Shift left with model-based testing of systems models!

Todd Dunnavant, PhD, PE – IBM
twdunnav@us.ibm.com

Todd Dunnavant, PhD, PE Bio

For the past 27 years, Todd's personal mission has been "to make the world safe for systems and software engineers". The journey along this path began in 1997, when Todd joined Rational Software as a field tech rep after being a customer for three years. Todd joined IBM in 2003, following IBM's acquisition of Rational. Being with IBM has enabled Todd to help a wider range of customers, including those in the US Federal Government, Aerospace & Defense, Automotive, Financial, Insurance, and Medical Devices sectors.

Todd's roles within IBM have included global lead of IBM's Architecture, Design, and Construction Community of Practice, global technical lead and principal architect for IBM's Engineering Lifecycle Management (ELM) services practice, and global lead for IBM's ELM Community of Practice. He has served for the past 3.5 years as an Executive IT Specialist and Solution Engineer in IBM's Federal Market. His primary focus has been to support the Federal System Integrators, NASA, and the Department of Energy's national labs in their adoption and use of IBM ELM. He also serves on the internal team that defines IBM's strategy and product roadmaps for model-based engineering.

Contents

- Engineering challenges driving the need for change
- Putting the foot on the accelerator with Model Based Systems Engineering
- General workflow for model-based requirements V&V
- Shifting left using model animation and system simulation
- Shifting left using model-based testing

“Document centric” engineering practices are challenged to keep up

- Lack traceability, versions & variants management, governance and testability

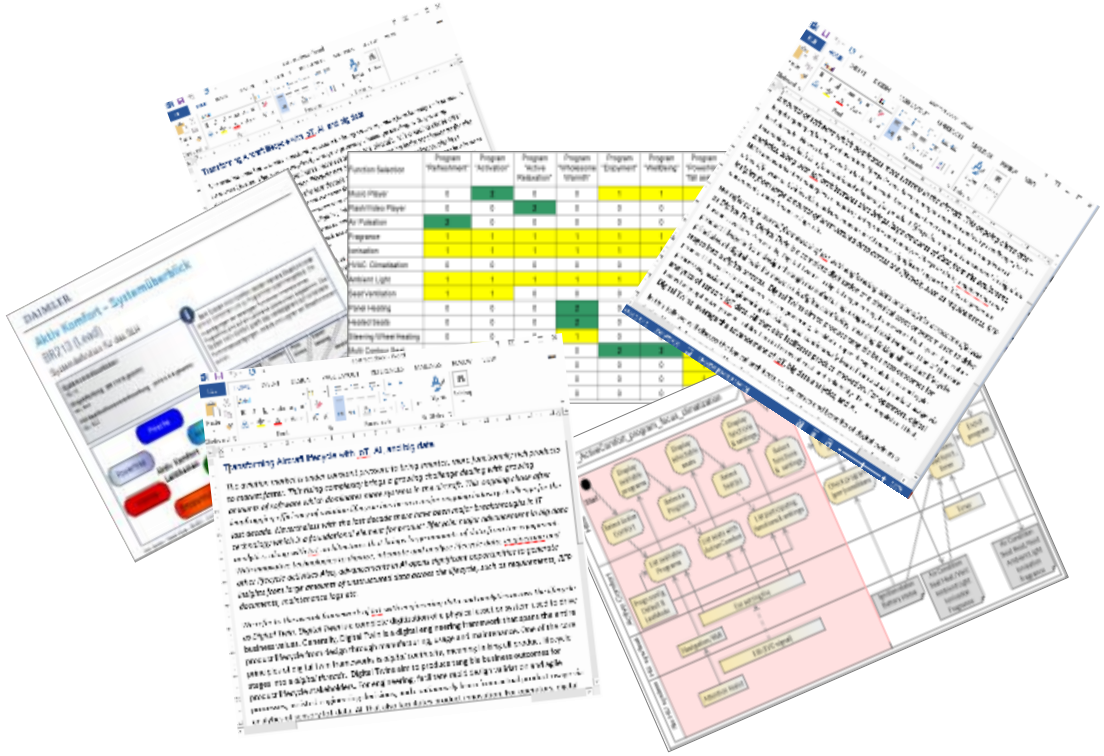
– Expensive rework due to late discovery of issues

– Slow response to bids and change requests

– Inefficient supplier collaboration

– Non-optimal designs due to early design lock

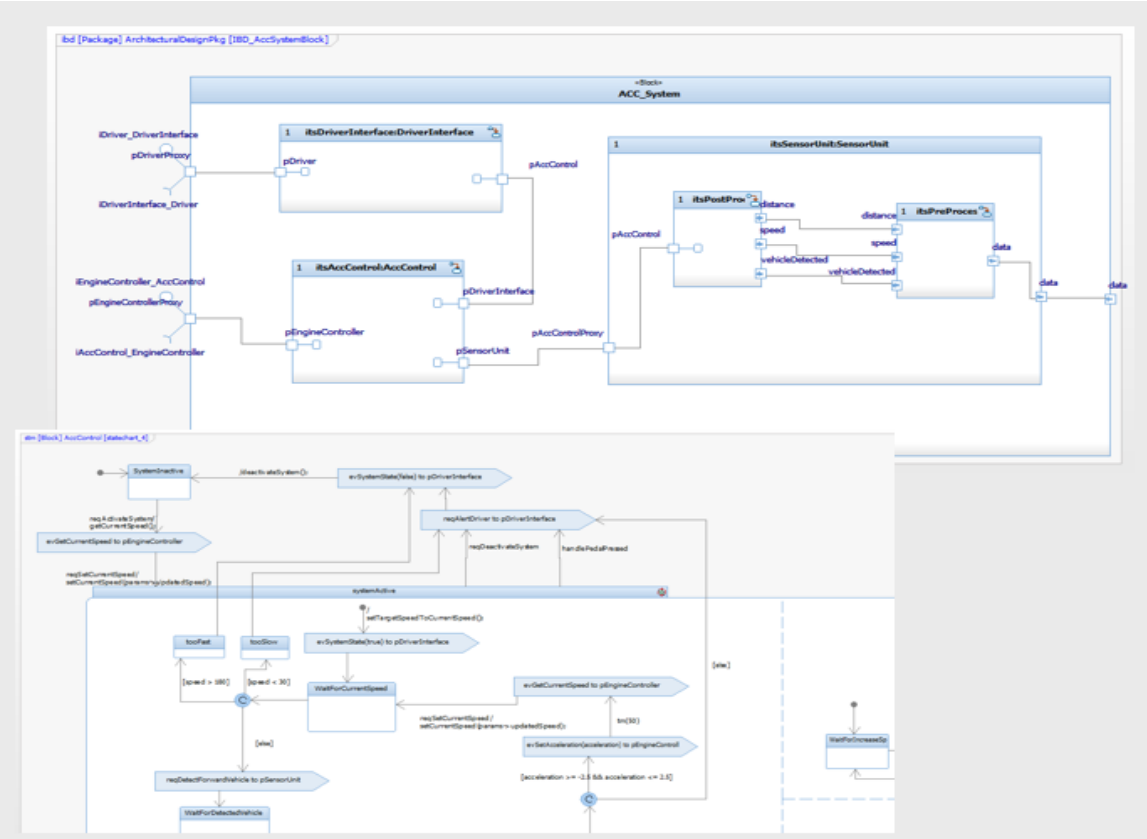
– High costs of regulatory compliance activities



Opportunity: Digital representation of the system using *Model Based Systems Engineering (MBSE)*

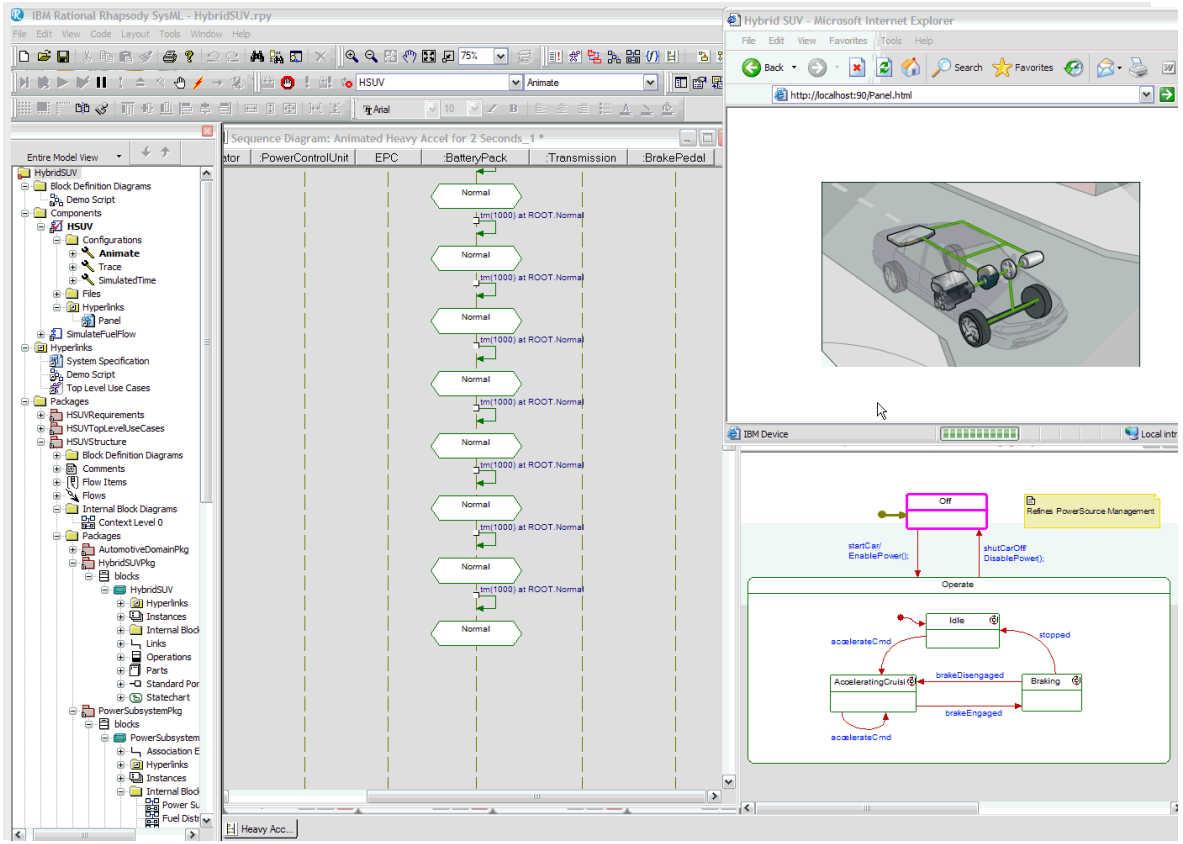
– Precise: with standards-based modeling language for systems engineering

– Visual: “a picture is worth a thousand words”, simplifies communication



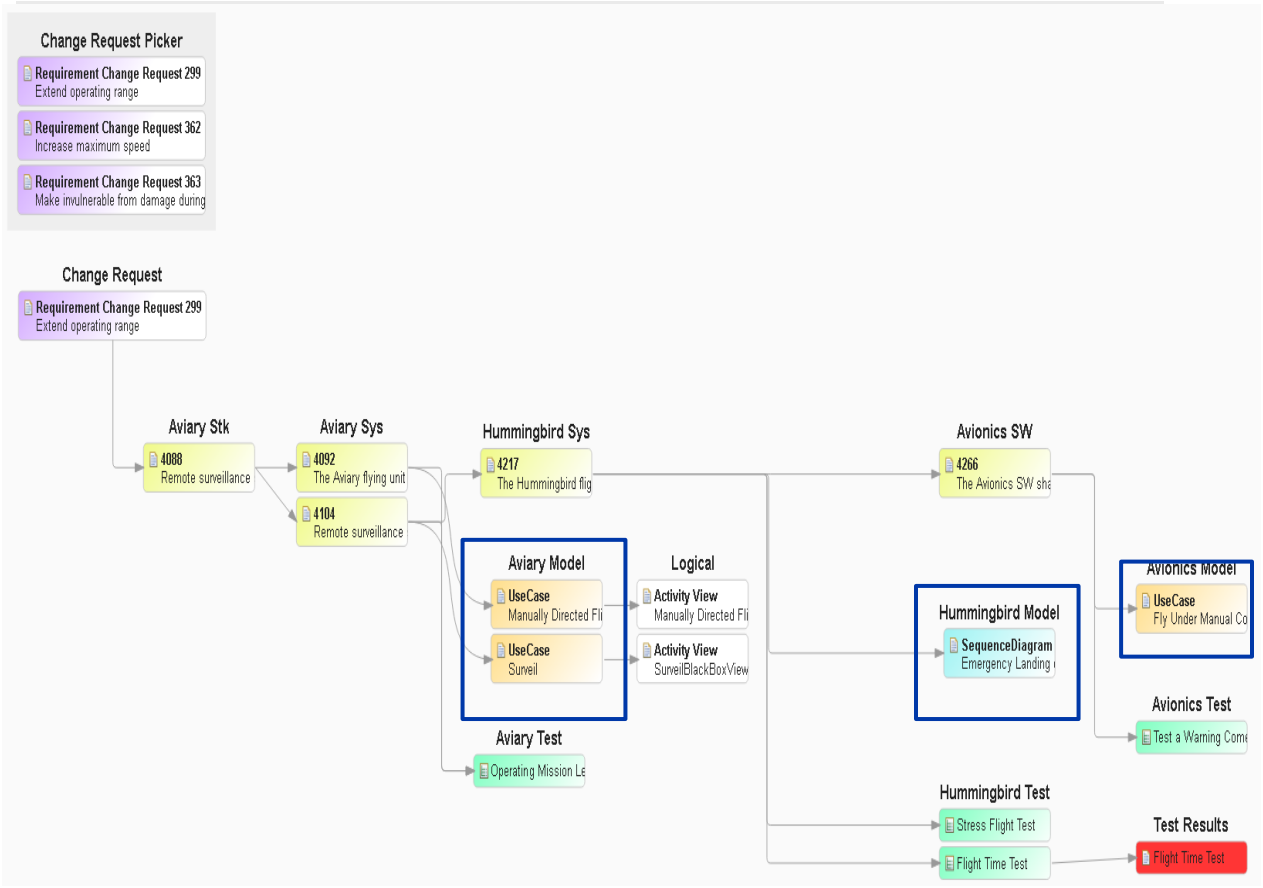
Opportunity: Digital representation of the system using *Model Based Systems Engineering (MBSE)*

- Precise: with standards-based modeling language for systems engineering
- Visual: “a picture is worth a thousand words”, simplifies communication
- Verifiable: models are formal artifacts that can be verified & validated against the requirements



Opportunity: Digital representation of the system using *Model Based Systems Engineering (MBSE)*

- Precise: with standards-based modeling language for systems engineering
- Visual: “a picture is worth a thousand words”, simplifies communication
- Verifiable: models are formal artifacts that can be verified & validated against the requirements
- Traceable: models are based on fine-grained digital artifacts traced to other essential lifecycle artifacts such as requirements, work items, and test artifacts



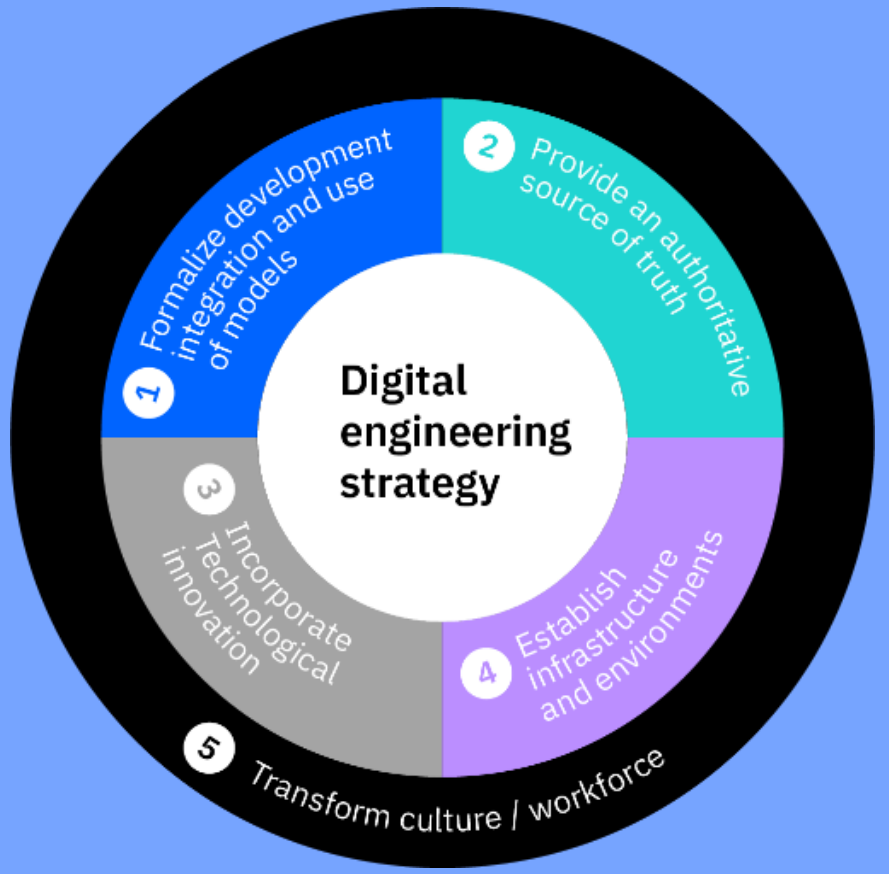
MBSE: Advocated by US Department of Defense

*"**conduct engineering in more integrated virtual environments** to increase customer and vendor engagement, improve threat response timelines, [..], reduce cost of documentation and impact sustainment affordability.*

Such engineering environments will allow DoD and industry partners to evolve designs at conceptual phase, reducing the need for expensive mockups, premature design lock, and physical testing." [1]

2018, the Office of the Under Secretary of Defense for Research and Engineering (USD(R&E)) released a Digital Engineering Strategy built on five foundational elements necessary **for a Digital Engineering Ecosystem to thrive.**

1. DoD Digital Engineering Initiative: https://www.acq.osd.mil/se/initiatives/init_de.html



"Systems Engineering focuses on ensuring the pieces work together to achieve the objective of the whole"

Systems Engineering Body of Knowledge (SEBoK)

MBDA, a European defense company, experienced the benefit of MBSE in accelerating systems delivery

Reduced typical system design time from

60+ months

to

18 months

while enabling design team collaboration across geographies

Business problem: Needed to improve complex engineering design collaboration and cycle time across geographical borders

Solution: A model-based systems engineering platform across widely distributed design and engineering teams to analyze and communicate software, mechanical and electrical requirements

"We're now able to define and model all the requirements of a complex missile system very early in the development process. This differentiates our delivery capabilities in a highly competitive global marketplace."

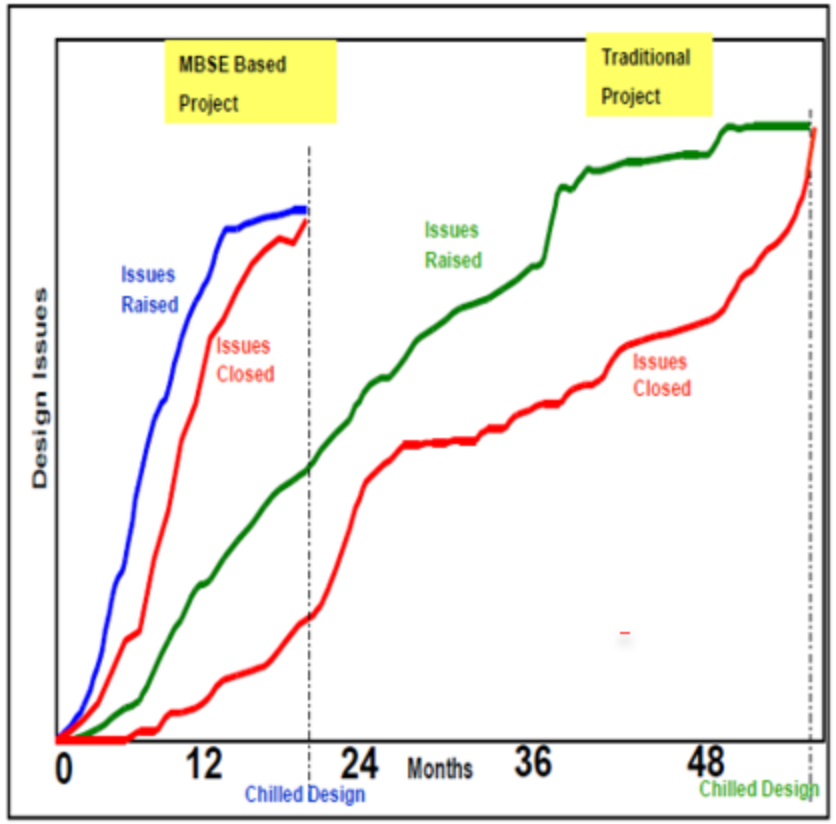
— *MBDA executive*

MBSE delivered higher-quality results, sooner

- **Results showed that teams which use MBSE approaches are more effective at delivering higher-quality products, sooner**

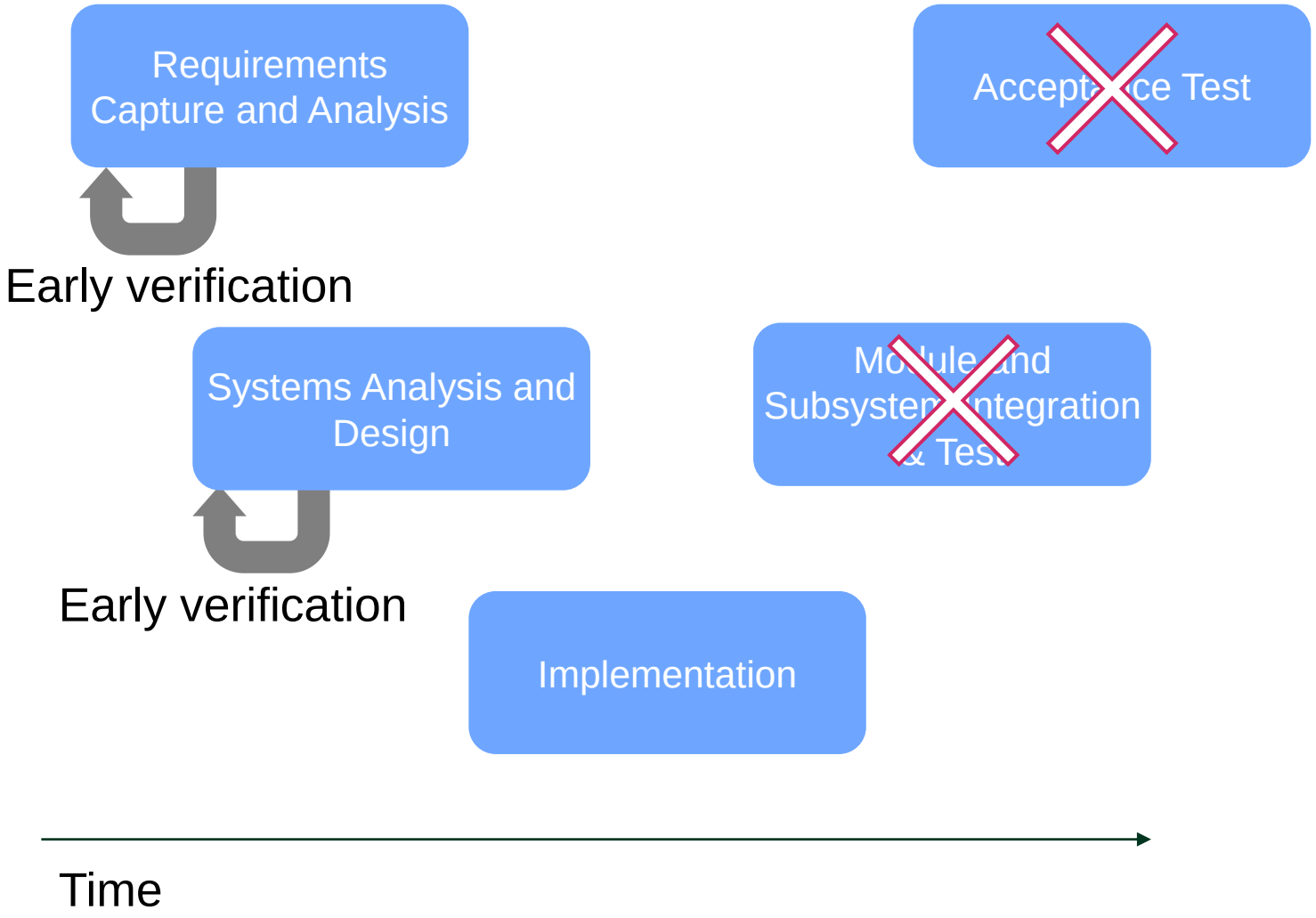
- MBSE project
  - Close collaboration with customers
  - Facilitated workshops based around prioritized Use Case Analysis
  - Issues resolved within days/weeks

- Traditional (textual requirements)
  - Regular customer design reviews
  - Design studies main source of issue resolution
  - Issues resolved within weeks/months
  - Several Design issues remained at "Design Chill"



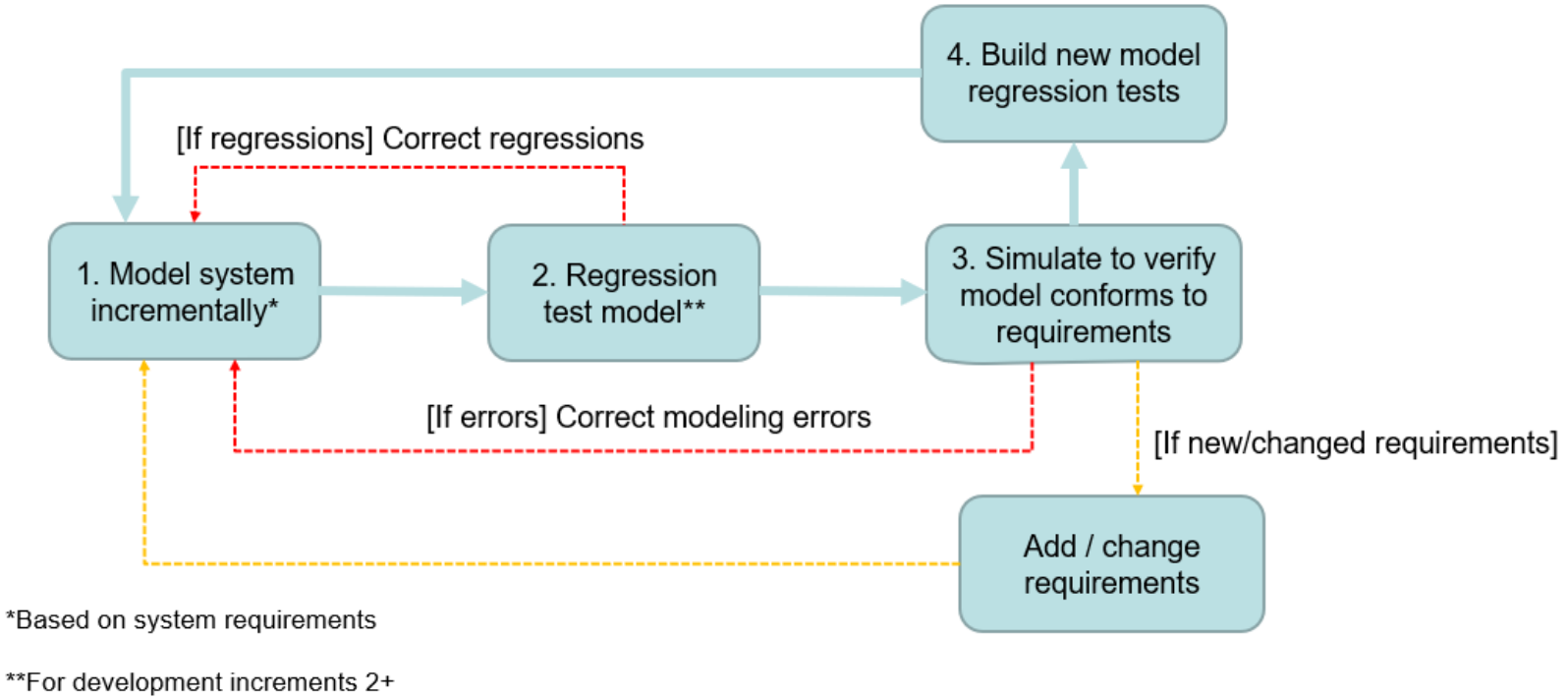
In addition to improving communication, models enable early verification and validation of complex systems – shift V & V left!

- Finding design errors during physical system integration is costly and inflicts significant delays
- System design models are testable virtual representations that identify requirements and design issues before anything physical is built
- Models are elaborated and verified iteratively enabling adoption of agile principles to systems engineering



General approach for model-based requirements V&V

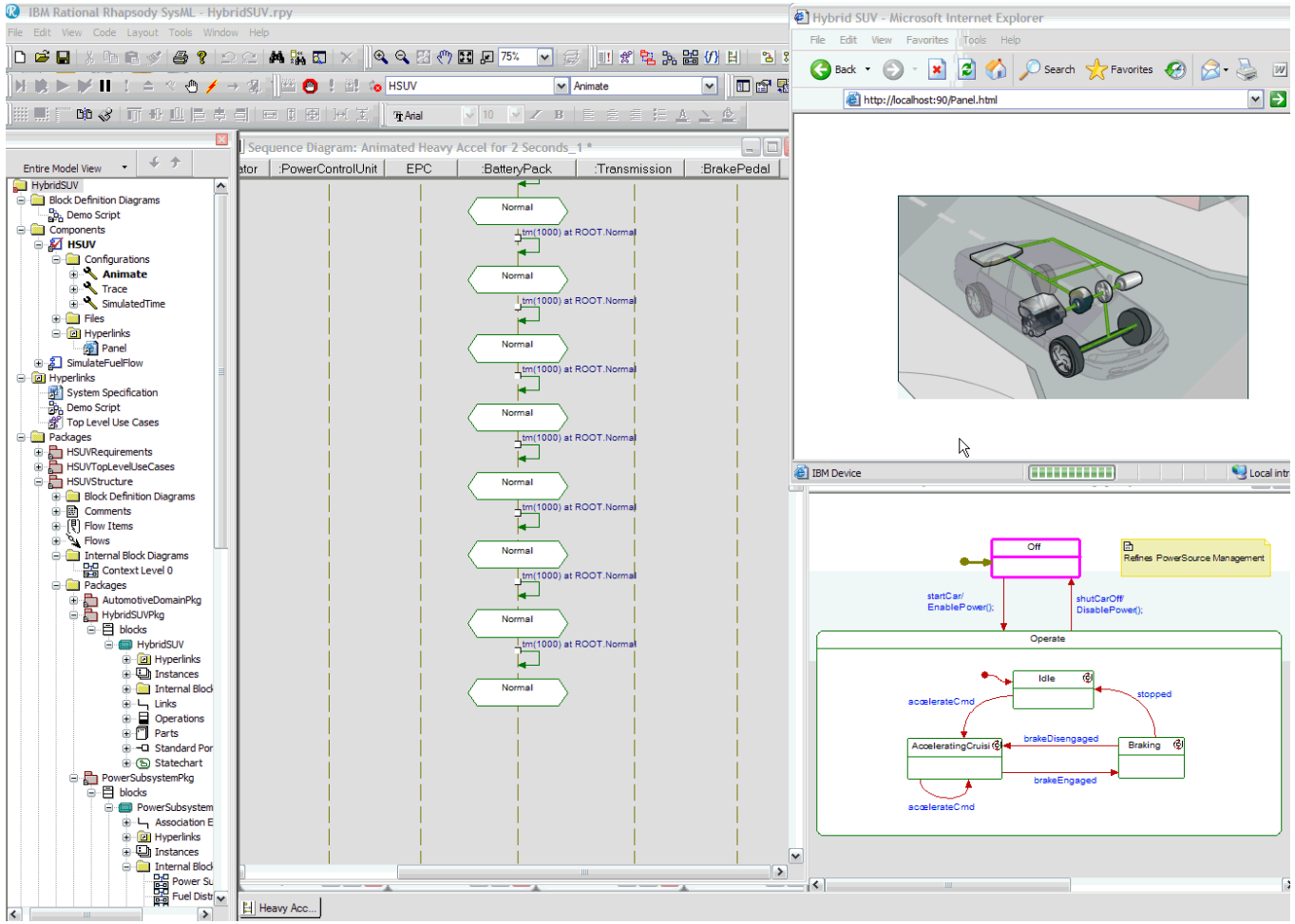
- **Adopt an incremental approach to systems modeling and performing model-based requirements V&V**
  - **At IBM, we refer to this as *Agile Model Based Systems Engineering***



- **We will talk next about Step 3, then address Steps 2 and 4 later**

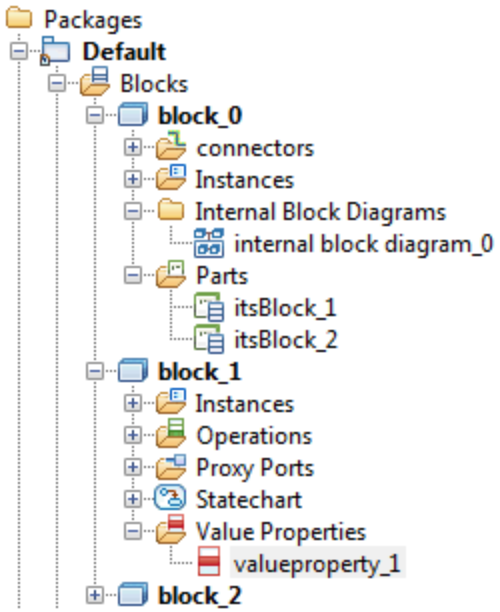
Step 3: Shifting Left with Model Animation and System Simulation

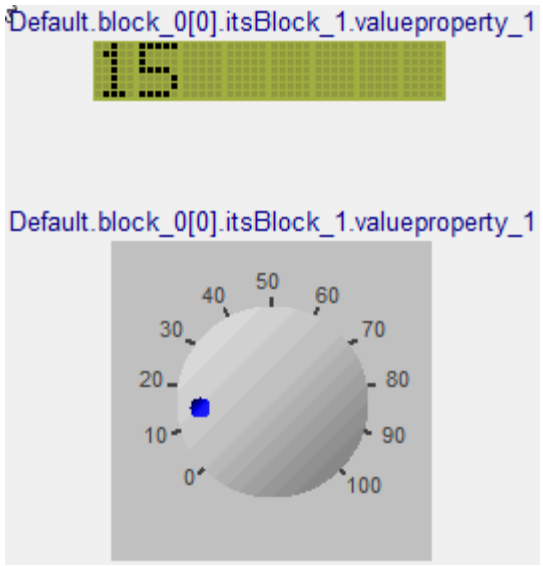
- General approach:
  1. Construct / acquire a systems model that addresses system structure and behavior
  2. Use the animation capabilities of your modeling tool to assess the system's conformance to requirements
- Capabilities for assessing conformance vary based on the modeling tool

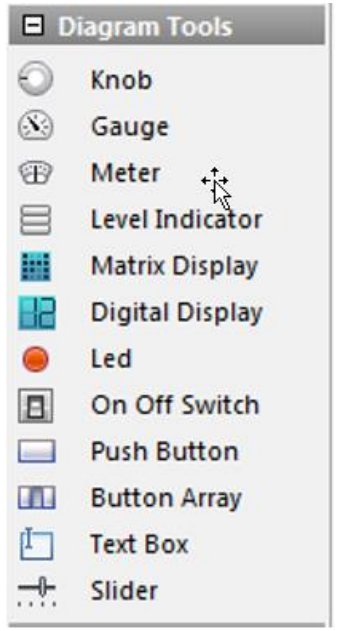


Panel Diagrams

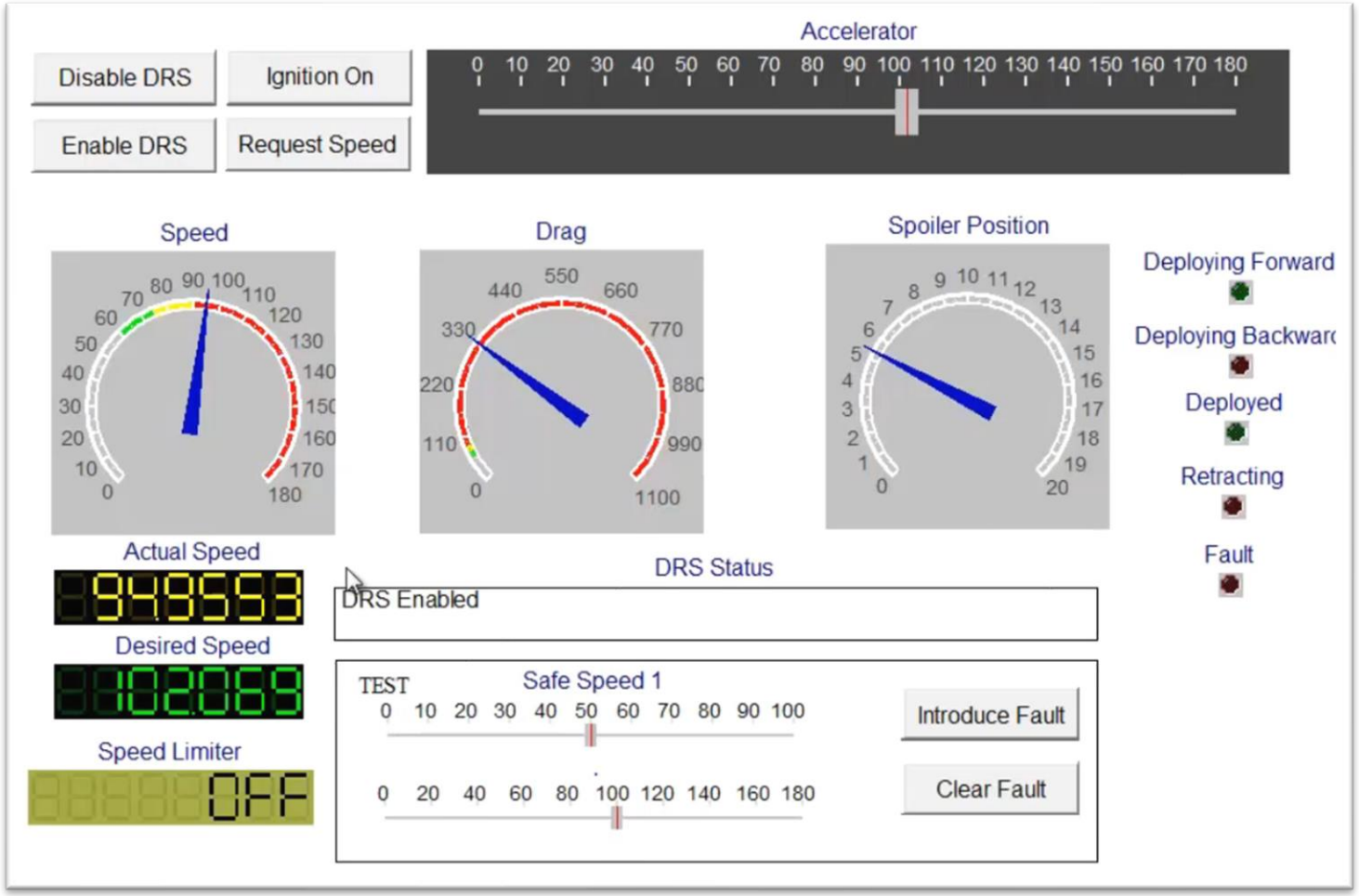
- **Panel Diagrams are an IBM Rhapsody diagram that provide a graphical control panel for your executing model**
  - **A pallet of built-in widgets is provided**
  - **Widgets are bound to model elements, to stimulate and read your model**







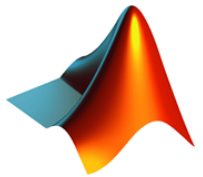
Panel Diagrams simplify the presentation and understanding of complex model behavior



Model-based verification of more complex systems

Use the right tool(s) for the right job

Example: co-simulation of combined reactive / continuous systems using Simulink and your favorite UML / SysML tool



Simulink is well suited to describe continuous transformational behavior

– Continuous algorithms for control systems

– Simulations of continuous physical processes

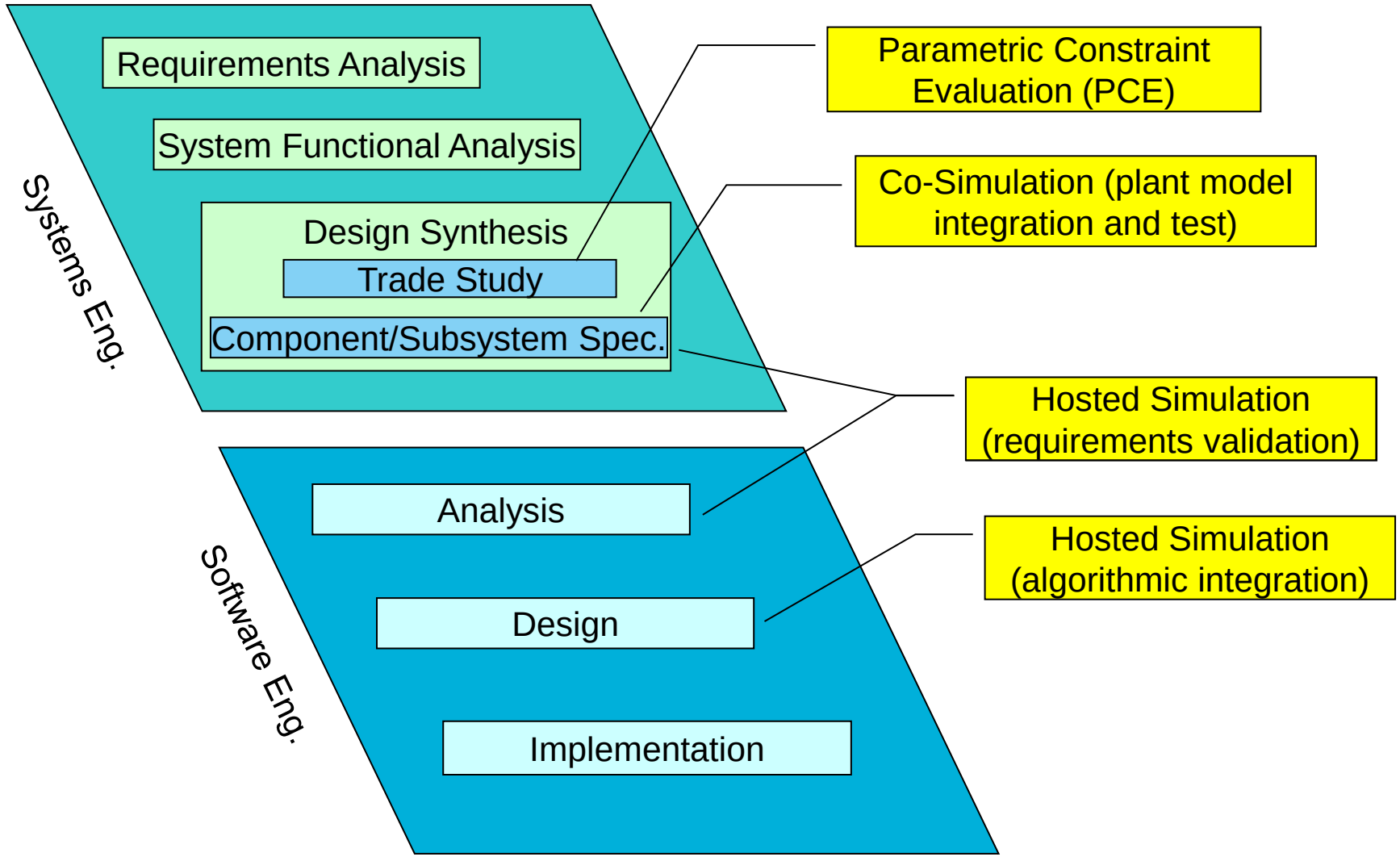
UML and SysML are well suited for

– Specifying dynamic behavior using activity diagrams and sequence diagrams

– Specifying complex reactive behaviors using state machines

– Specifying procedural behavior using activity diagrams

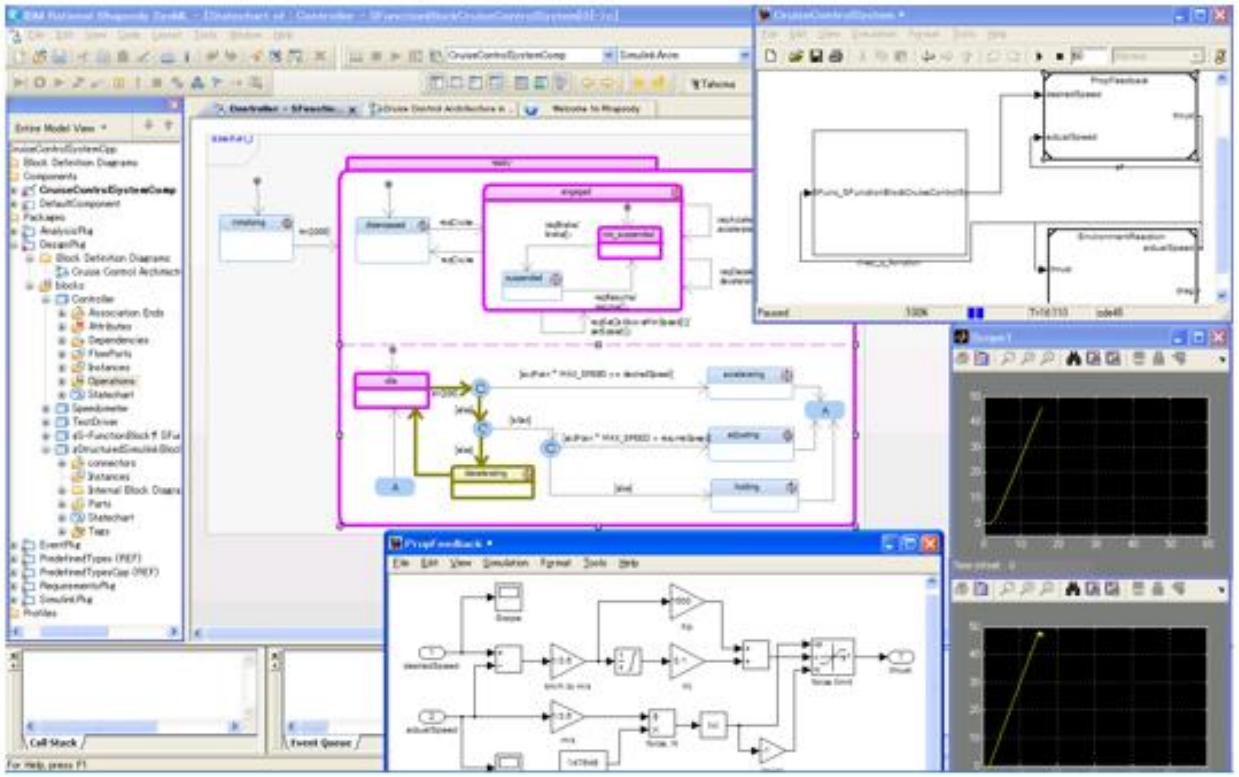
Integration points usage on the “V-Model”



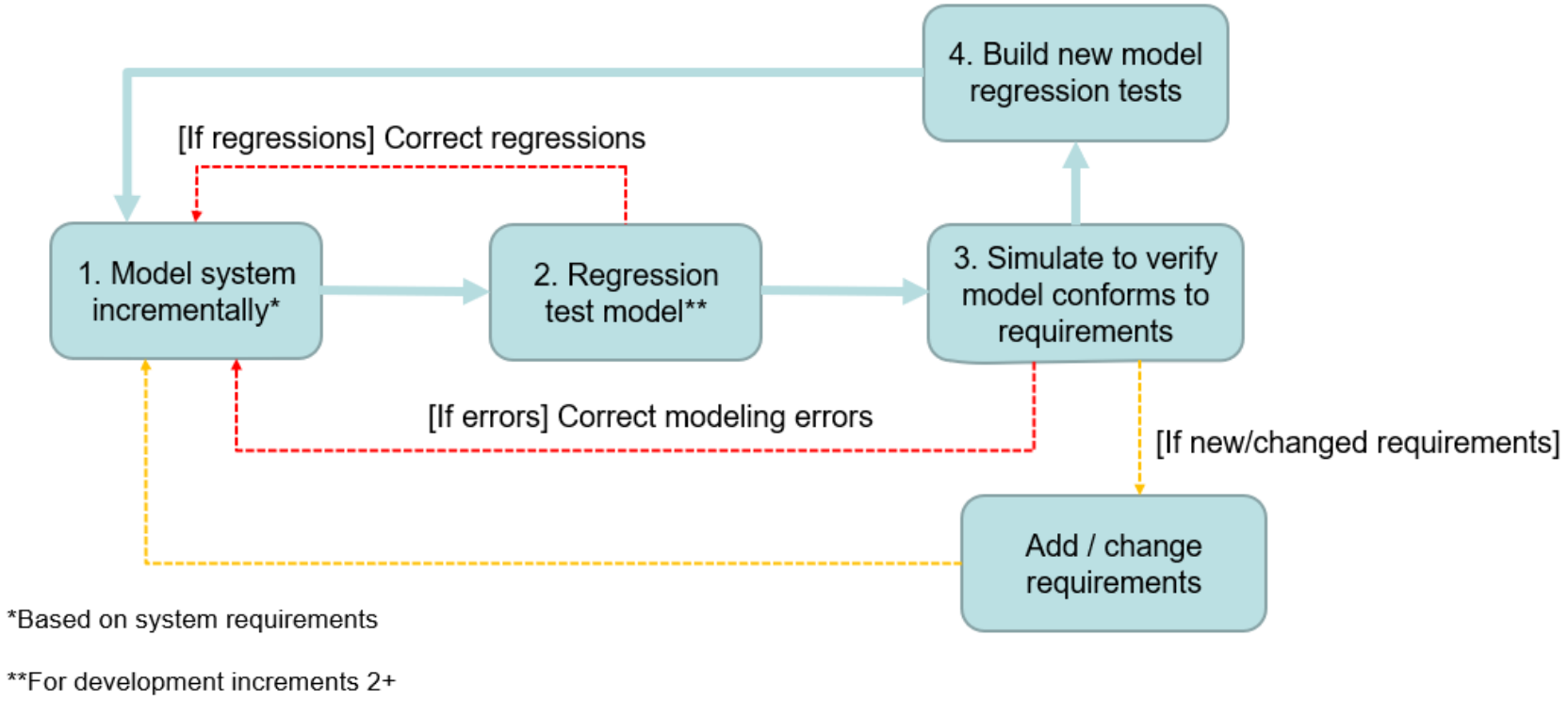
Co-simulation demonstration video

- This video does a good job of showing co-simulation using Rhapsody and Simulink
- For a quick look at co-simulation, see 12:00-13:40
- The RetractableSpoiler example at the end, starting at 13:40 is a good example of using a Panel Diagram with a hosted simulation

https://www.youtube.com/watch?v=MtoCq2u2_Ec



Remember this?



- **We will now talk about Steps 2 and 4**

Steps 2 and 4: Shifting Left with Model Based Testing

- Imagine if you could:
  - Build your system and software models using Agile Test-Driven Development (TDD) approaches
  - Generate test models and executable test cases for your system models, source code models, and source code, leveraging your system and software models as input
  - Automatically regression test your system and software models, to assess variances vs. approved modeled structure and behavior
- These things can be done using the **UML Testing Profile (UTP)** and appropriate tooling that leverages UTP to automate model-based testing activities

Overview of UML Testing Profile (UTP)

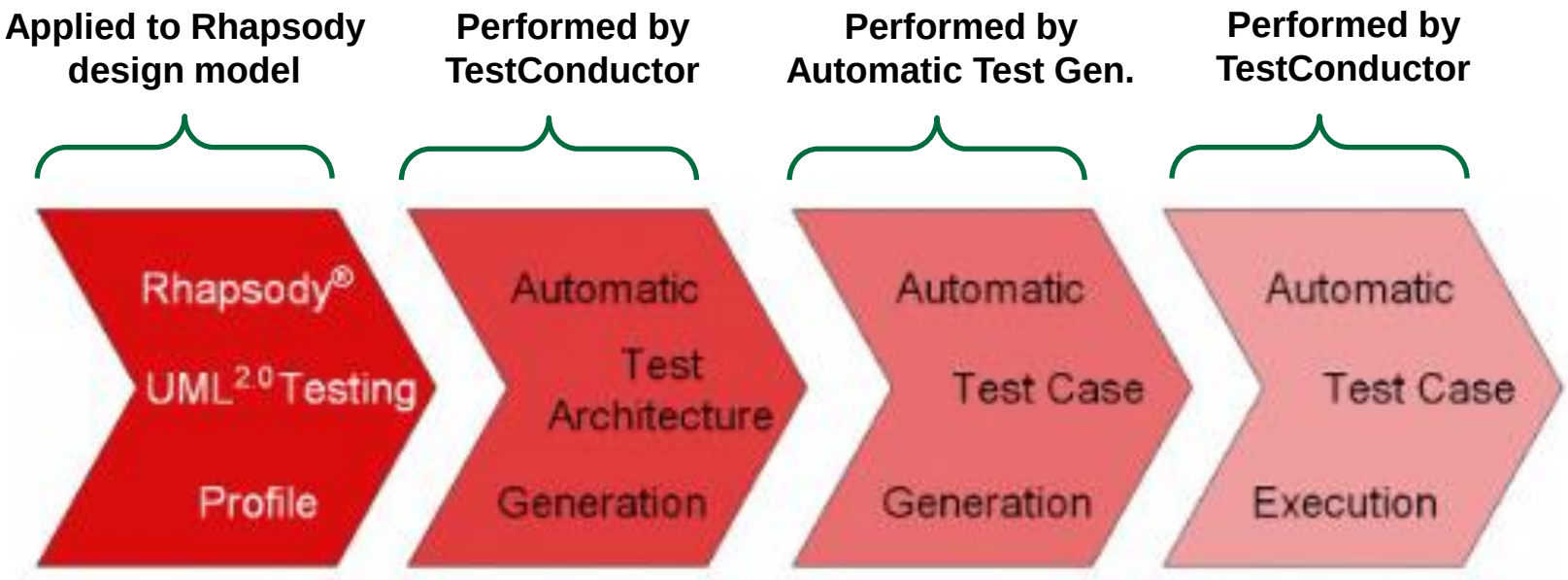
- Unified Modeling Language (UML) – general-purpose modeling language
- Includes extension mechanisms, such as profiles and stereotypes, for creating domain-specific modeling languages (DSLs)
  - For example, SysML 1.x is a UML profile
- UML Testing Profile (UTP) is a DSL for the testing domain
- The UTP DSL supports the following:
  - Test modeling
  - Test case specification
  - Test data specification
  - Test deployment
  - Test result visualization
  - Combination with other profiles, such as SysML and UAF
- For details, see the following:
  - UTP 1.2 specification: https://www.omg.org/spec/UTP/1.2/PDF
  - UTP 2.1 specification: https://www.omg.org/spec/UTP2/2.1/PDF
  - UTP Tutorial: http://www.model-based-testing.de/mbtuc11/presentations/Wendland_etal-UTP-Tutorial_1.pdf

Automated support for model-based testing

- **UTP by itself – as a manual DSL – has limited value**
- It gains value when used as a notation and meta-model to support automated test tooling
- In the rest of this presentation, we will describe model-based testing in the context of a concrete set of tools, those being
  - **IBM Rhapsody** for systems and software modeling
  - **Rhapsody TestConductor** for defining test models, model-based testing artifacts, and performing automated model-based tests
  - **Automatic Test Generation (ATG)** for automatically generating TestConductor test cases from Rhapsody system and software models

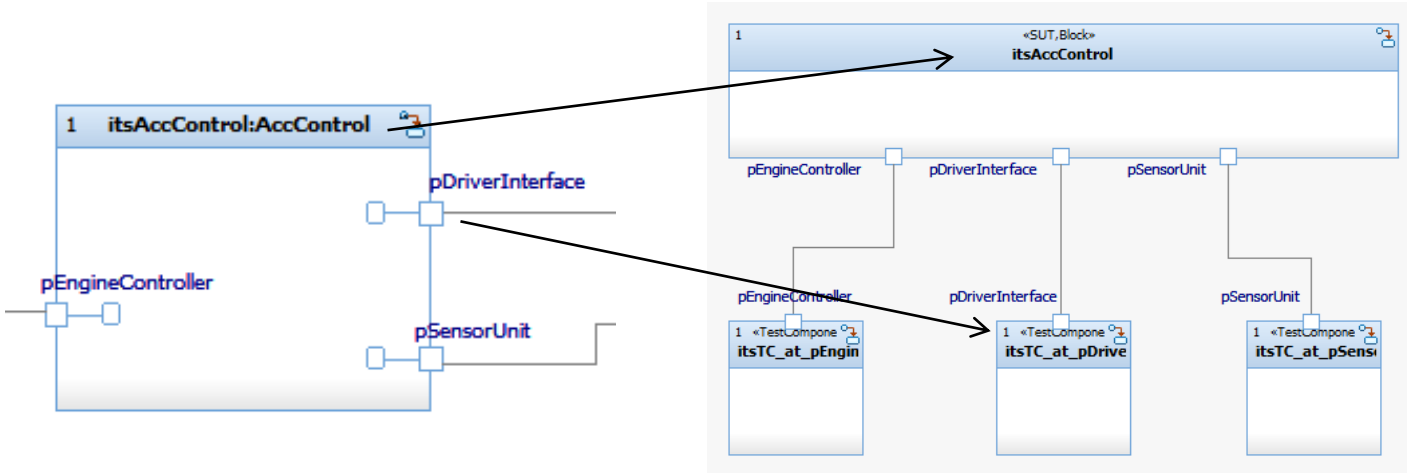
Model-based testing workflow

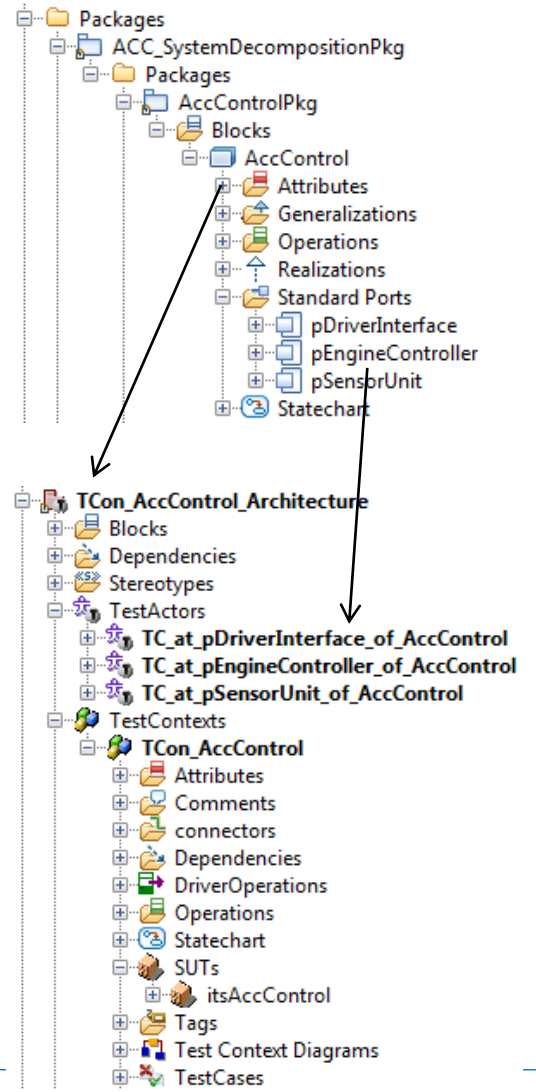
- The tools play these roles in the MBT workflow



TestConductor Test Architecture

- TestConductor automatically creates a test architecture for the system under test (SUT) based on the design architecture
- Test components represent counterparts of SUT interfaces (ports, associated classes)
  - **Separation of design and test artifacts**
  - **Maintaining traceability by dependencies and other UML artifacts**

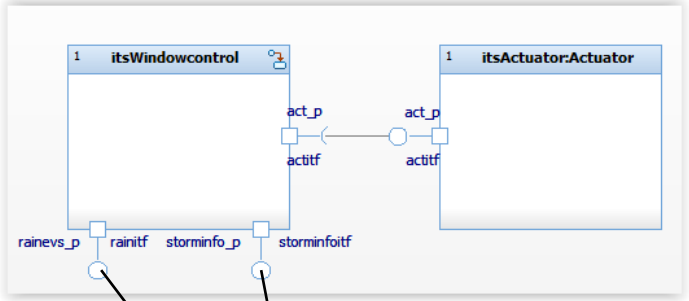




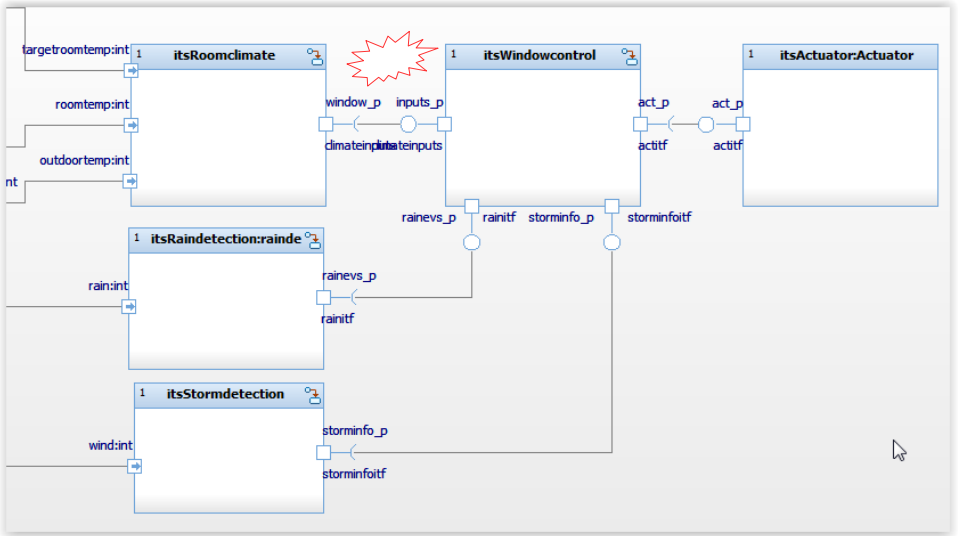
TestConductor Continuous Testing

- Test Components allow stubbing of not yet realized design elements
- Enable testing from start of development, for agile or test-driven process
- Continuous testing helps detect regressions early
- Update of test architecture to sync with design
  - Add test components if design architecture changed
  - Add interfaces to test components if SUT interfaces changed
  - Test architecture grows with design while test cases ensure quality

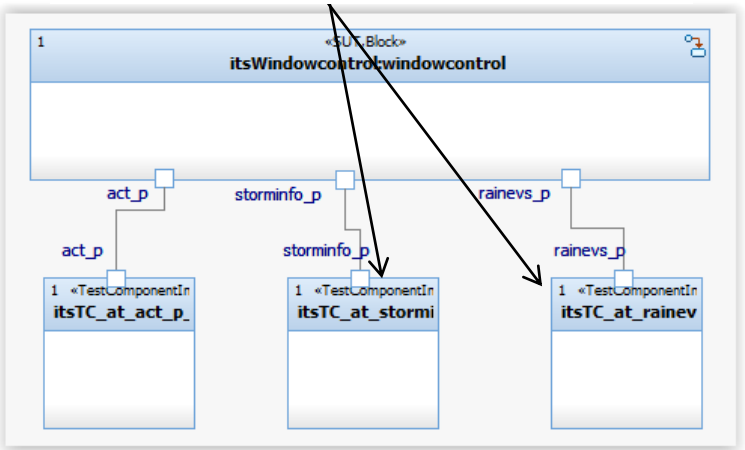
Initial design

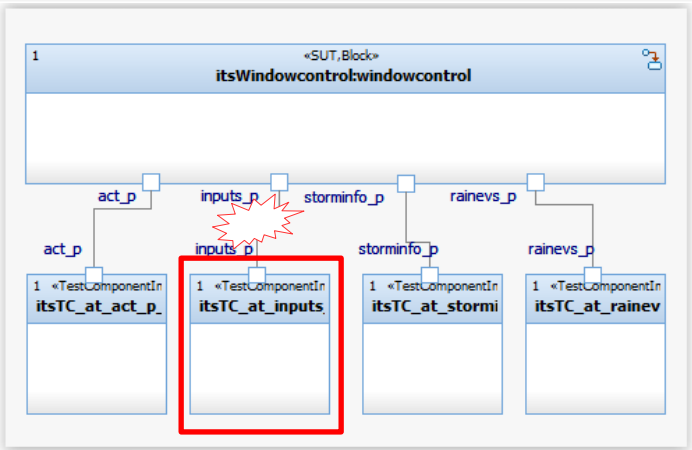


Easily update test architecture as design evolves



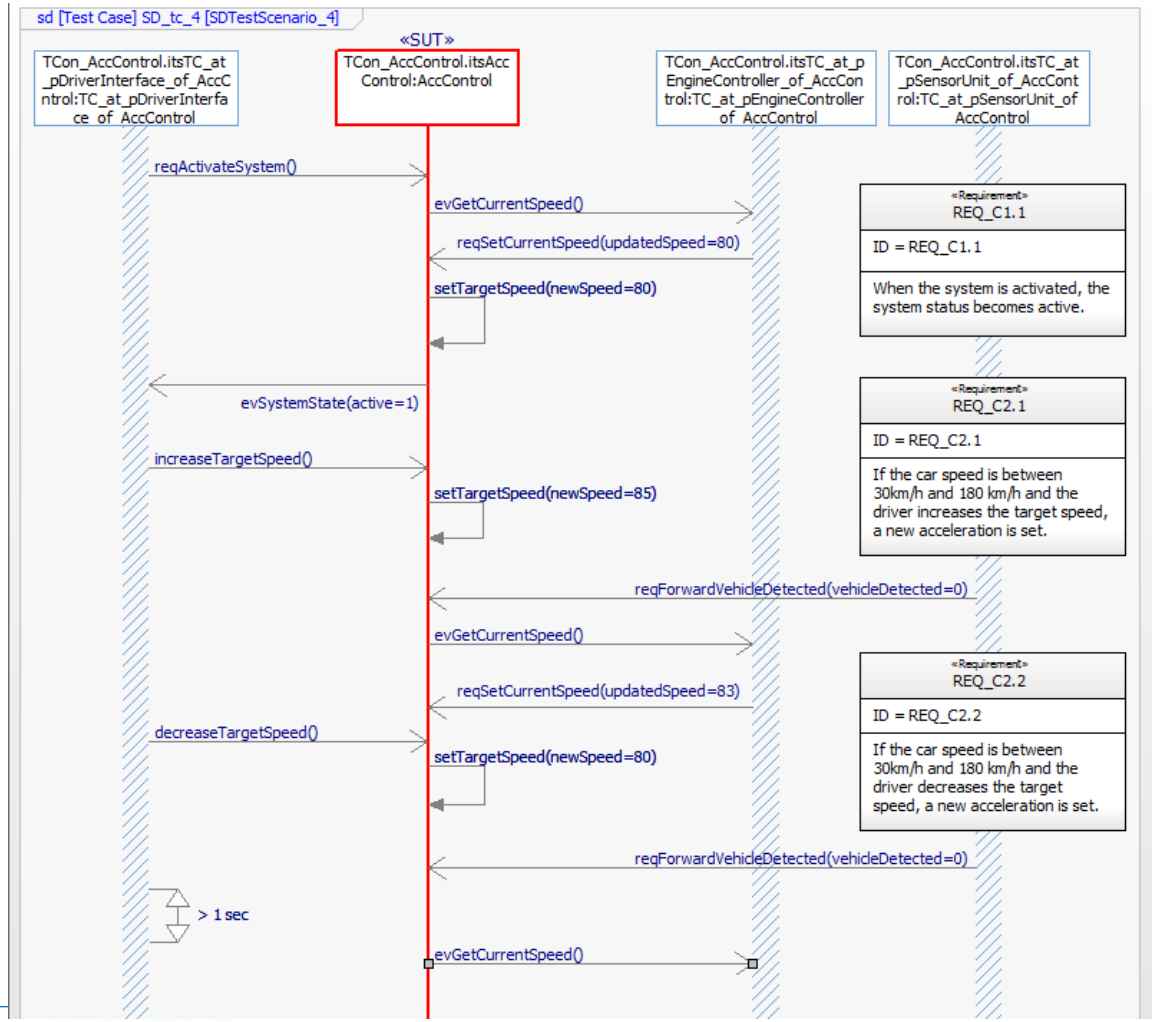
Generated test arch. w test components generated for designed ports





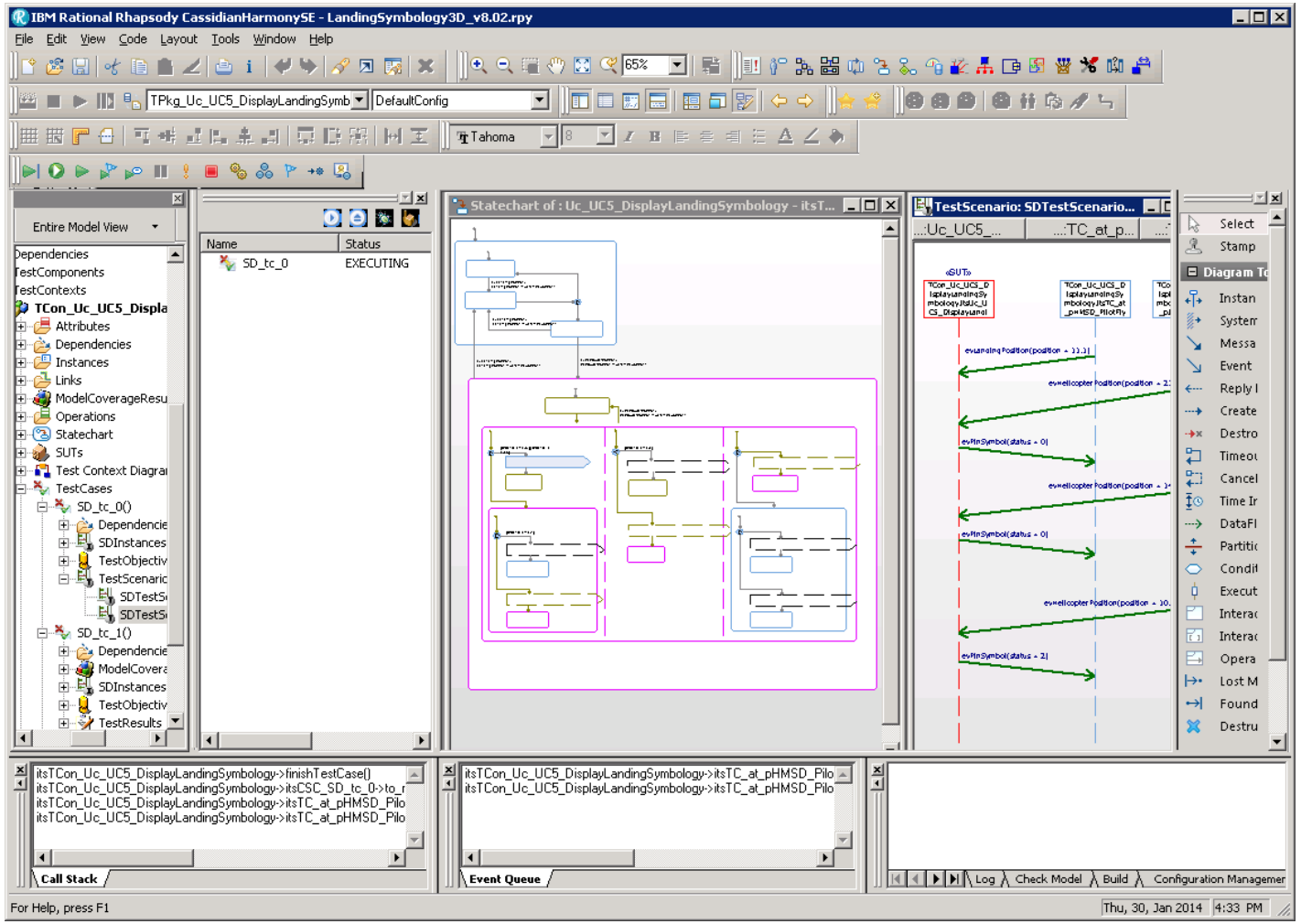
TestConductor Test Cases

- Test scenarios specify test cases
- Can be either
  - Manually created within the test architecture or
  - Generated from manually-created design-model sequence diagrams or
  - Generated from sequence diagrams auto-created during design model execution sessions
- TestConductor automatically generates driver and stub code in test components according to test specification
- Test components create test stimuli and observe the behavior of the system under test



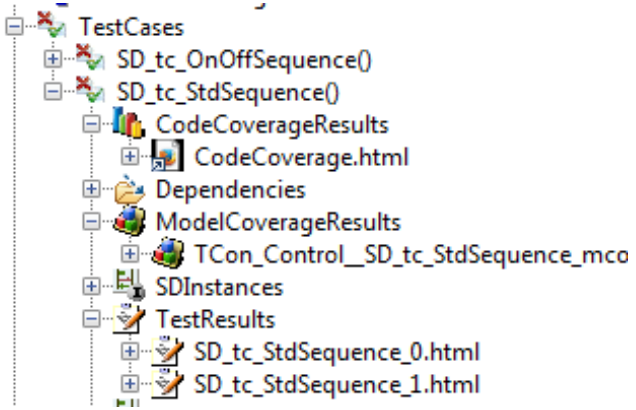
TestConductor Test Execution

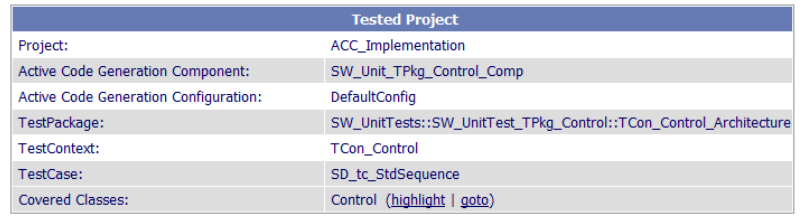
- During test execution, the SUT is stimulated, and responses are validated
- Tests can be executed on host, simulator or target
- Rhapsody capabilities (animated state charts, sequence diagrams) help in analyzing tests
- Witness scenario of test execution shows color coded test scenario



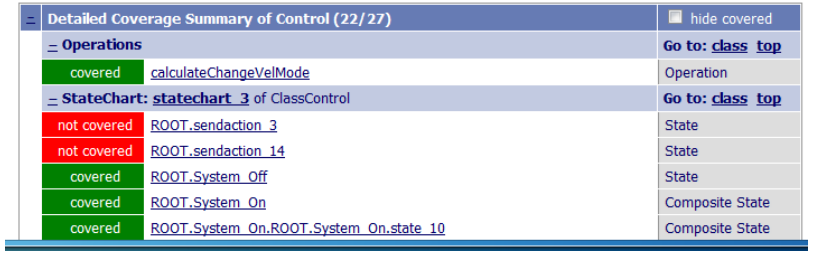
TestConductor Test Results and Coverage Reports

- After test execution, test result is computed and added to the model
- Optional reports for different coverage criteria
  - Model coverage
  - Requirement coverage
  - Code coverage

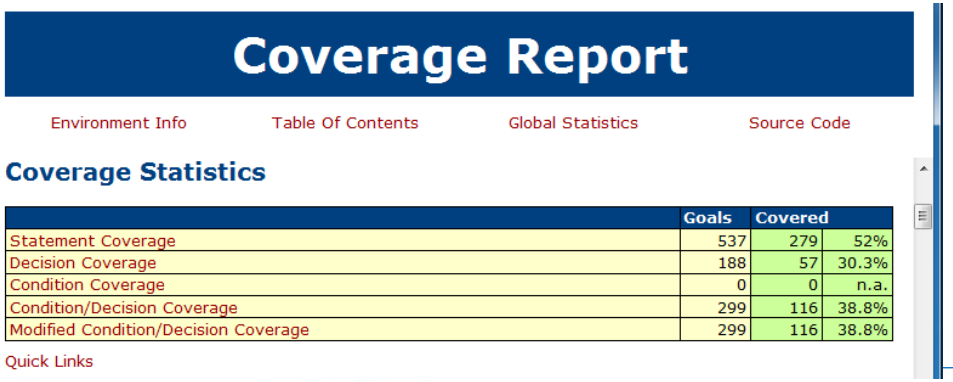




Tested Project	
Project:	ACC_Implementation
Active Code Generation Component:	SW_Unit_TPkg_Control_Comp
Active Code Generation Configuration:	DefaultConfig
TestPackage:	SW_UnitTests::SW_UnitTest_TPkg_Control::TCon_Control_Architecture
TestContext:	TCon_Control
TestCase:	SD_tc_StdSequence
Covered Classes:	Control (highlight | goto)



Detailed Coverage Summary of Control (22/27)		hide covered
– Operations		Go to: class top
covered	calculateChangeVelMode	Operation
– StateChart: statechart_3 of ClassControl		Go to: class top
not covered	ROOT.sendaction_3	State
not covered	ROOT.sendaction_14	State
covered	ROOT.System_Off	State
covered	ROOT.System_On	Composite State
covered	ROOT.System_On.ROOT.System_On.state_10	Composite State



Coverage Report

Environment Info | Table Of Contents | Global Statistics | Source Code

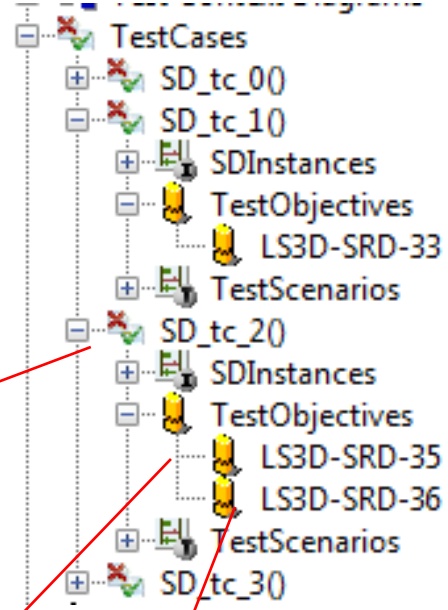
Coverage Statistics

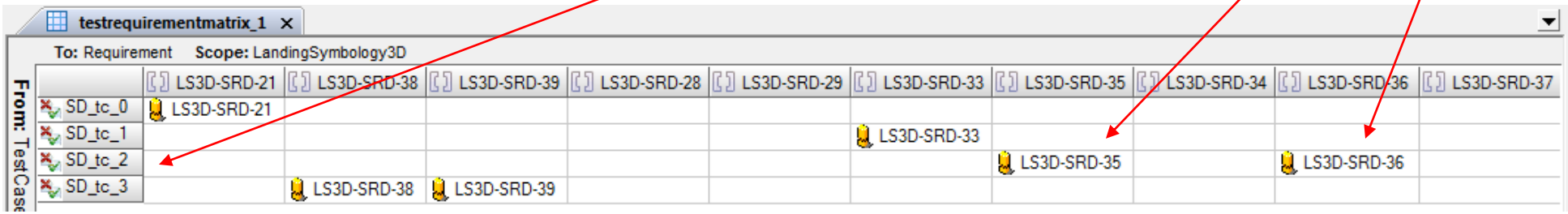
	Goals	Covered	
Statement Coverage	537	279	52%
Decision Coverage	188	57	30.3%
Condition Coverage	0	0	n.a.
Condition/Decision Coverage	299	116	38.8%
Modified Condition/Decision Coverage	299	116	38.8%

Quick Links

TestConductor traceability of tests, design, and requirements

- Traceability by modeling requirements, design, tests and the relations between them
- Allows automated reporting on tested/untested requirements and test results by standard Rhapsody functionality: Queries, tables, matrices etc.
- Allows report generation with IBM Engineering Publishing
- OSLC based traceability to work items, requirements, and other artifacts





The bottom line

- **We must shift left** – converge on our requirements and designs sooner, so that we can deliver innovative capabilities to our stakeholders sooner, with less cost, with higher quality
- Model Based Systems Engineering and Extensions provide proven approaches for shifting left
- Model Animation and System Simulation enable us to verify & validate our requirements and designs long before production software is written, and hardware components are developed
- Model Based Testing capabilities, such as those provided by IBM Rhapsody, enable us to detect regressions in approved models, build higher-quality models using TDD-based approaches, and test source code using test scenarios created directly from approved models

For further exploration

- See Model Animation and System Simulation with Rhapsody
  - Demo video illustrating model animation: Video 4.Architecture Management, at *https://ibm.ent.box.com/s/lym0y0xkt84ay59uwc5ymx1l7uhcg0vy*
  - Rhapsody integration with Simulink: *https://www.youtube.com/watch?v=MtoCq2u2_Ec*
- Learn more about Model Based Testing with Rhapsody and Rhapsody TestConductor
  - IBM Document Center content for Rhapsody TestConductor and Automated Test Generation (ATG): *https://www.ibm.com/docs/en/engineering-lifecycle-management-suite/design-rhapsody/10.0?topic=addons-model-based-testing-testconductor-automatic-test-generation-atg*
- Explore Model Based Testing with Rhapsody and Rhapsody TestConductor
  - Self-service, Cloud-based, hands-on labs can be found here: *https://learn.ibm.com/course/view.php?id=9173;*
  - *https://learn.ibm.com/course/view.php?id=9202*

Questions?