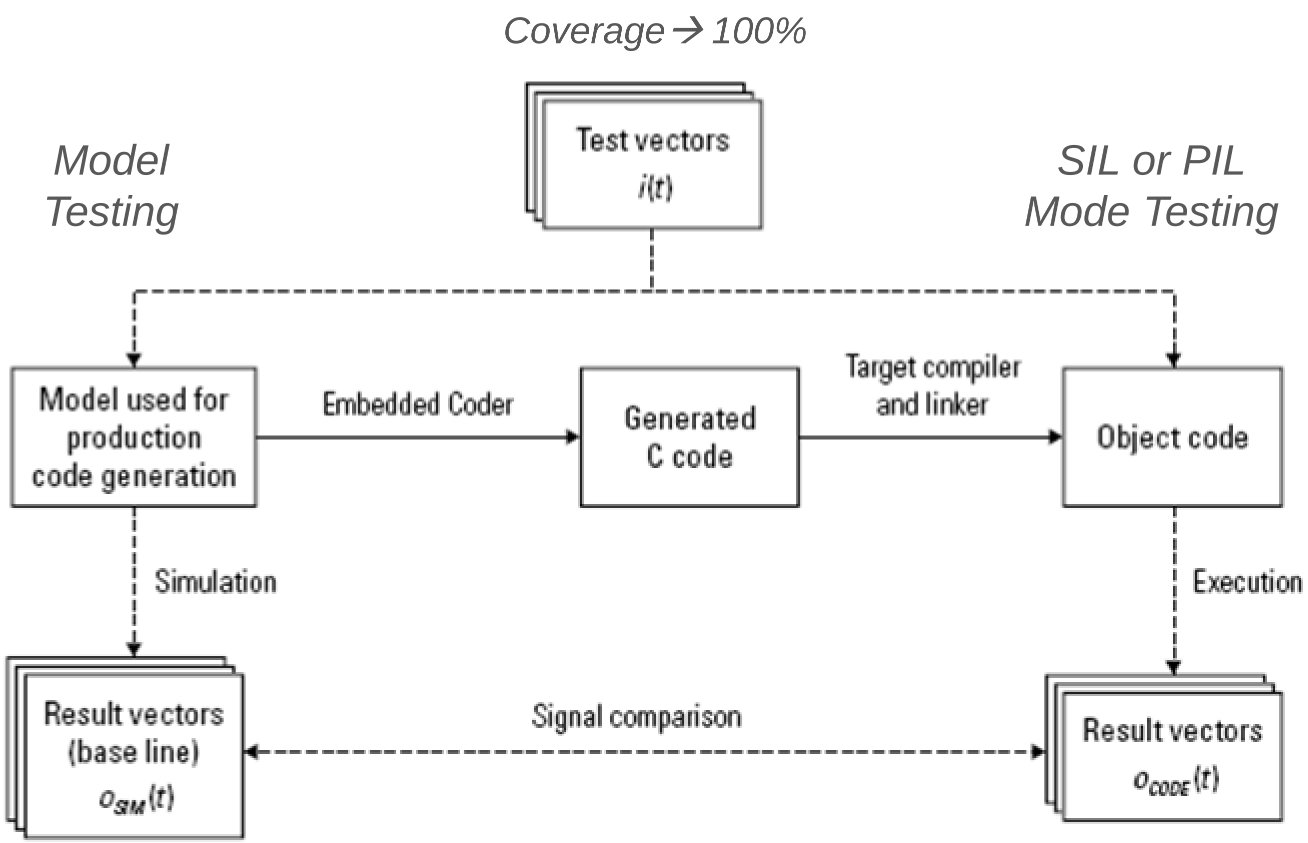




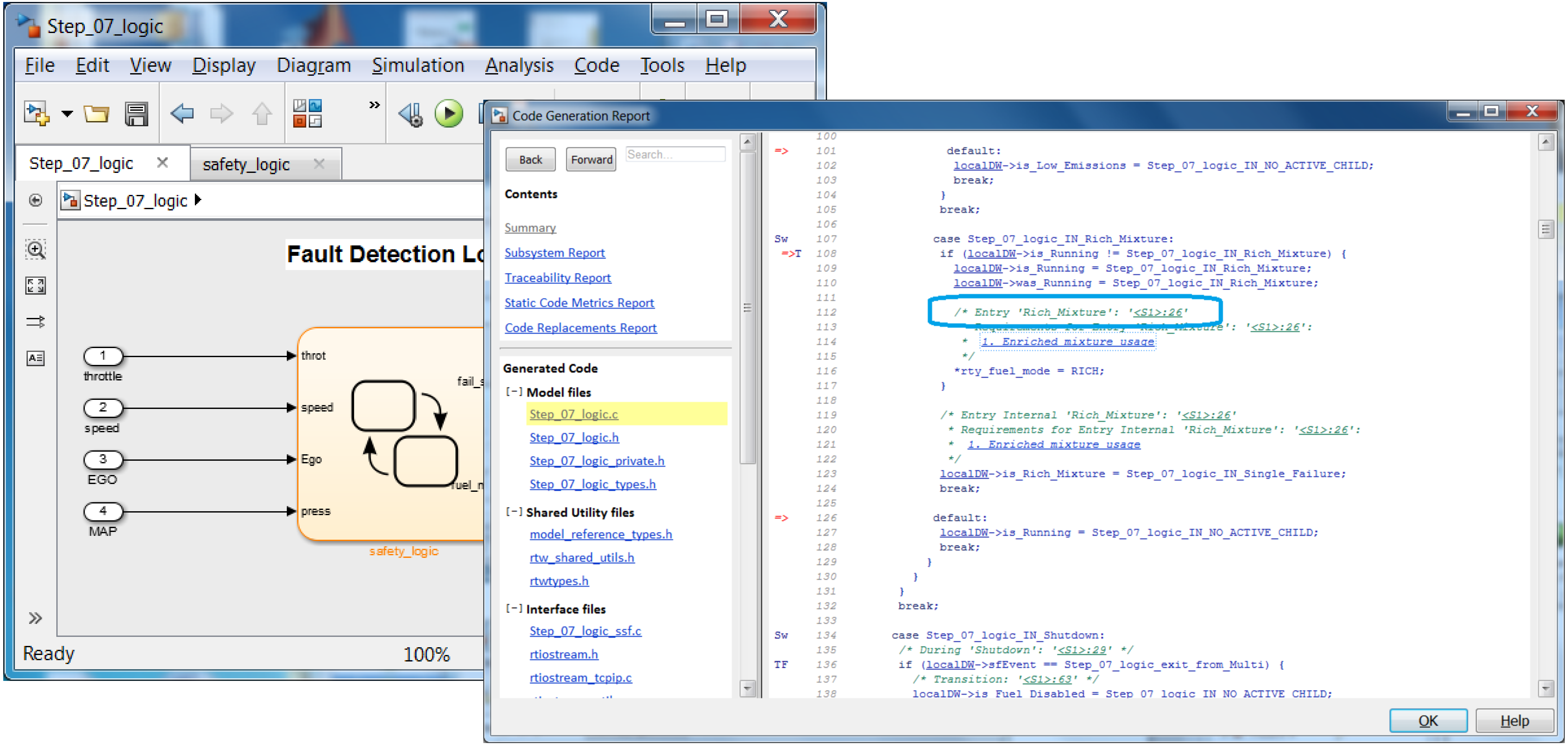
Gaining Confidence in our Design



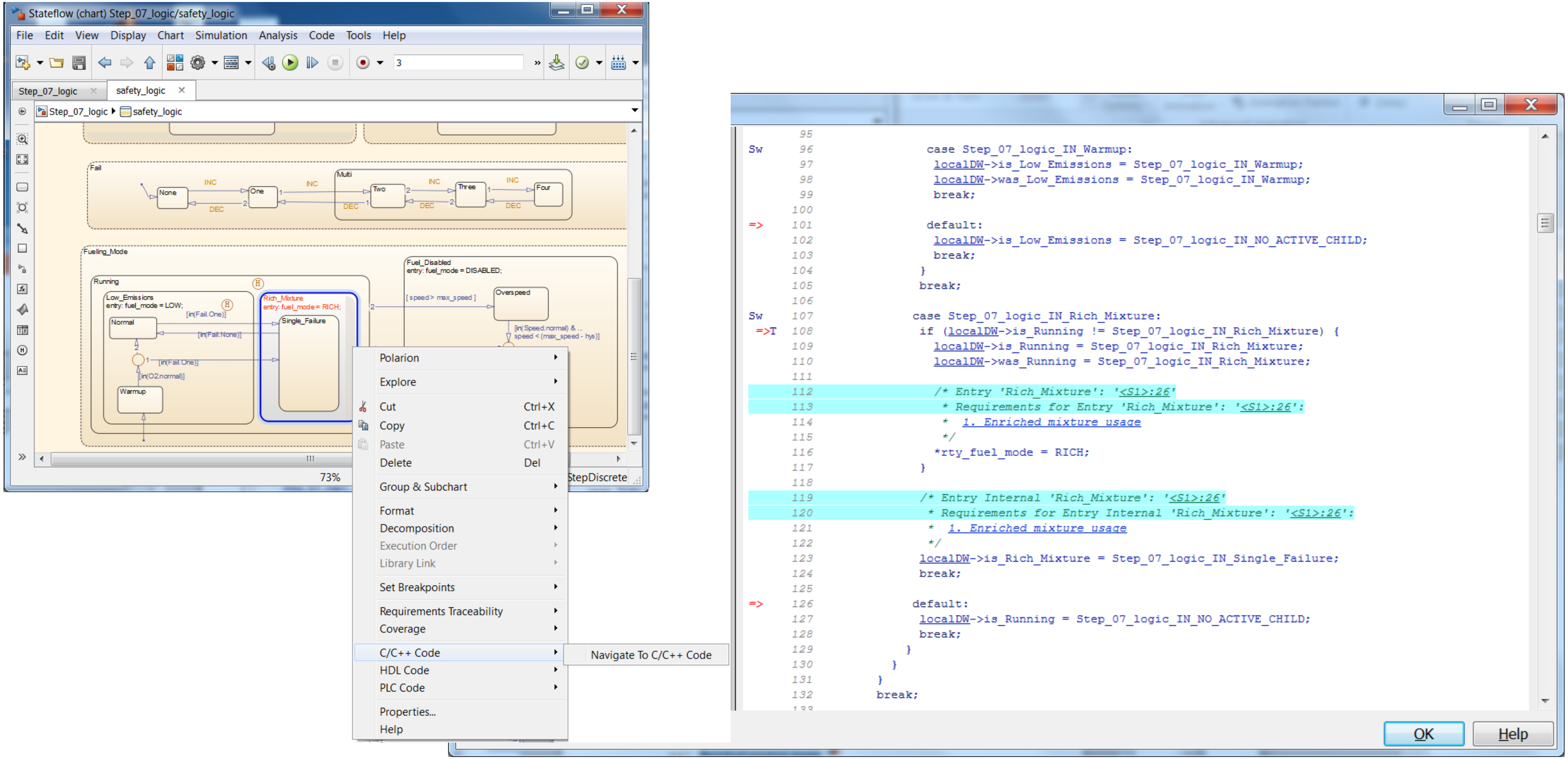
Equivalence Testing: Model vs SIL or PIL Mode Testing



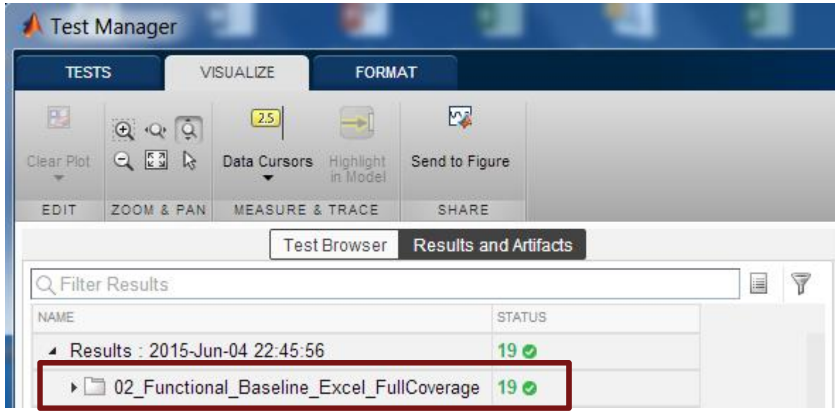
Code Generation with Model-to-Code Traceability

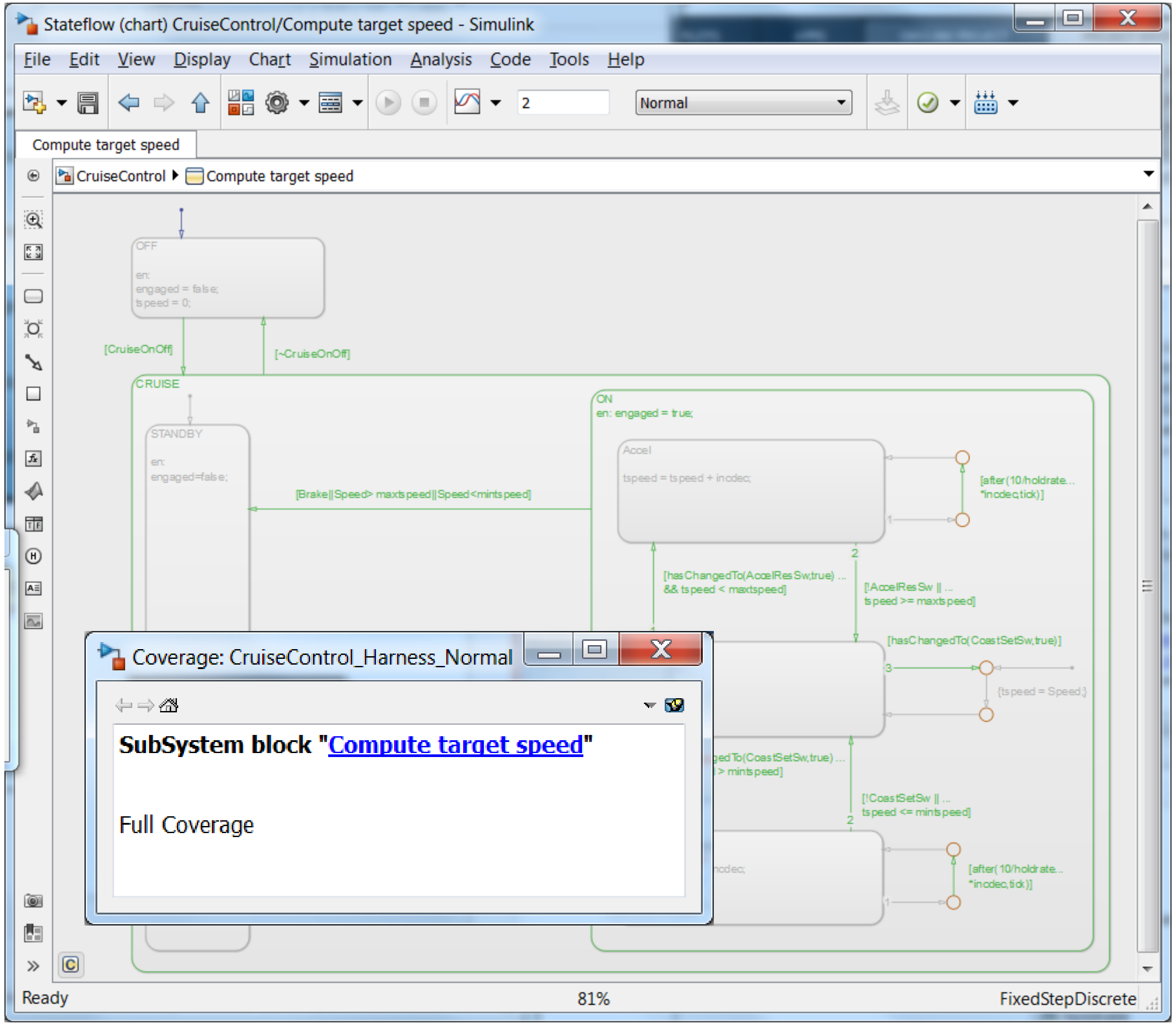


Code Generation with Model-to-Code Traceability

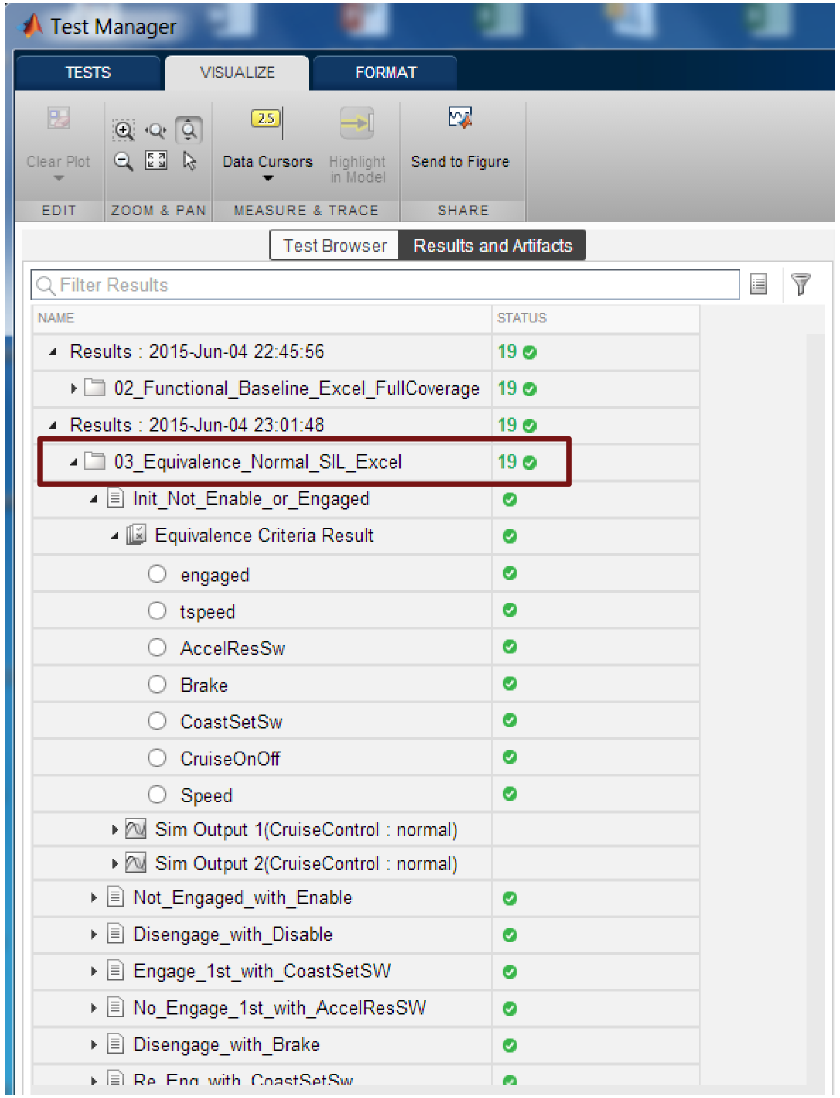


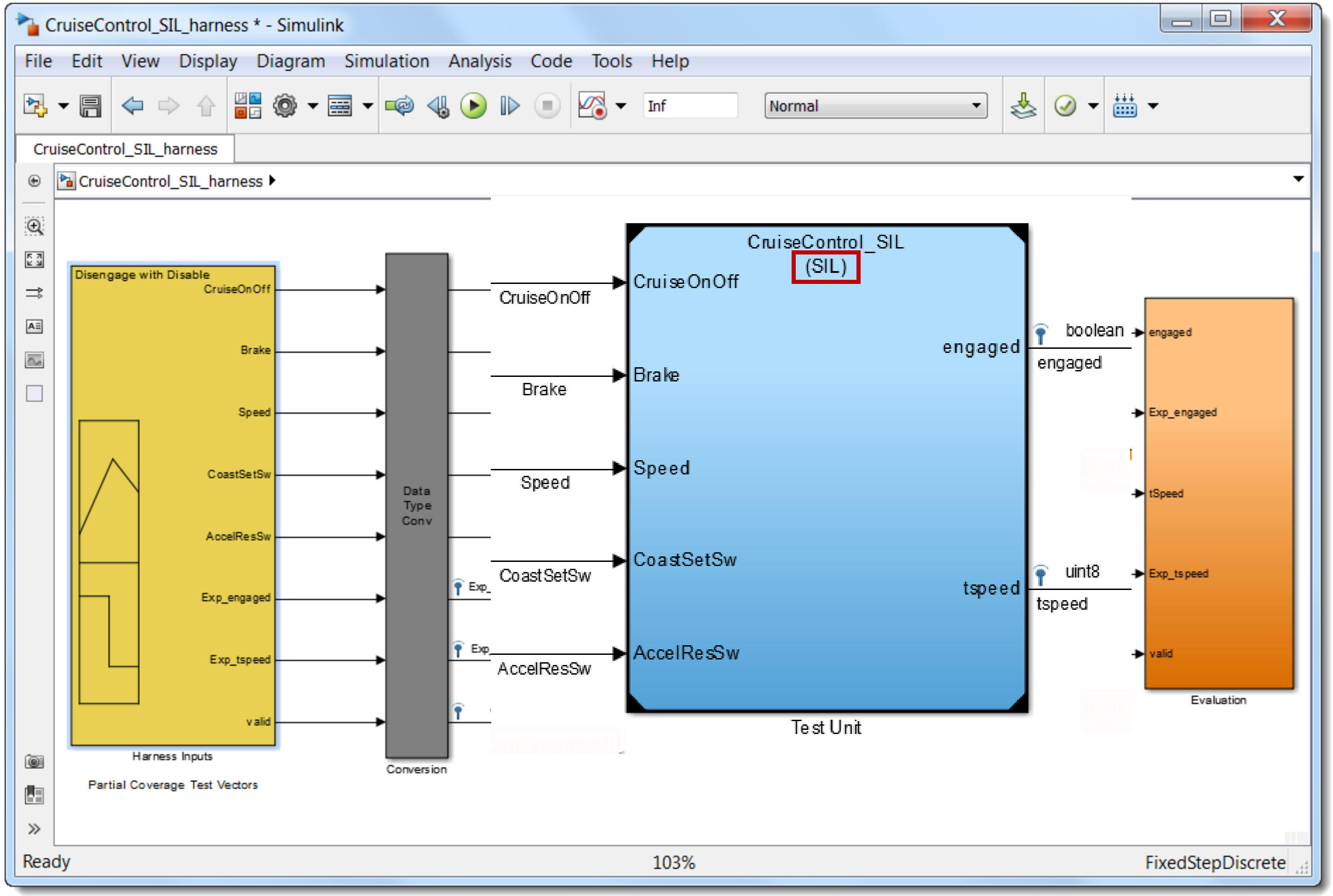
Code Equivalence Check Results: Model vs Code



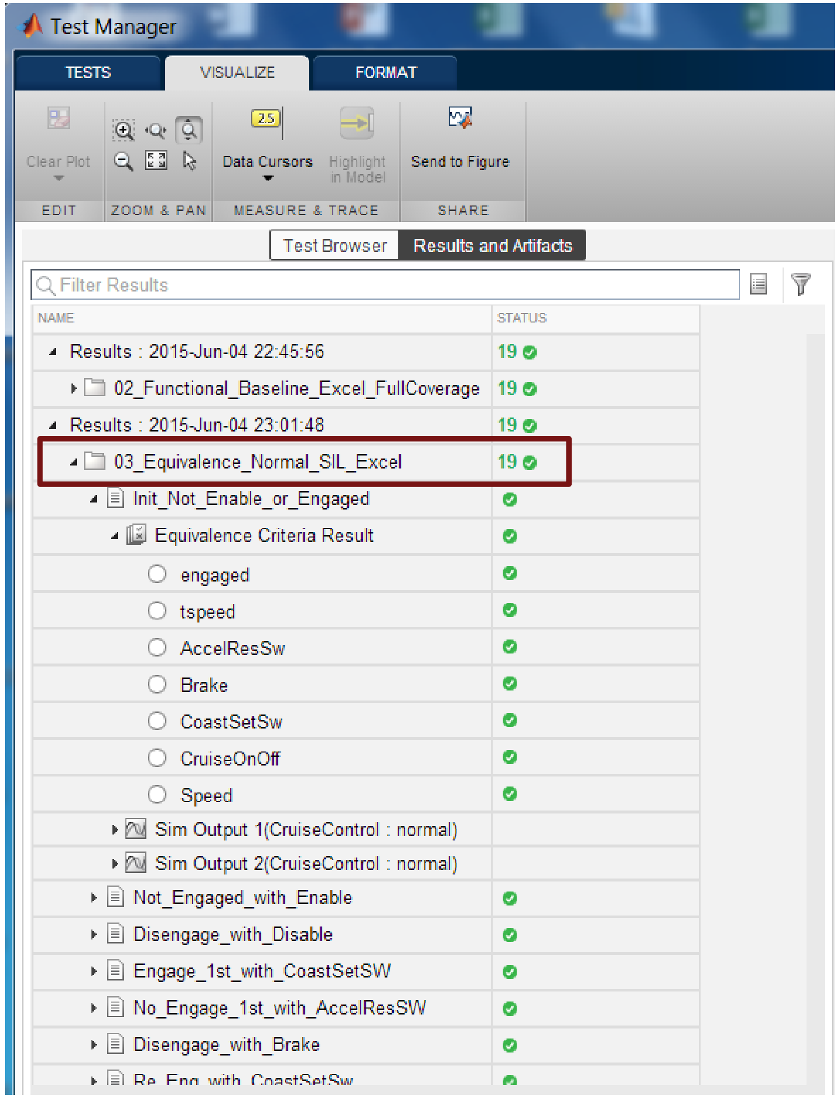


Code Equivalence Check Results: Model vs Code



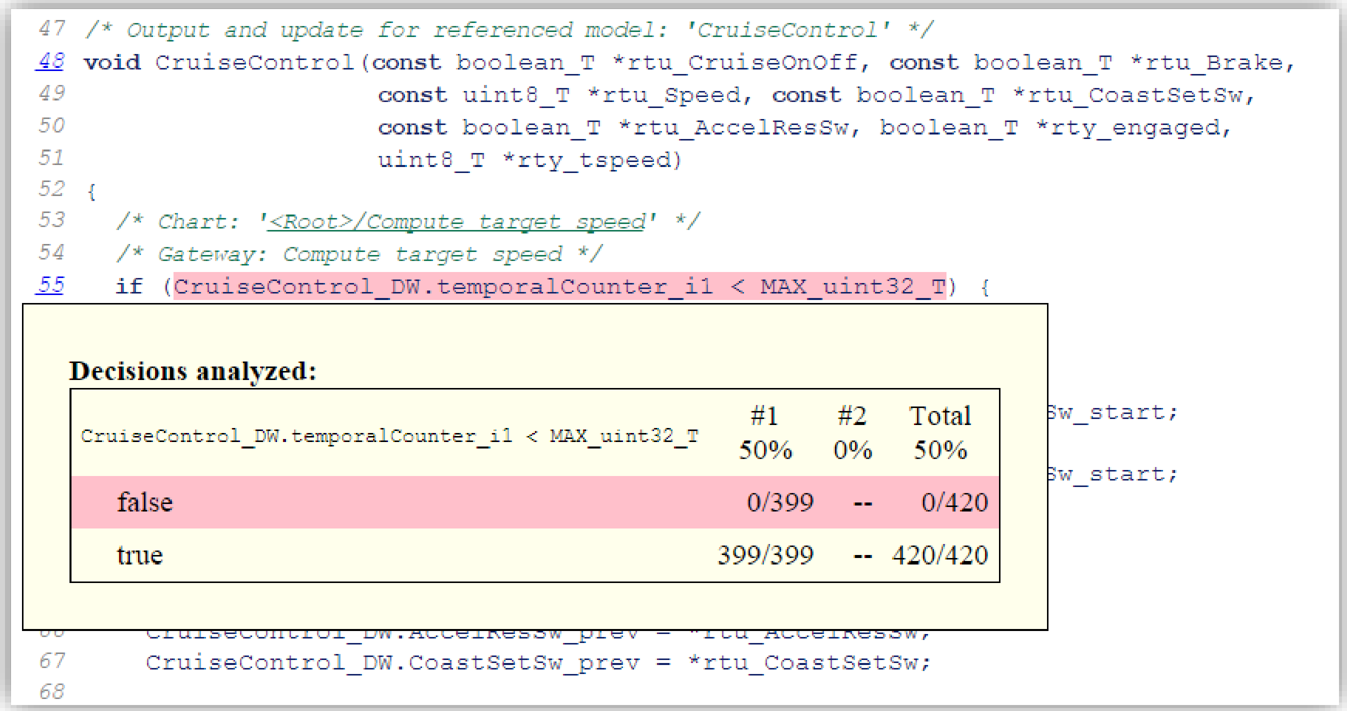


Code Equivalence Check Results: Model vs Code

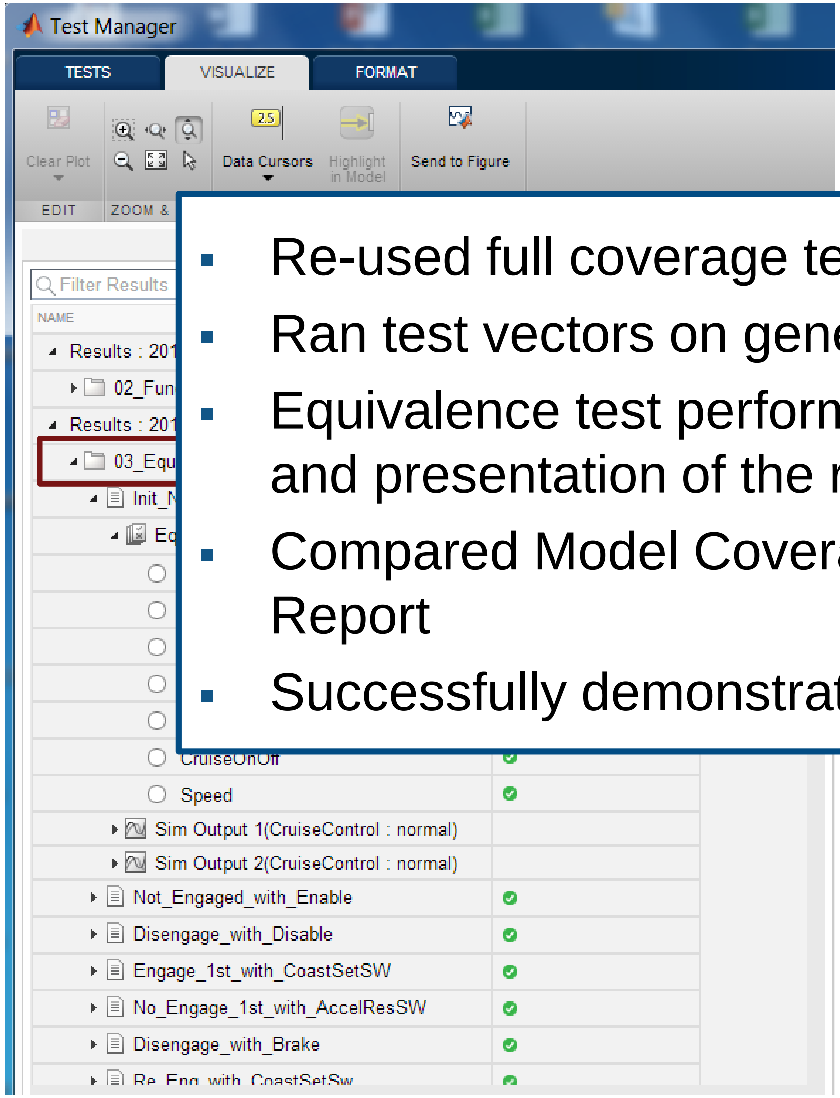


Code Coverage

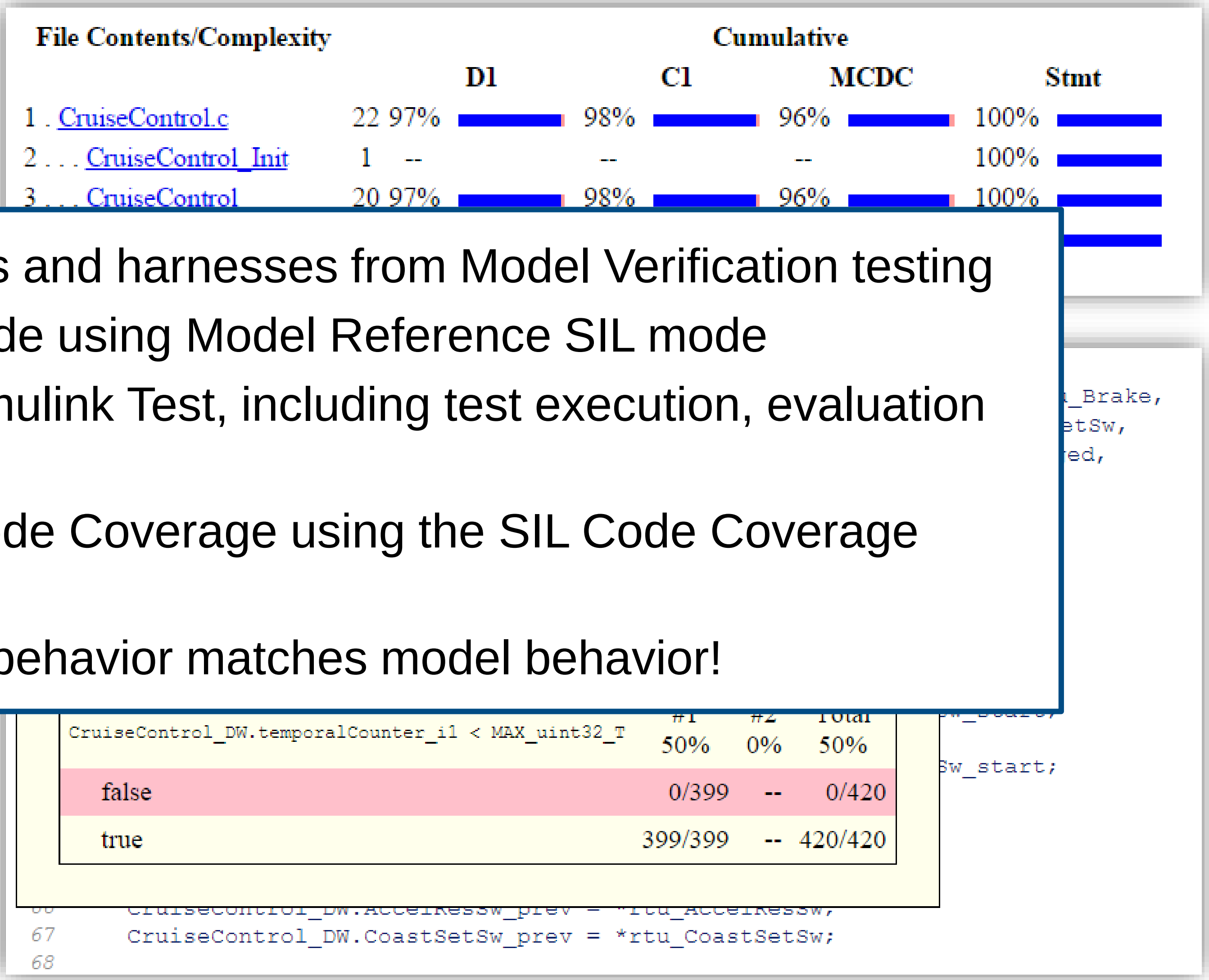
File Contents/Complexity		Cumulative D1	C1	MCDC	Stmt
1 . CruiseControl.c	22	97%	98%	96%	100%
2 . . . CruiseControl_Init	1	--	--	--	100%
3 . . . CruiseControl	20	97%	98%	96%	100%
4 . . . CruiseControl_initialize	1	--	--	--	100%



Code Equivalence Check Results: Model vs Code

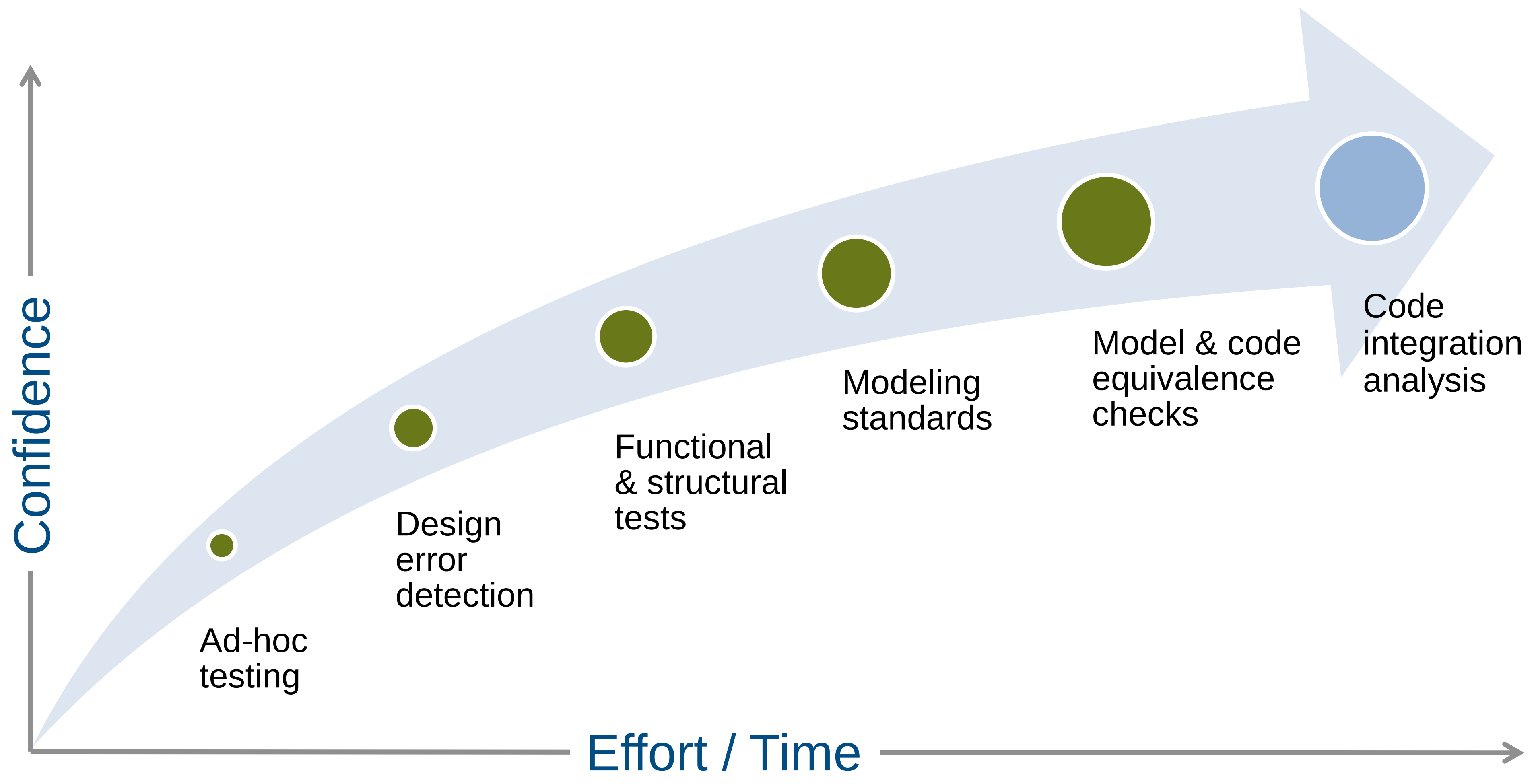


Code Coverage

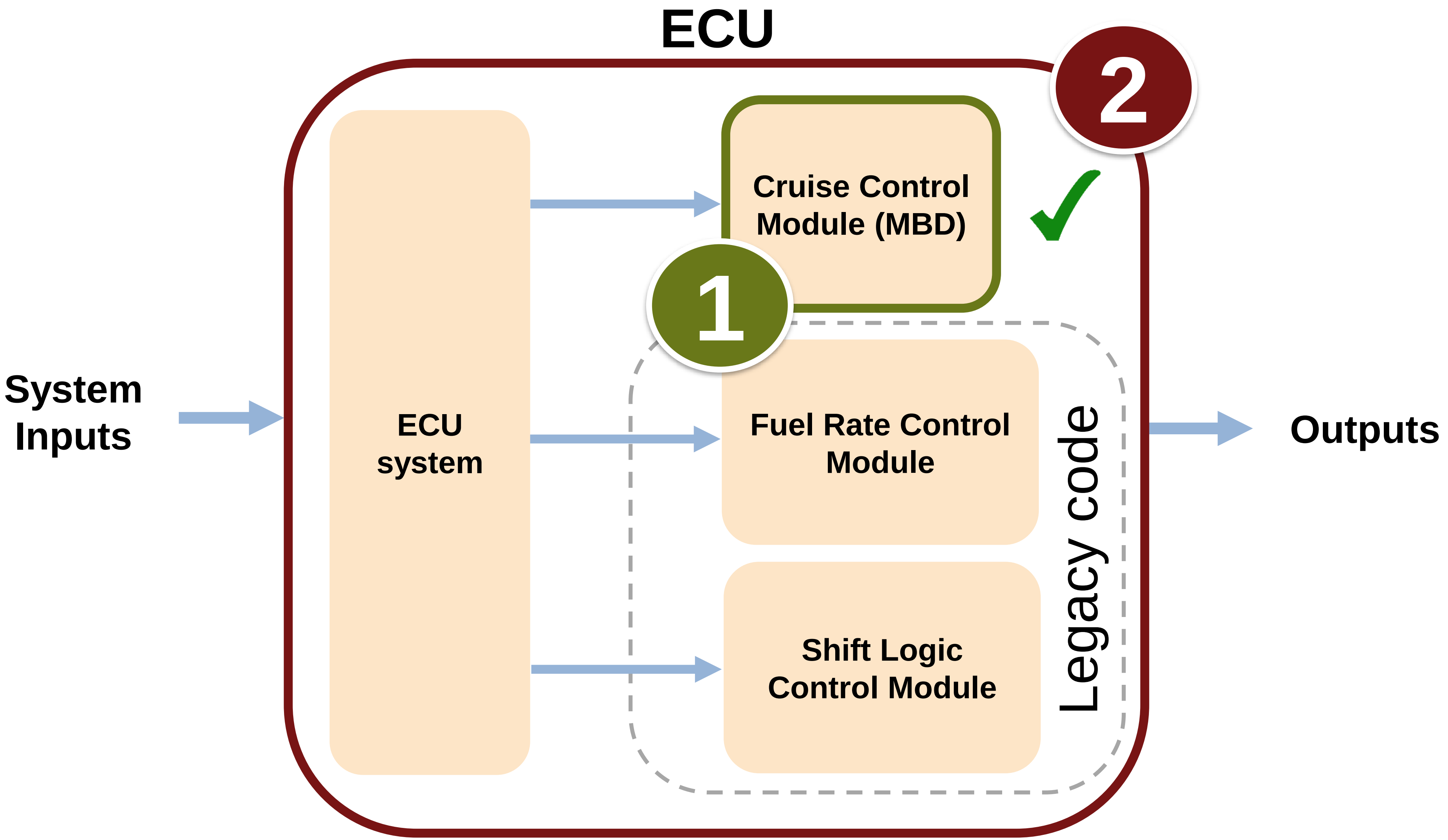


- Re-used full coverage test vectors and harnesses from Model Verification testing
- Ran test vectors on generated code using Model Reference SIL mode
- Equivalence test performed in Simulink Test, including test execution, evaluation and presentation of the results
- Compared Model Coverage to Code Coverage using the SIL Code Coverage Report
- Successfully demonstrated code behavior matches model behavior!

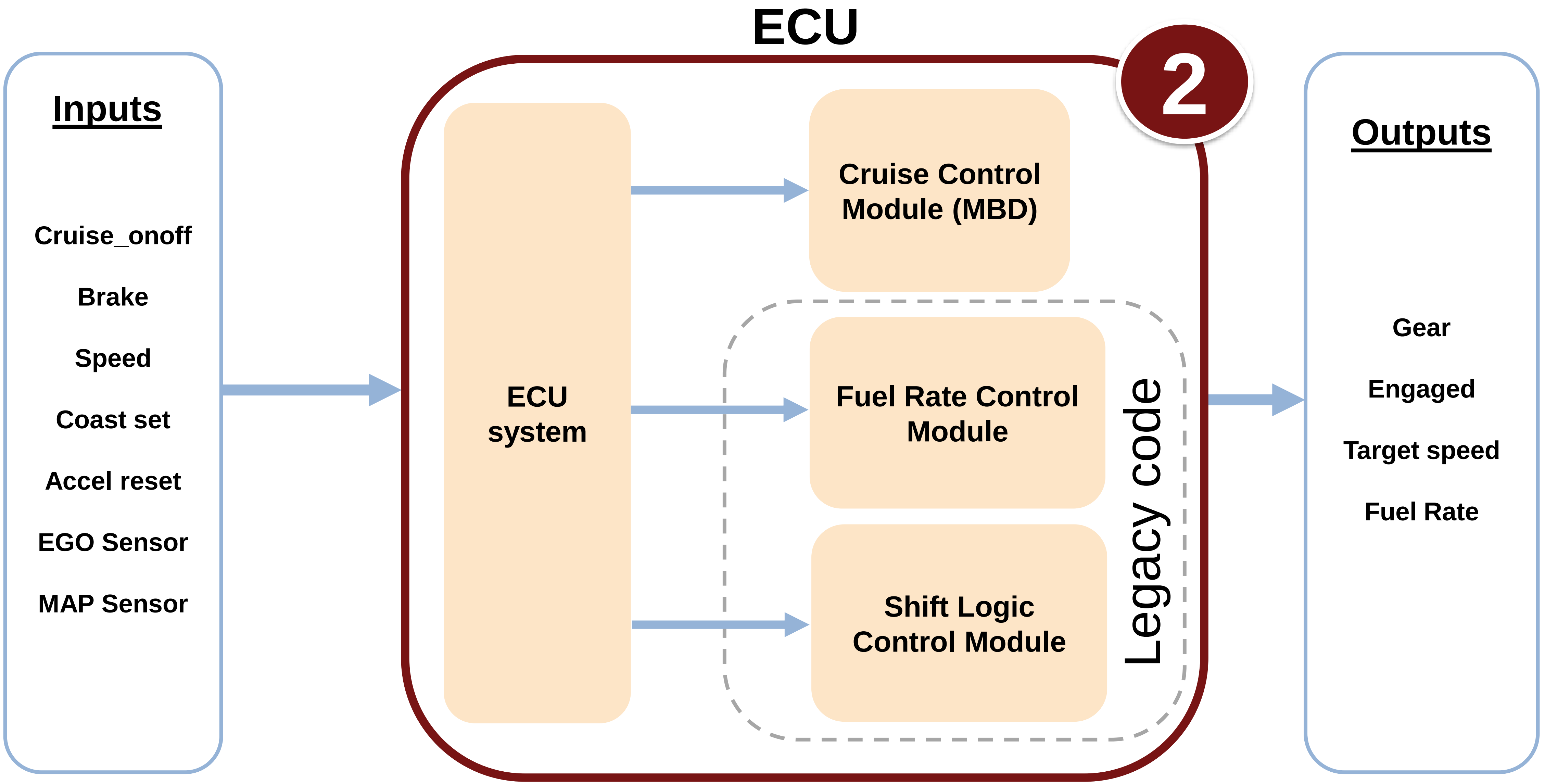
Gaining Confidence in our Design



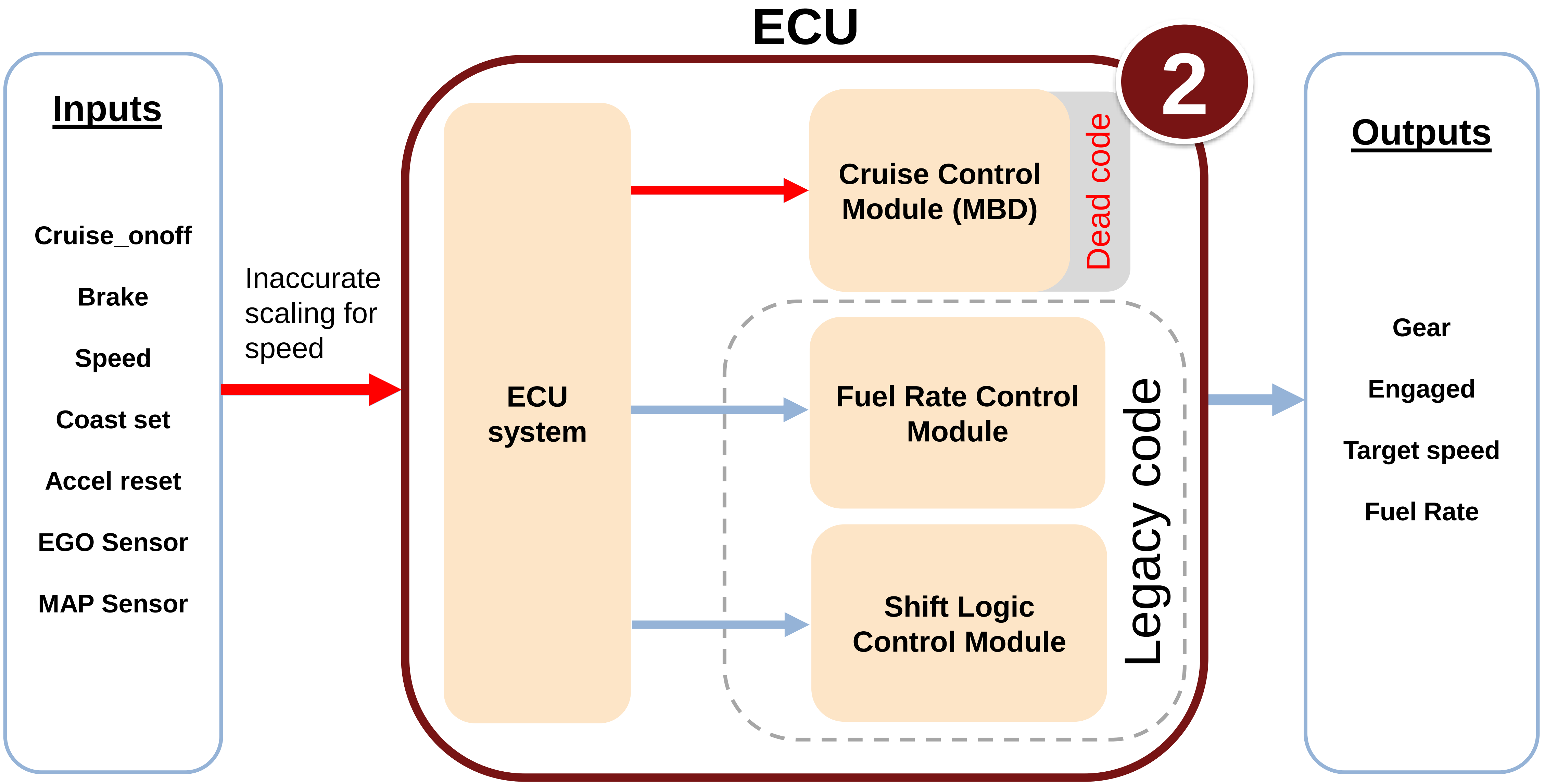
Code Integration Analysis



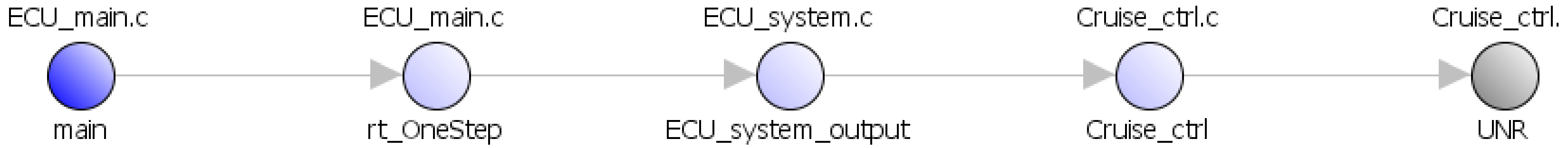
Code Integration Analysis



Finding Dead Code During Integration



Finding Dead Code with Polyspace



Target speed parameter propagated to "Cruise_ctrl.c" [0 … 40]

Maximum target speed = 90

```
    /* Entry 'STANDBY': '<S5>:52' */
    *rty_Engaged = false;
  } else if (rtu_Speed > maxtspeed) {
    /* Transition: '<S5>:55' */
    /* Exit Internal 'ON': '<S5>:54' */
    localDW->is_ON = IN_NO_ACTIVE_CHILD;
    localDW->is_CRUISE = IN_STANDBY;

    /* Entry 'STANDBY': '<S5>:52' */
    *rty_Engaged = false;
  } else if (rtu_Speed < mintspeed) {
    /* Transition: '<S5>:113' */
```

Dead code

Root Cause for Dead Code: Speed Sensor Input Hand Code

Changing analog-to-digital converter from 14 to 12-bit results in dead code

MASK – accounts for scaling down for new ADC from 14-bit to 12-bit

CONV_FACTOR – accounts for translating sensor input counts to mph

Overlooked changing CONV_FACTOR for new ADC

*C:\Working\V...d_ECU\inFunc.c

```
#define NEW_ECU
/* Changing the mask (MASK) to account for new ECU per requirement CR102*/
#ifdef NEW_ECU
MASK = 0xFFF; //New ECU
#else
MASK = 0x3FFF; //Original design specification
#endif
/* Scaling factor CONV_FACTOR for translating sensory input to miles/hr */
const real32_T CONV_FACTOR = 0.01; // FAILS
void readHW(void)
{
gvar_U_ECU_system_CruiseOnOff = CruiseOnOff;
gvar_U_ECU_system_Brake = Brake;
uint16_T temp=speed_inp;
gvar_U_ECU_system_Speed = CONV_FACTOR * (temp & MASK) ;
```

Polyspace Code Analysis

Start with C/C++ source code

```
static void pointer_arithmetic (void) {
    int array[100];
    int *p = array;
    int i;

    for (i = 0; i < 100; i++) {
      *p = 0;
      p++;
    }

  if (get_bus_status() > 0) {
    if (get_oil_pressure() > 0) {
      *p = 5;
    } else {
      i++;
    }
  }

  i = get_bus_status();

  if (i >= 0) {
    *(p - i) = 10;
  }
}
```

Polyspace Code Analysis

Source code painted in green, red, gray, orange

Green: reliable
safe pointer access

Red: faulty
out of bounds error

Gray: dead
unreachable code

Orange: unproven
may be unsafe for some conditions

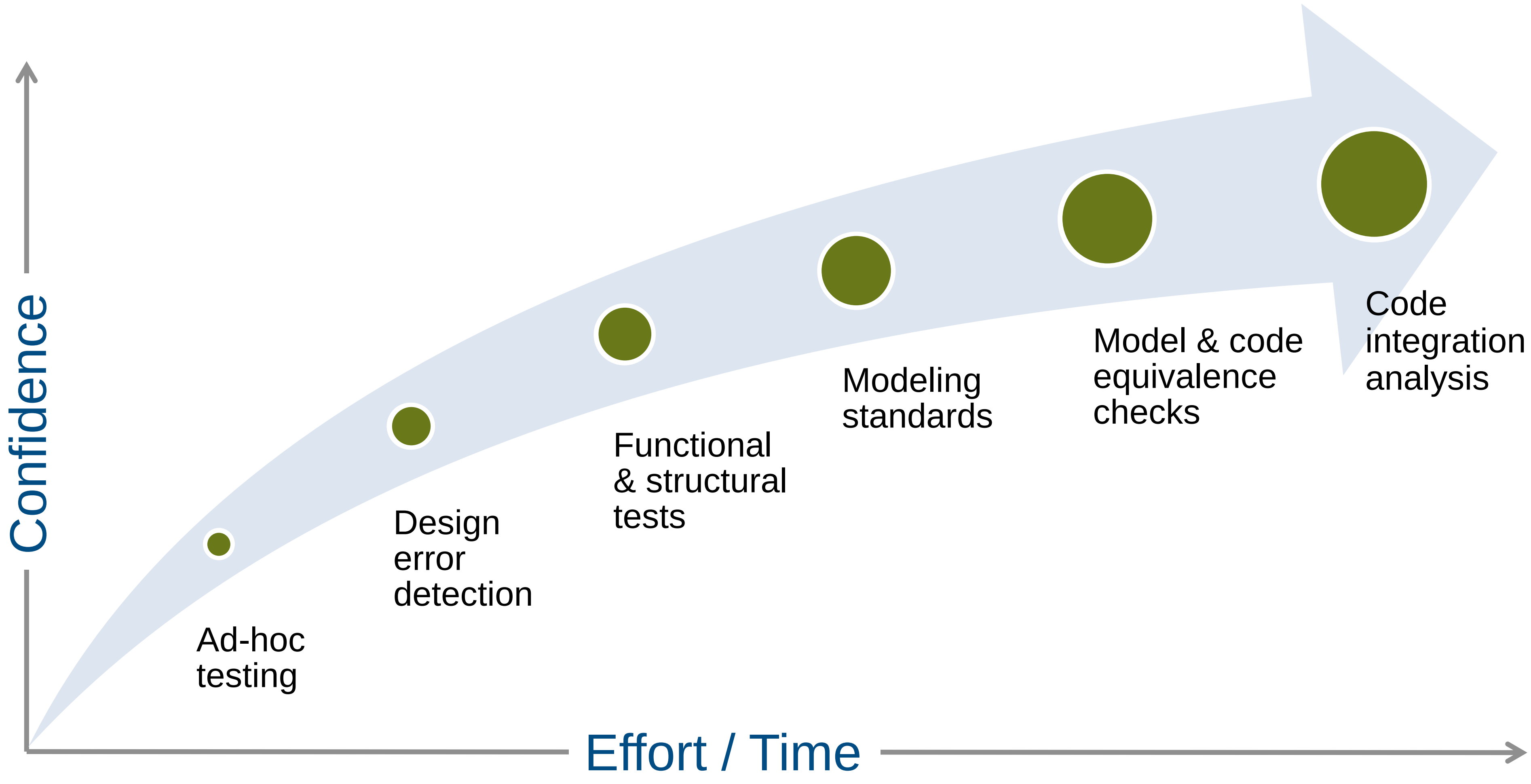
Purple: violation
MISRA-C/C++ or JSF++ code rules

Range data
tool tip

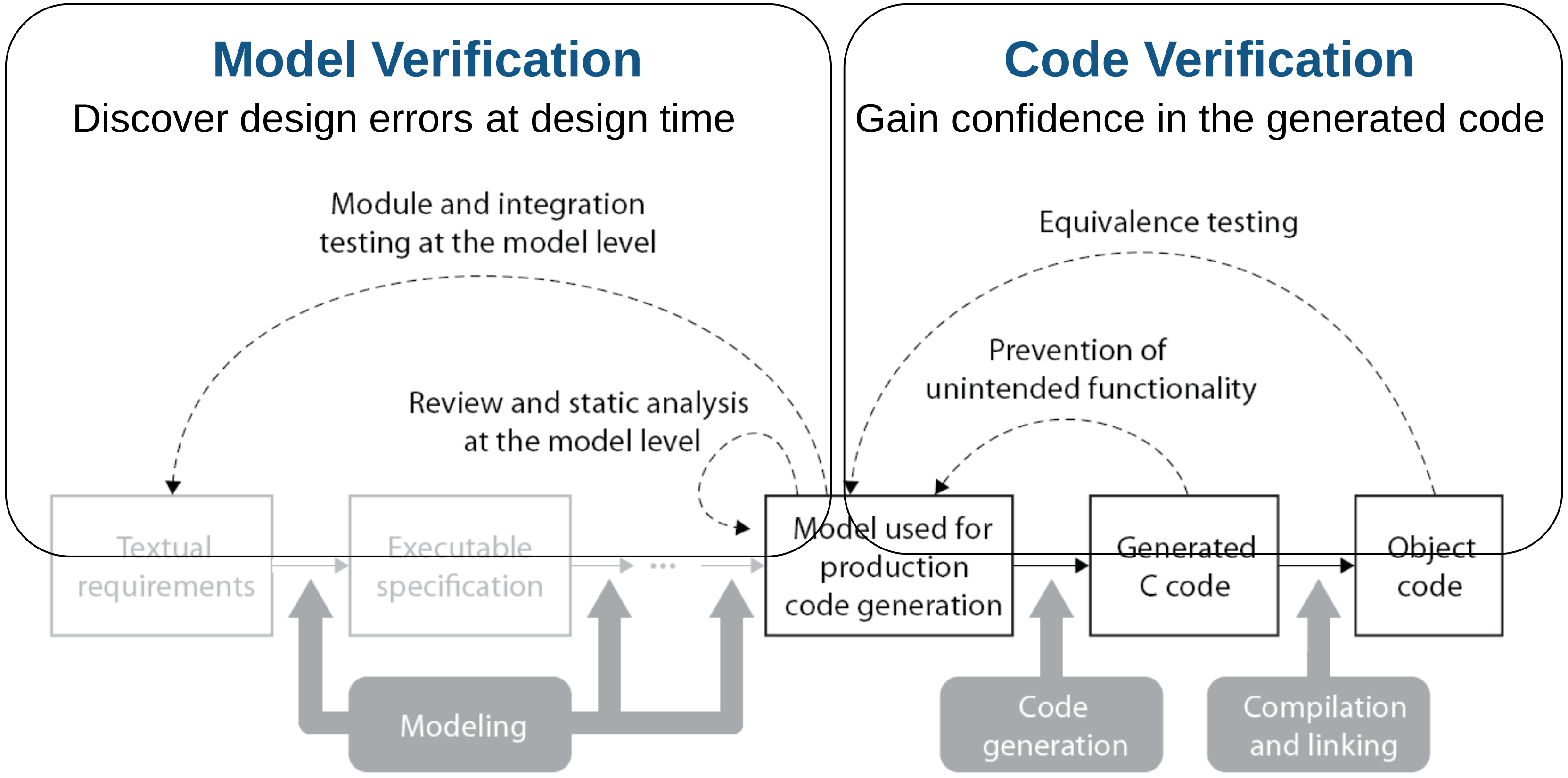
```
static void pointer_arithmetic (void) {
    int array[100];
    int *p = array;
    int i;

    for (i = 0; i < 100; i++) {
      *p = 0;
      p++;
    }

  if (get_bus_status() > 0) {
    if (get_oil_pressure() > 0) {
      *p = 5;
    } else {
      i++;
    }
  }

  i = get_bus_status();

  if (i >= 0) {
    *(p - i) = 10;
  }
}
```

variable ‘I’ (int32): [0 .. 99]
assignment of ‘I’ (int32): [1 .. 100]

Gaining Confidence in our Design



Conclusion: Model-Based Design Verification Workflow



Workflow approved by TÜV SÜD for development of safety-critical software in accordance with ISO 26262 (automotive), IEC 61508 (industrial), EN 50128 (railway), IEC 62304 (medical devices)

Conclusion

It is easier and less expensive to fix design errors early in the process when they happen.

Model-Based Design enables:

1. *Early testing to increase confidence in your design*
2. *Delivery of higher quality software throughout the workflow*

Change the world by

Accelerating the pace

of discovery, innovation, development, and learning

in engineering and science