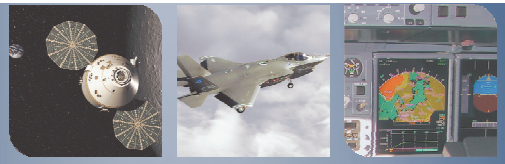
The MathWorks™ MATLAB®&SIMULINK®

Verification and Validation of Models and Code

Presenter: Mark Walker
mark.walker@mathworks.co.uk



MathWorks
Aerospace and Defence Conference '08

Agenda

- Introductions
- Workflows for verification and validation

Introductions

- I spend most of my time:

  A. Creating specifications and requirements (systems and software)

  B. Implementation based on specification and requirements created by somebody else (generating / writing / deploying / debugging code)

  C. Other (including both, or none of the above)

Demo

- How much time do we need to get 100% MC/DC coverage?

Summary | Details | Help

Coverage Report for ThrustReverserDeployLogic_harness

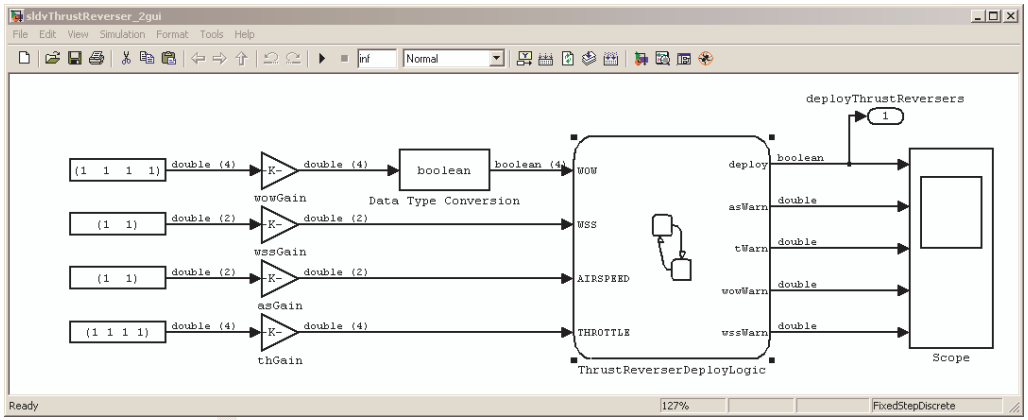
Tests

Test 1

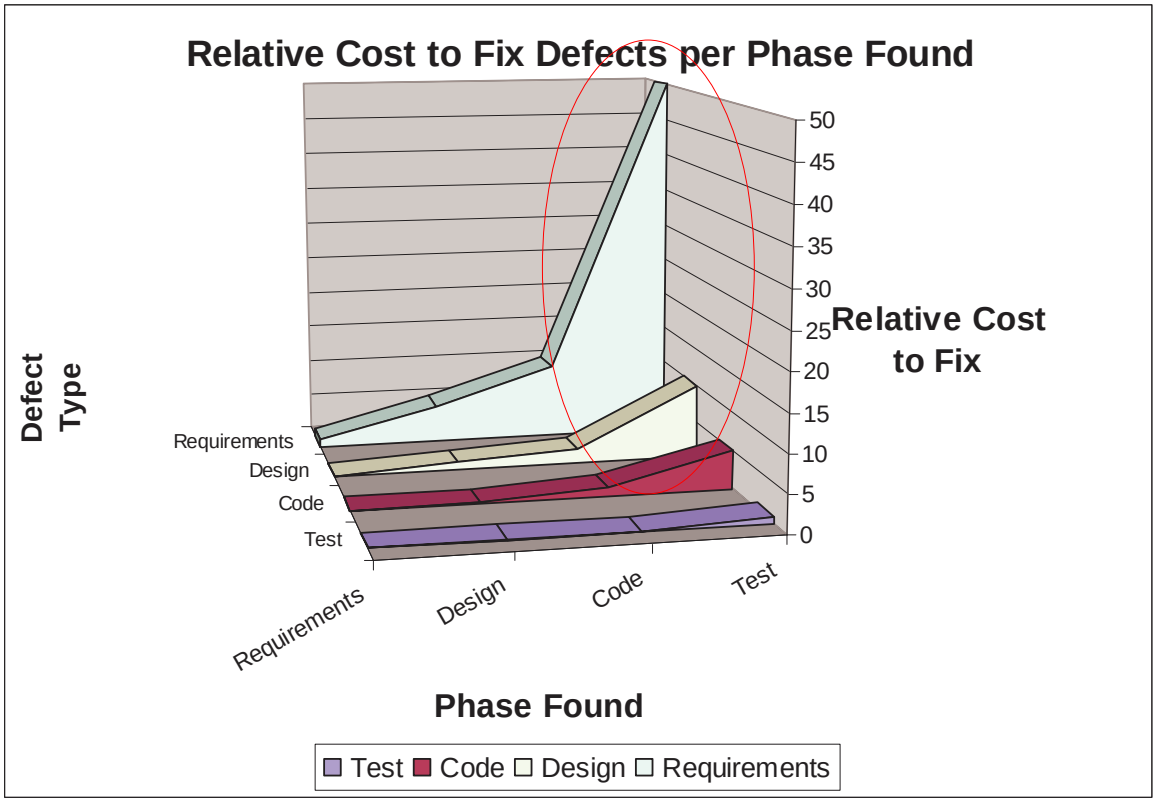
Started Execution: 17-Mar-2008 14:08:14
Ended Execution: 17-Mar-2008 14:08:14

Summary

Model Hierarchy/Complexity:		Test 1 D1	C1	MCDC
1. ThrustReverserDeployLogic_harness	46	100%	100%	100%
2. . . . Test Unit (copied from ThrustReverserDeployLogic)	45	100%	100%	100%
3. . . . . ThrustReverserDeployLogic	45	100%	100%	100%
4. . . . . SF: ThrustReverserDeployLogic	44	100%	100%	100%
5. . . . . SF: airspeedOK	2	100%	NA	NA
6. . . . . SF: isRolling	4	100%	100%	100%
7. . . . . SF: onGround	12	100%	100%	100%
8. . . . . SF: throttleOK	12	100%	100%	100%
9. . . . . SF: thrustReversers	4	100%	100%	100%
10. . . . . SF: warnings	8	100%	NA	NA
11. . . . . SF: airspeedSensors	2	100%	NA	NA
12. . . . . SF: throttleSensors	2	100%	NA	NA
13. . . . . SF: wowSensors	2	100%	NA	NA
14. . . . . SF: wsSensors	2	100%	NA	NA



Costs of Embedded Software Fault Propagation



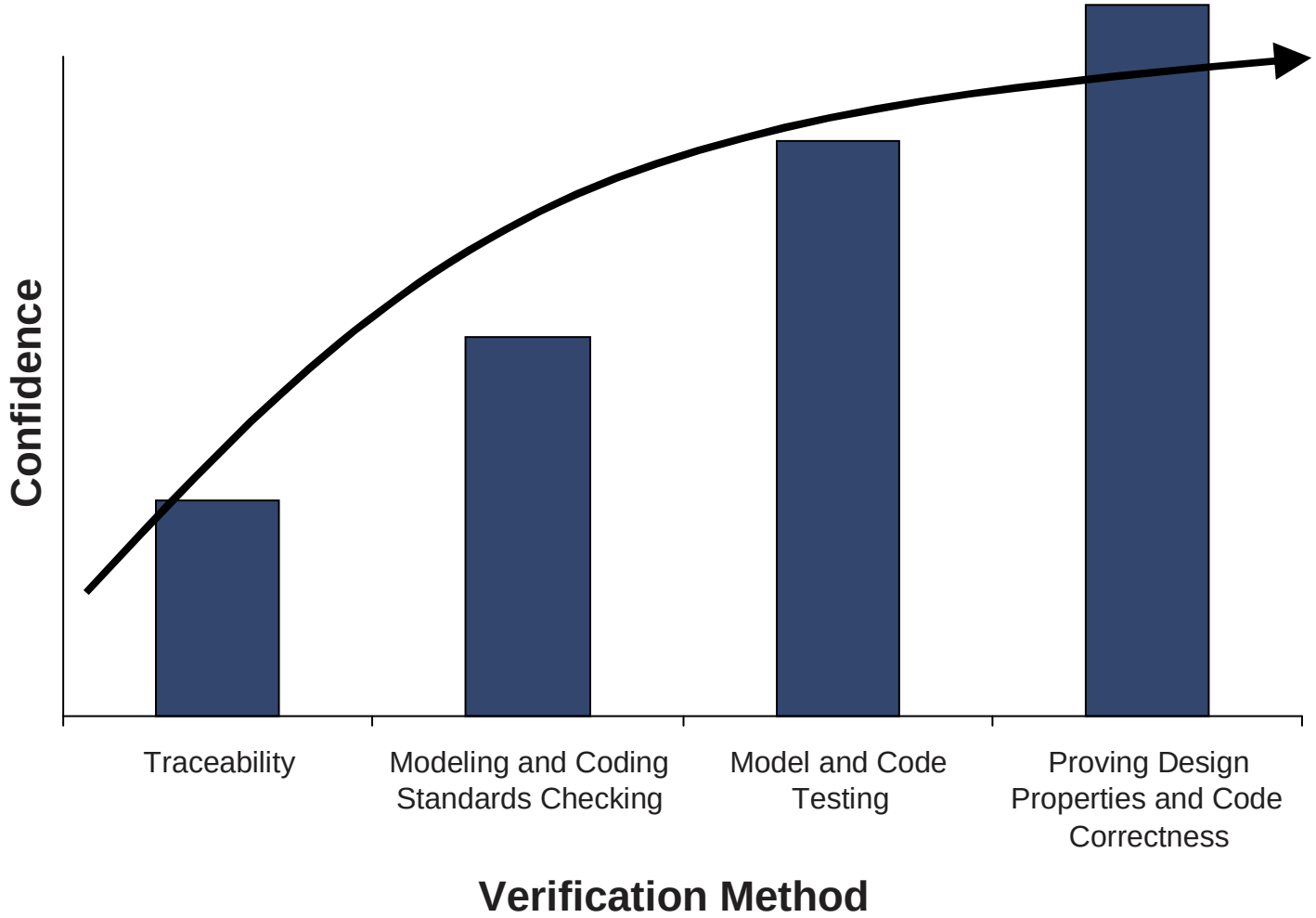
Source: Return on Investment for Independent Verification & Validation, NASA, 2004.

Cost of fixing defects detected depending on where they are introduced

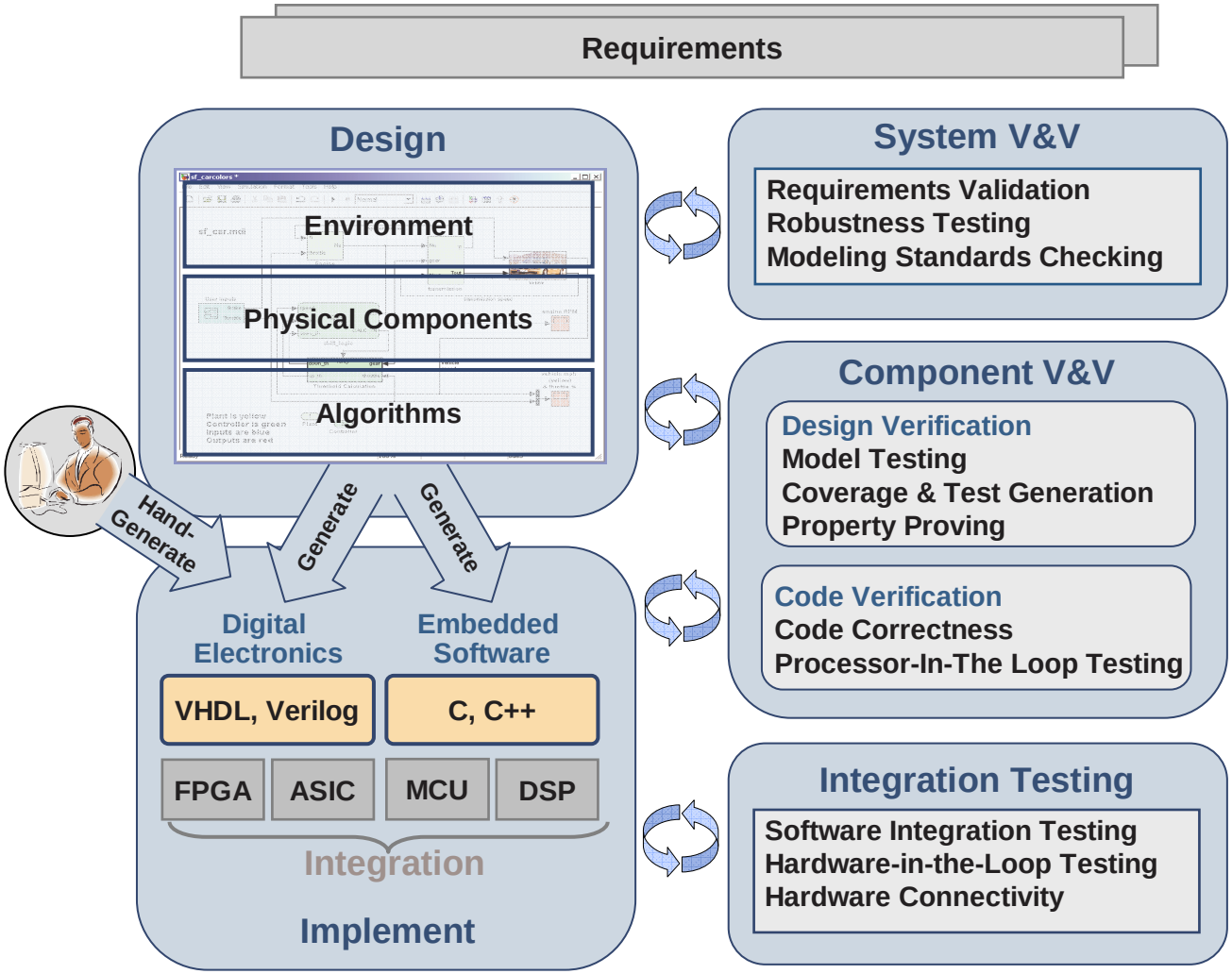
Methods for Early Verification and Validation

- Traceability
  - Requirements to model and code
  - Model to code
- Modeling and Coding Standards
  - Modeling standards checking
  - Coding standards checking
- Testing
  - Model testing in simulation
  - Processor In the loop
- Proving
  - Proving design properties
  - Proving code correctness

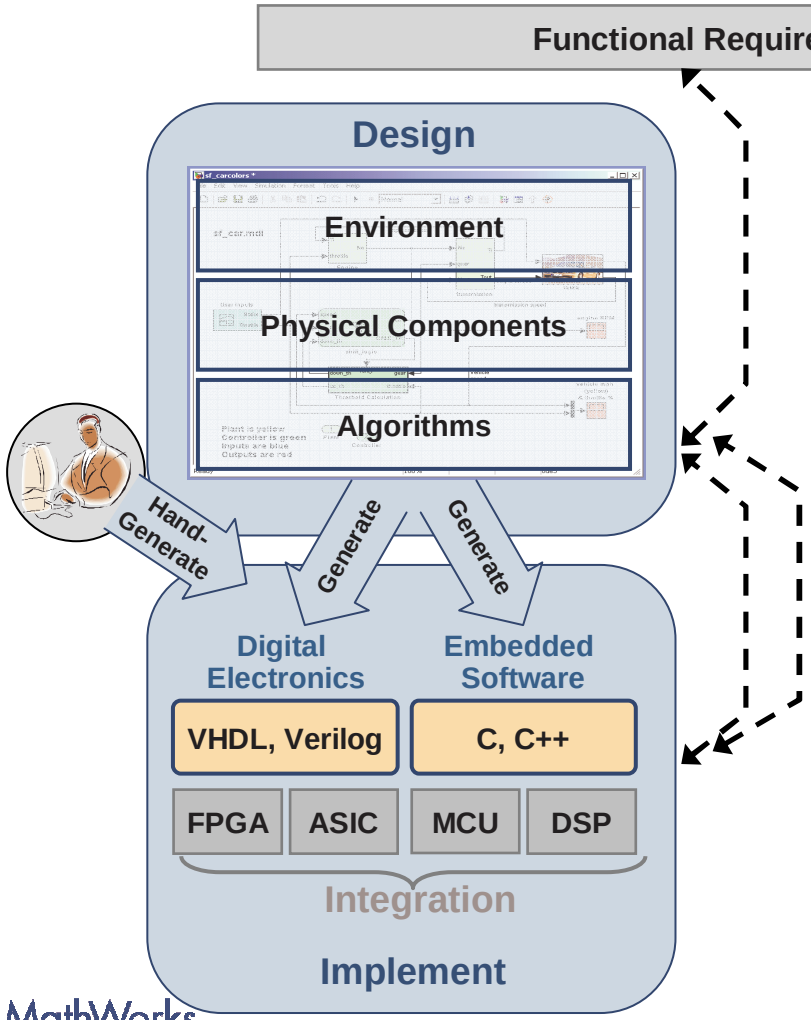
Increasing Confidence In Your Designs



Address the Entire Development Process

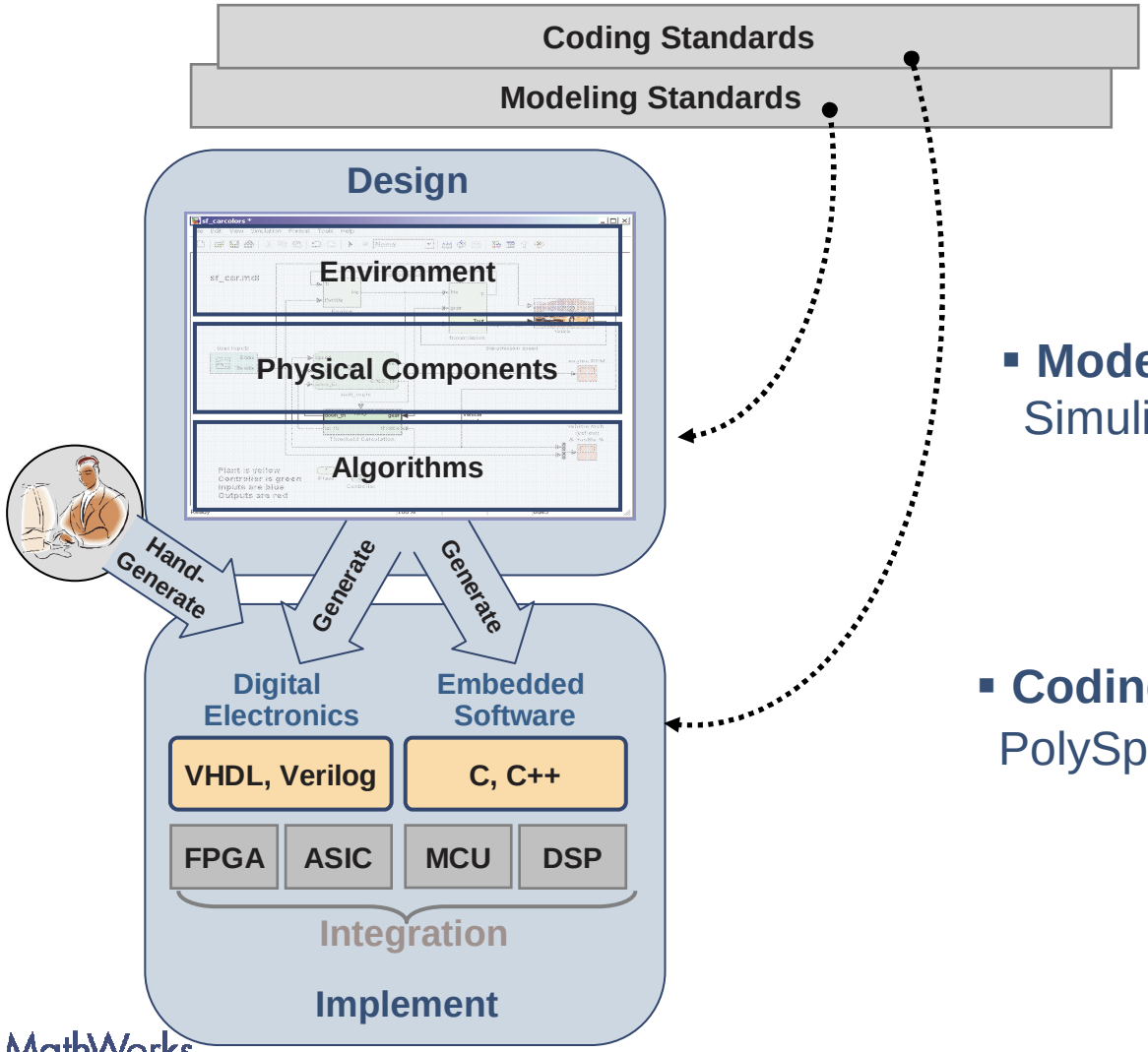


Traceability



- **Tracing Requirements⇔Model**
  Simulink® Verification and Validation™
- **Tracing Model⇔Source Code**
  Real-Time Workshop® Embedded Coder™
- **Tracing Requirements⇔Source Code**
  Simulink Verification and Validation

Modeling and Coding Standards



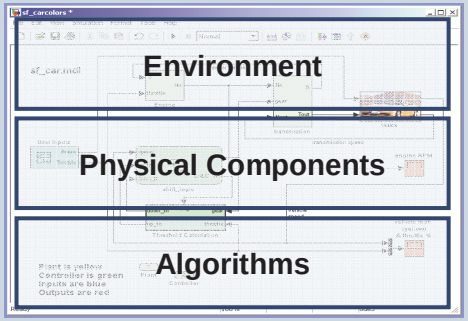
- **Modeling Standards Checking**
  Simulink Verification and Validation

- **Coding Standards Checking**
  PolySpace™ Client™ for C/C++

Early Validation and Robustness Testing

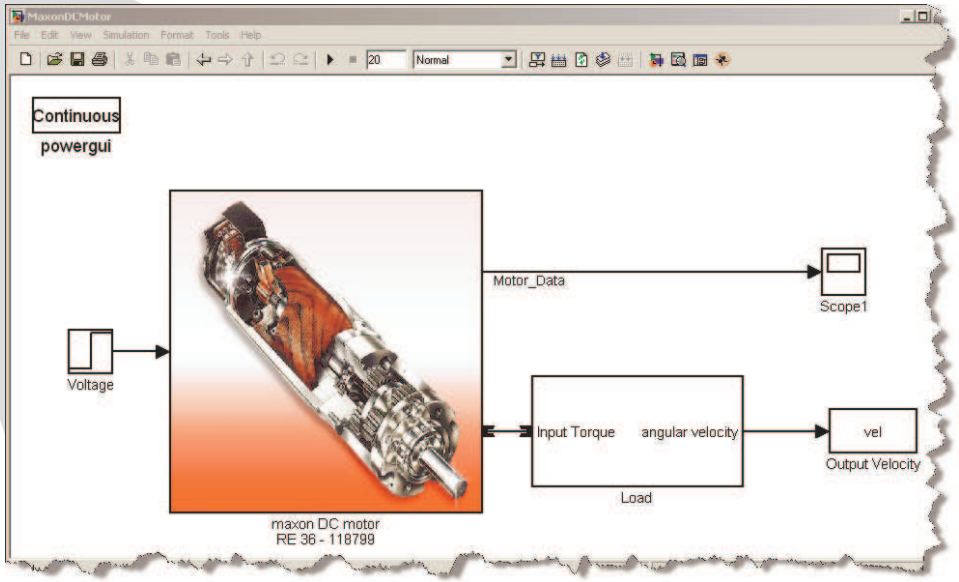
Requirements

Design

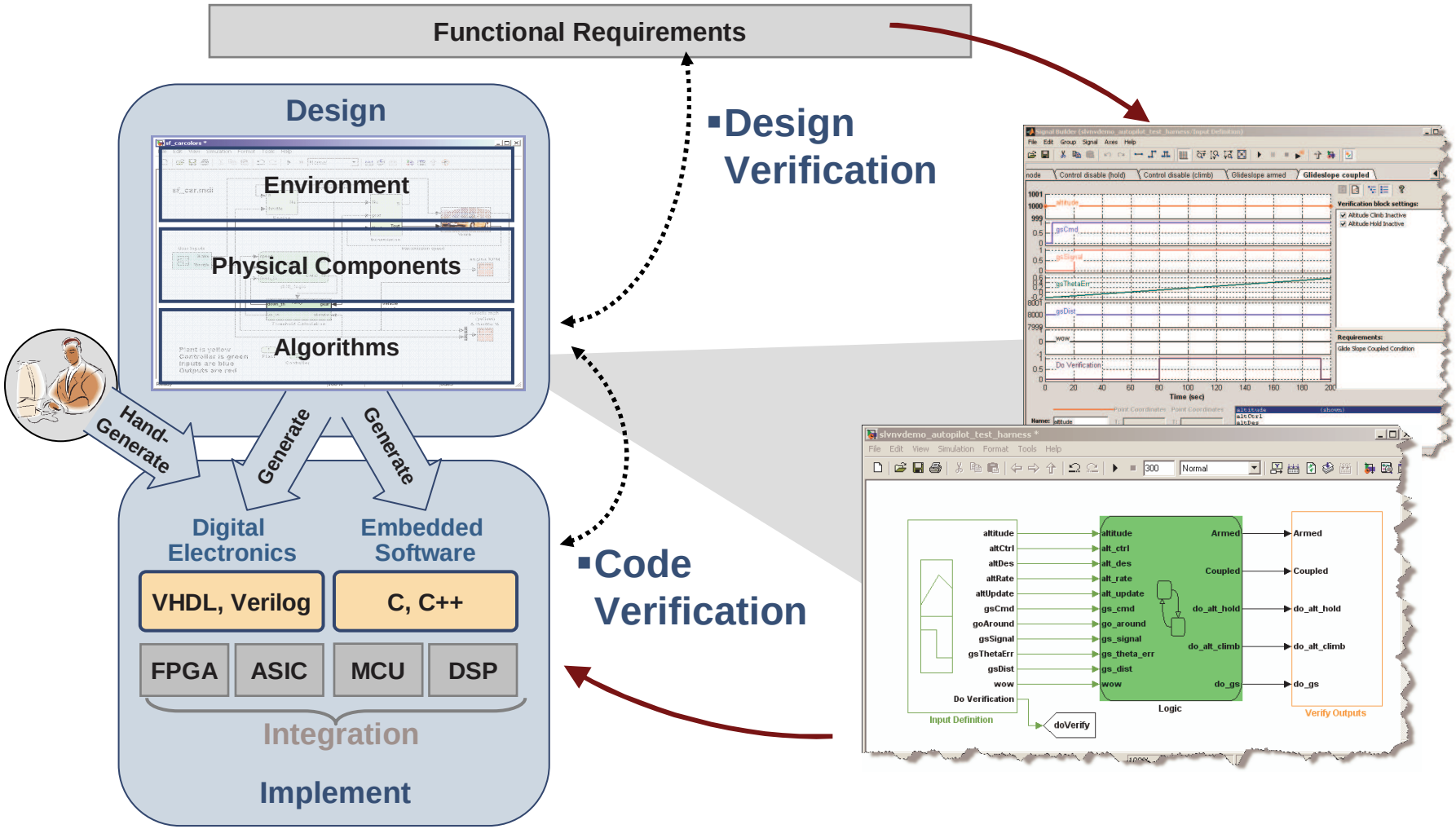


System V&V

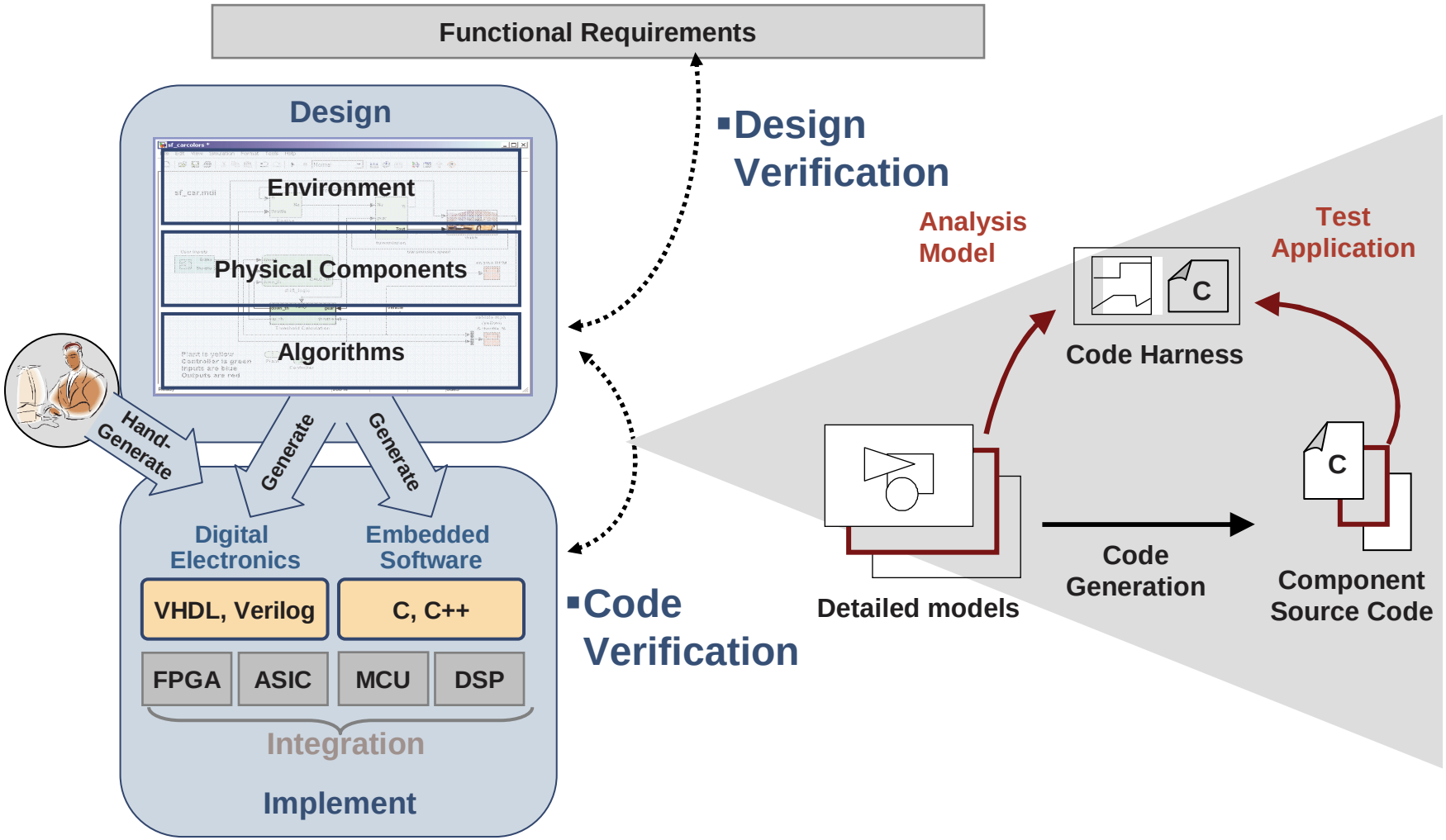
- Requirements Validation
- Robustness Testing
- Modeling Standards Checking



Component Testing



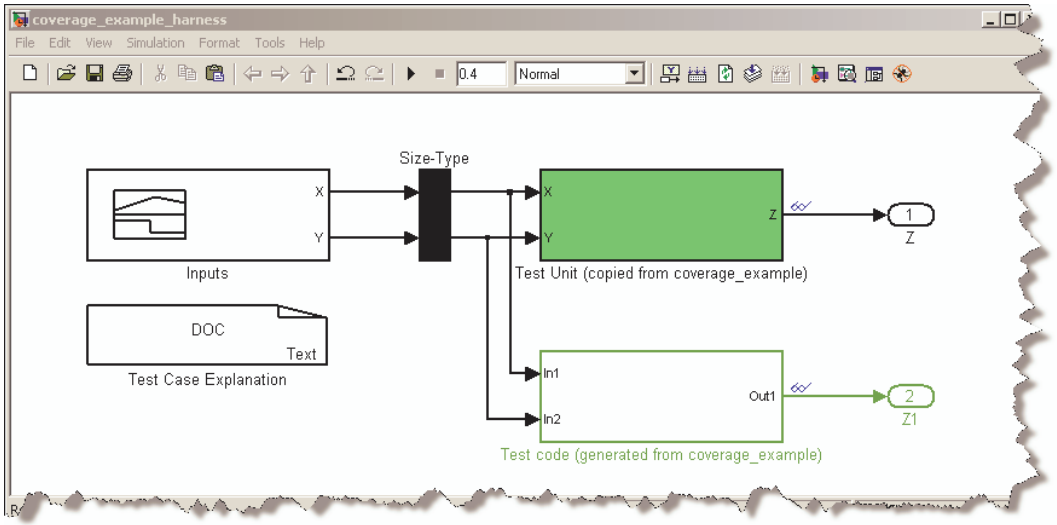
Test Generation Workflow



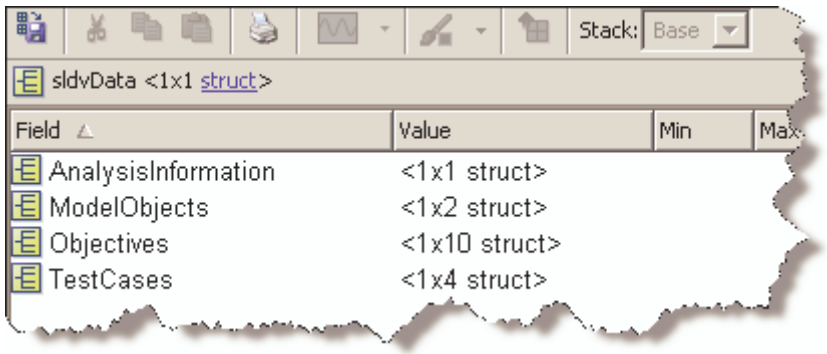
Code Testing with Generated Signals

Simulink

- Software-in-the-loop
  - On the host
- Processor-in-the-loop
  - On the target processor

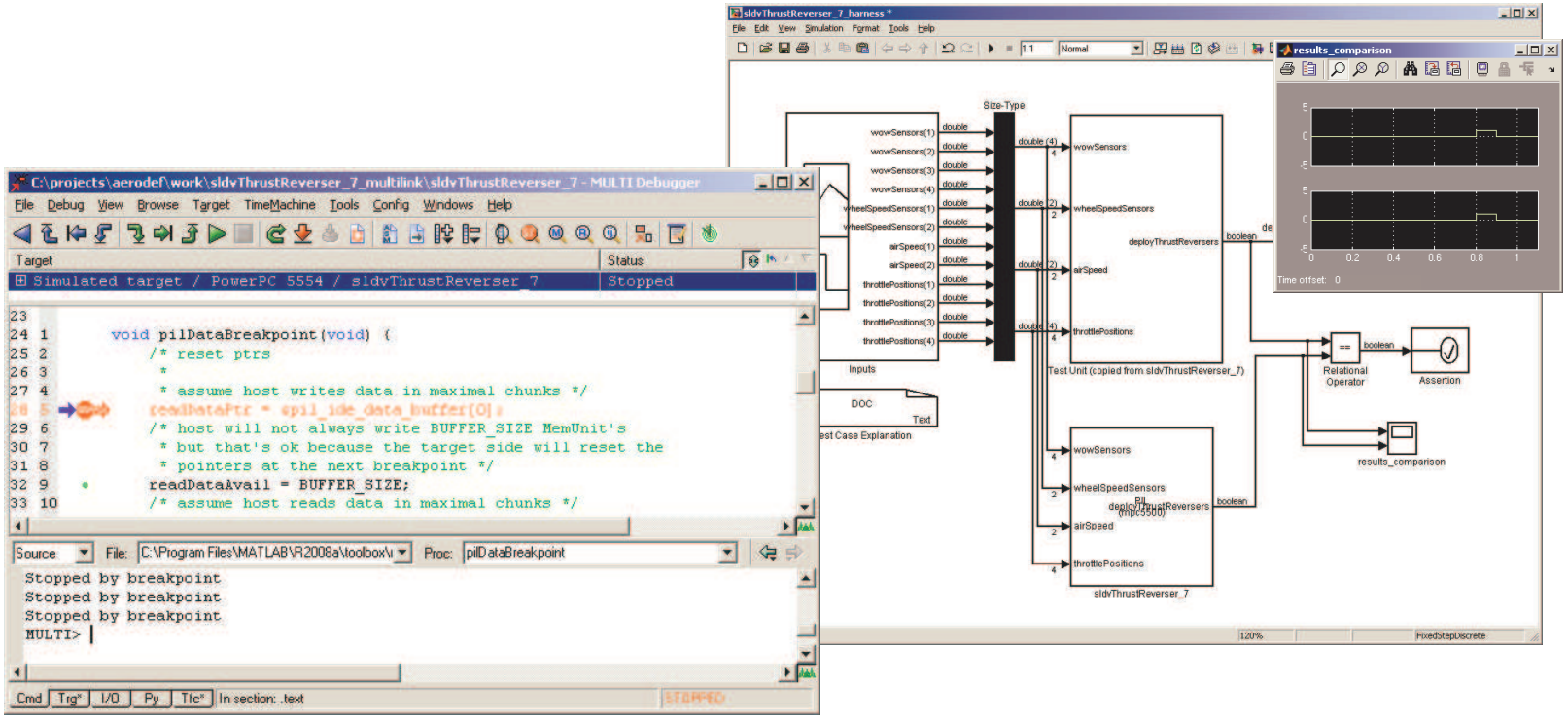


- Independent code testing environment
  - Generated signals and model outputs are saved as a .mat data file
  - Exported input signals drive code tests
  - Exported model outputs become expectation values for code testing

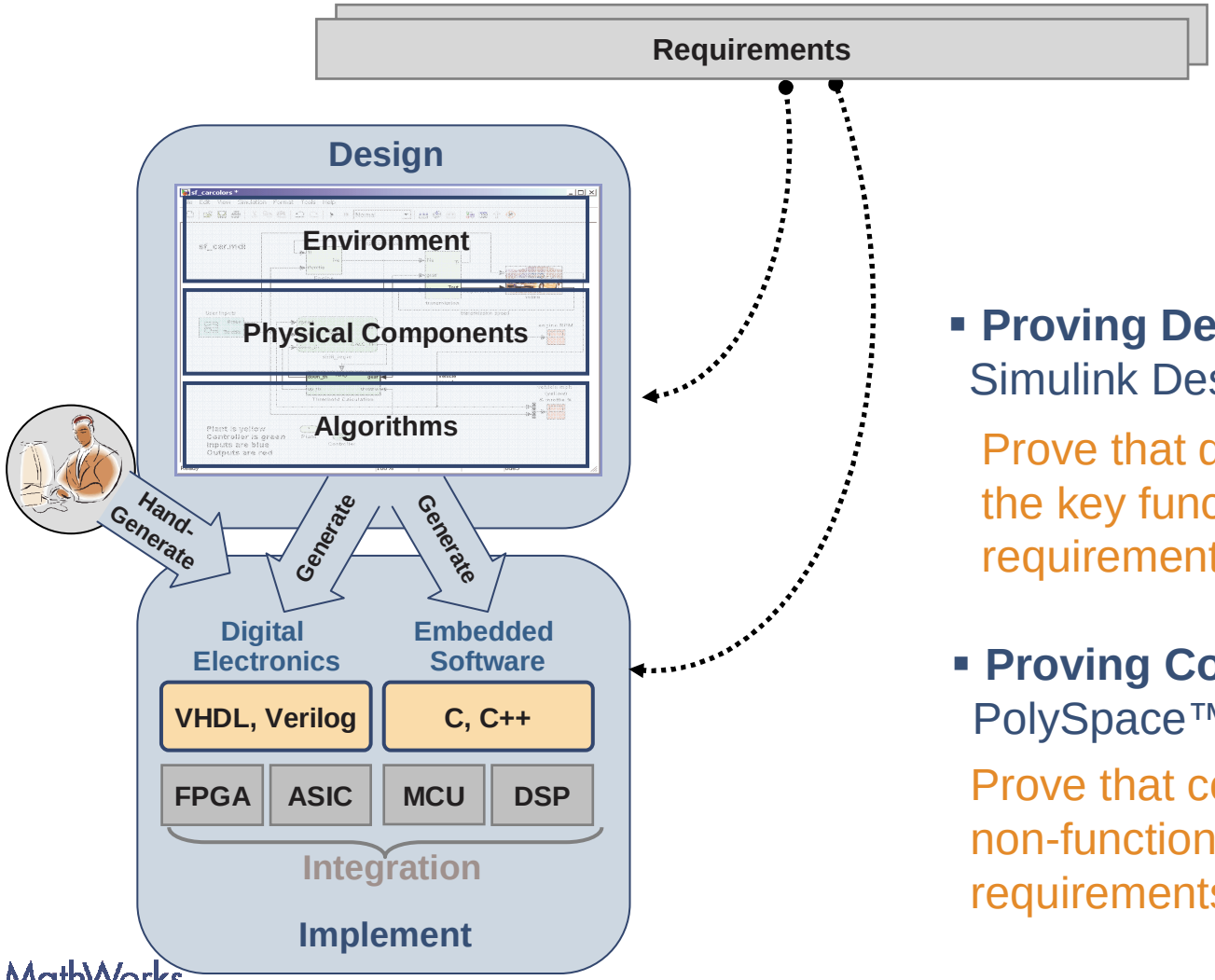


Demo

- Processor-in-the-loop co-simulation



Proving



- **Proving Design Properties**
  Simulink Design Verifier

  Prove that design meets the key functional requirements

- **Proving Code Correctness**
  PolySpace™ Server for C/C++

  Prove that code meets non-functional runtime requirements

Code Correctness

Formal method:
Abstract Interpretation

```
static void Pointer_Arithmetic ()
{
  int tab[100];
  int i, *p = tab;

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

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

  i = random_int();
  if (random_int())  *(p-i) = 10;

  if (0<i && i<=100)
  { p = p - i;
    *p = 5;      /* Safe pointer access */
  }
}
```

Proven

Green reliable

Red faulty

Grey dead

Orange unproven

Green reliable

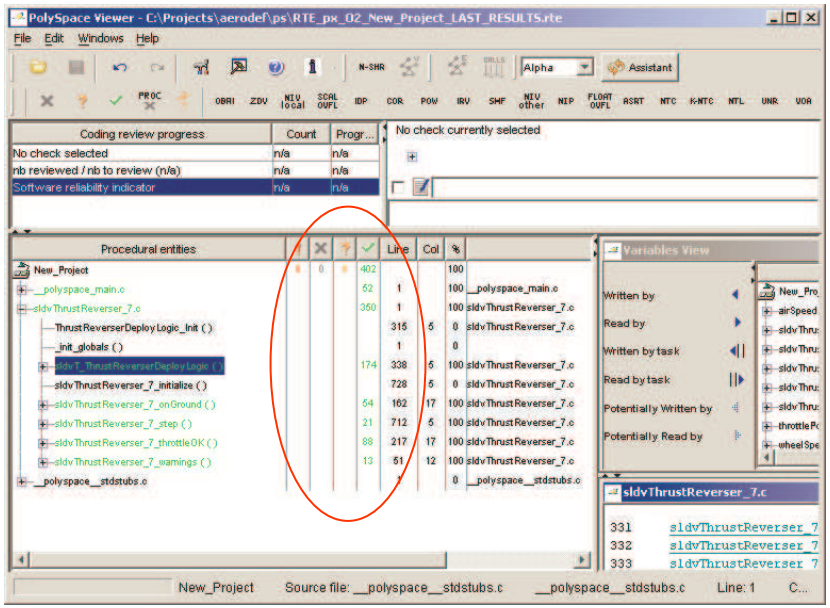
Green reliable

Green reliable

Results are proven for all possible executions of the code!!

Code Correctness

- A model is a well controlled way to specify system behaviour
  - Generated code matches the model
  - Few ambiguities, low warning rate
  - 100% green is a realistic target



Demo

- Proving a functional requirement

Chapter 1. Summary

Input Model

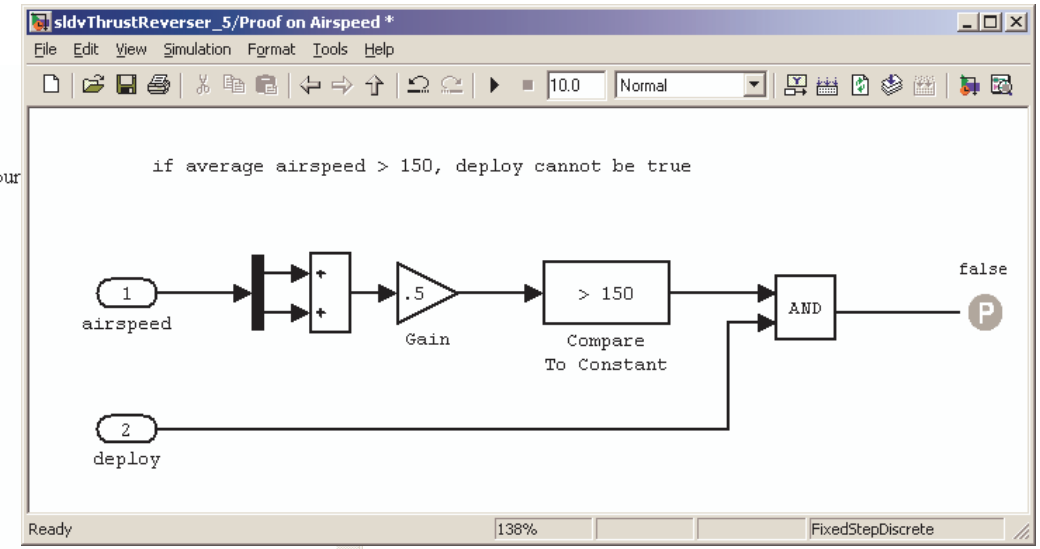
File:	C:\perforceWS\main\demos\sldvThrustReverser\sour
Version:	1.53
Time Stamp:	Fri Nov 09 19:25:32 2007
Author:	cstephen

Analysis Information

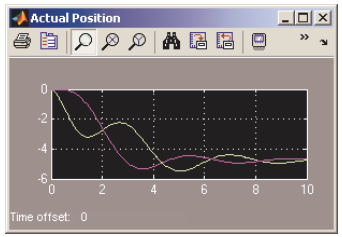
Design Verifier Version:	1.2
Total Analysis Time:	26 secs
Status:	Completed normally
Approximations:	1
Objectives Proven Valid:	4
Objectives Falsified with Counterexamples:	0
Objectives Falsified - No Counterexample:	0
Objectives Undecided:	0
Objectives Producing Errors:	0

Output Files

C:\perforceWS\main\demos\sldvThrustReverser\source\sldv_output\sldvThrustReverser_5

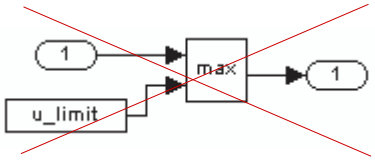


Example Problems vs. Tools



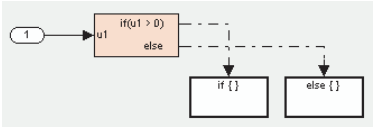
- Incorrect Dynamic Response
  - Simulation Testing
  - Rapid Prototyping and Hardware-in-the-Loop

- Model Error: max(a,b) instead of min(a,b) to apply upper clip
  - Simulation Testing
  - Property proving with Simulink Design Verifier



Example Problems vs. Tools

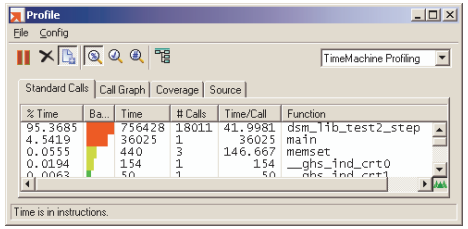
- Unreachable state / transition / code
  - Test generation with Simulink Design Verifier
  - PolySpace



- Overflow / underflow
  - Simulation
  - PolySpace

```
111   .....   /* Sum: '<S1>/Sum3' */
112   UOVFL   rtb_Sum3 = rtb_Product2 + rtb_Gain2_e;
113   .....
114   .....   /* Saturate: '<S1>/Saturation1' */
115   .OK..   tmp = rtb_Sum3;
```

- Execution time exceeds deadline
  - Simulation (requires execution time model)
  - Processor-in-the-loop



Summary

- Model-Based Design enables early verification and validation!
- Early verification and validation methods improve and optimize your existing development process.
- Early problem detection significantly reduces time spent debugging – shorter time to resolution

Master Class Invitation

- Methods for Early Verification and Validation
  - Robustness Testing
  - Automatic Test Generation
  - Property Proving