

IBM Symposium Systèmes 2014

Concevoir plus rapidement des systèmes
de plus en plus flexibles et complexes



Agile Model-Based Systems Engineering (aMBSE)

Bruce Powel Douglass, Ph.D.

Chief Evangelist, Global Technology Ambassador
IBM Rational

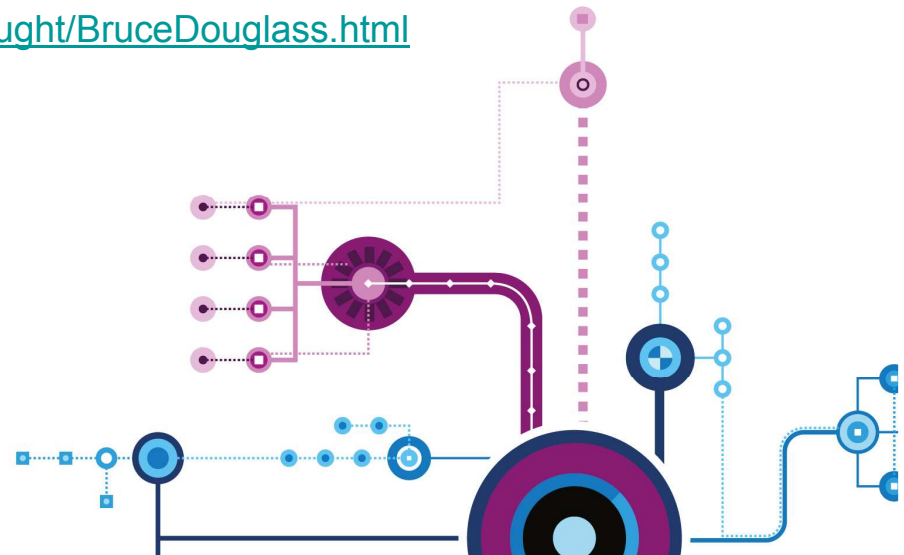
Bruce.Douglass@us.ibm.com

Twitter: @BruceDouglass

Yahoo: tech.groups.yahoo.com/group/RT-UML/

IBM: www-01.ibm.com/software/rational/leadership/thought/BruceDouglass.html

Jeudi 27 mars 2014
à l'IBM Client Center Paris



State of the Practice for Systems Development

- Systems Engineering Environments in general
 - Are document-centric
 - Require huge investment in planning that doesn't reflect actual project execution
 - Have difficulty adapting to change.
 - Require expensive and error-prone manual review and update processes.
 - Require long integration and validation cycles
 - Are difficult to maintain over the long haul
- Additional standards constraints (eg DO-178C, ARP4761, ISO26262, AUTOSAR, DoDAF) add to the challenge
 - Tooling Selection
 - Dependability engineering
 - Safety
 - Reliability
 - Security
 - System certification



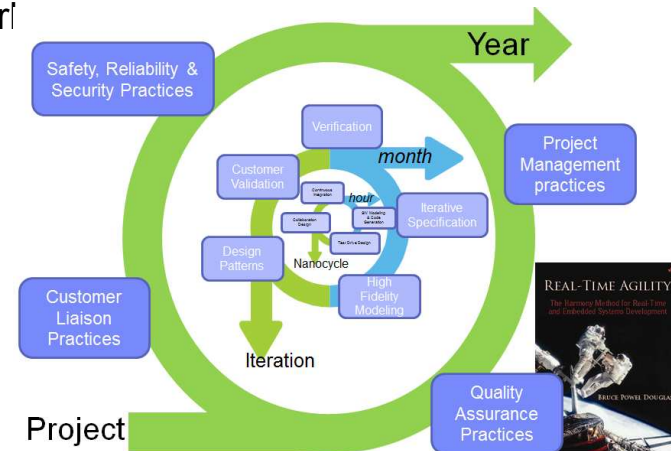
What do we mean by “verification”?

- **Syntactic verification** – “well-formedness” (*compliance in form*)
 - Performed by quality assurance personnel
 - Two types
 - Audits – work tasks are performed as per plan and guidelines
 - Syntactic review – work products conform to standard for organization, structure and format
 - Ex:
 - Requirements shall be uniquely numbered, be organized by use case, use the word “shall” to indicate the normative phrase of a requirement; functional requirements shall be modified by at least one quality of service requirement, ...
- **Semantic verification** – “correct” (*compliance in meaning*)
 - Performed by engineering personnel
 - Three basic techniques
 - Testing – requires Executability of work products, impossible to fully verify
 - Formal methods – strongest but hard to do and subject to invariant violation
 - Semantic review (subject matter expert & peer) – most common, weakest means



What does “agile” mean for Systems Engineering?

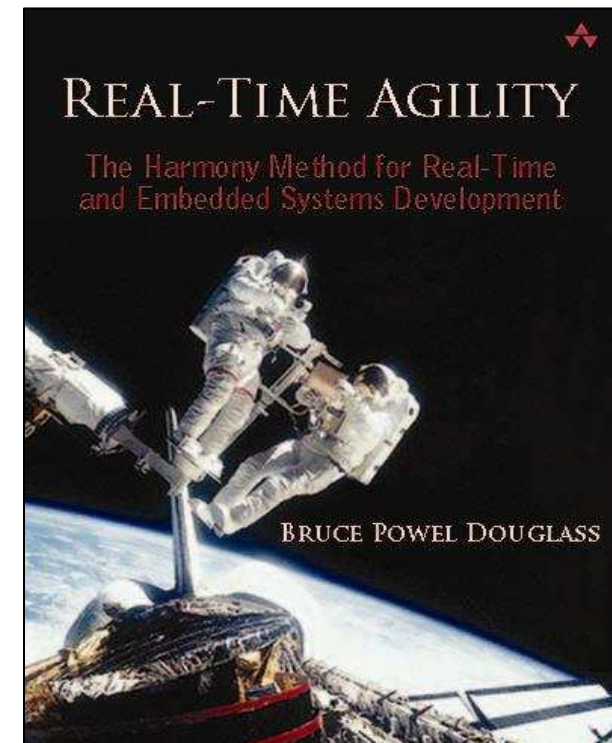
- Do what you need to do, no more and no less
 - This depends heavily on industry, regulation, and business environment
 - Provide the necessary level of rigor, precision, and repeatability
 - Often requires detailed traceability links among work products (e.g. requirements traceability)
 - Use tooling to automate manually-intensive, error-prone work
- Work iteratively and incrementally
 - Group requirements with user stories or use cases
 - Incrementally add traceability
 - Incrementally develop system architecture
- Verify work products continuously
 - With syntactic verification (Q/A) activities
 - With semantic verification
 - With customer (aka “validation”)
- Outcome contains textual specifications but also linked executable specifications
- Use dynamic planning to adjust project plans based on “ground truth” and responsiveness to change
 - Use goal-based metrics (KPIs) to track project progress
 - Continuously track progress against plan. Adjust planning frequently
- Safety, Reliability, Dependability
 - Not “done once” but continuously assessed



Best Practices for Agile Systems Engineering

- High-fidelity model-based engineering (Hi-MBE)
- Incremental functional analysis with use cases
- Test-driven development of system specifications
 - Example: Requirements verification via executable requirements modeling with SysML / UML
- Project risk management
- Incrementally add traceability
- Integrated safety and reliability analysis
- Model-based handoff to downstream engineering
- Automated document generation from model artifacts

Note: a key difference between agile SW and agile SE is that the *outcome* of SE is *specifications* and the *outcome* of SW is *implementation*



Model-Based Systems Engineering and Agile?

IBM Rational Harmony for Systems Engineering

Introduction to IBM® Rational® Harmony™ for Systems Engineering

Introduction to IBM® Rational® Harmony™ for Systems Engineering

Expand All Sections Collapse All Sections

Relationships

Contents

- The IBM® Rational® Harmony™ Library of Best Practices

Back to top

Main Description

| | | | | | |
|-----------------|-----------------|-------|---------------|-------------|------------------|
| | | | | | |
| Getting Started | Core Principles | Roles | Work Products | Disciplines | Delivery Process |

Welcome to IBM® Rational® Harmony™ for Systems Engineering

Harmony for Systems Engineering is a member of the The IBM® Rational® Harmony™ Library of Best Practices specifically for Systems Engineering development. The Harmony for Systems Engineering process focusses around integrated systems and software development. The process provides systems engineers with a step-by-step guide on using SysML in a way that allows a seamless transition to subsequent system development.

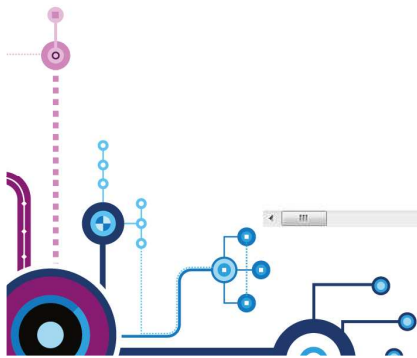
Harmony for Systems Engineering is model-based systems engineering process that leverages industry standard SysML language to develop executable requirements and architecture models in order to create verified and validated inputs to the system development and V&V activities.

Harmony for Systems Engineering may be used "out-of-the-box" or as the starting point for process tailoring and continuous improvement.

| | | | | | |
|-----------------|-----------------|-------|---------------|-------------|------------------|
| | | | | | |
| Getting Started | Core Principles | Roles | Work Products | Disciplines | Delivery Process |

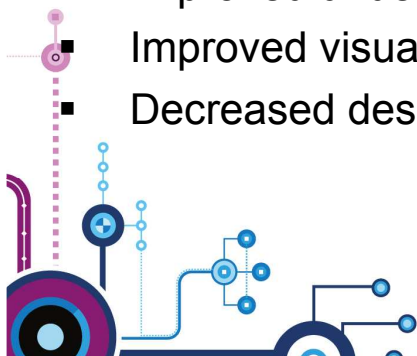
Back to top

IBM® Rational® Harmony™ for Systems Engineering Process Plug-in Copyright

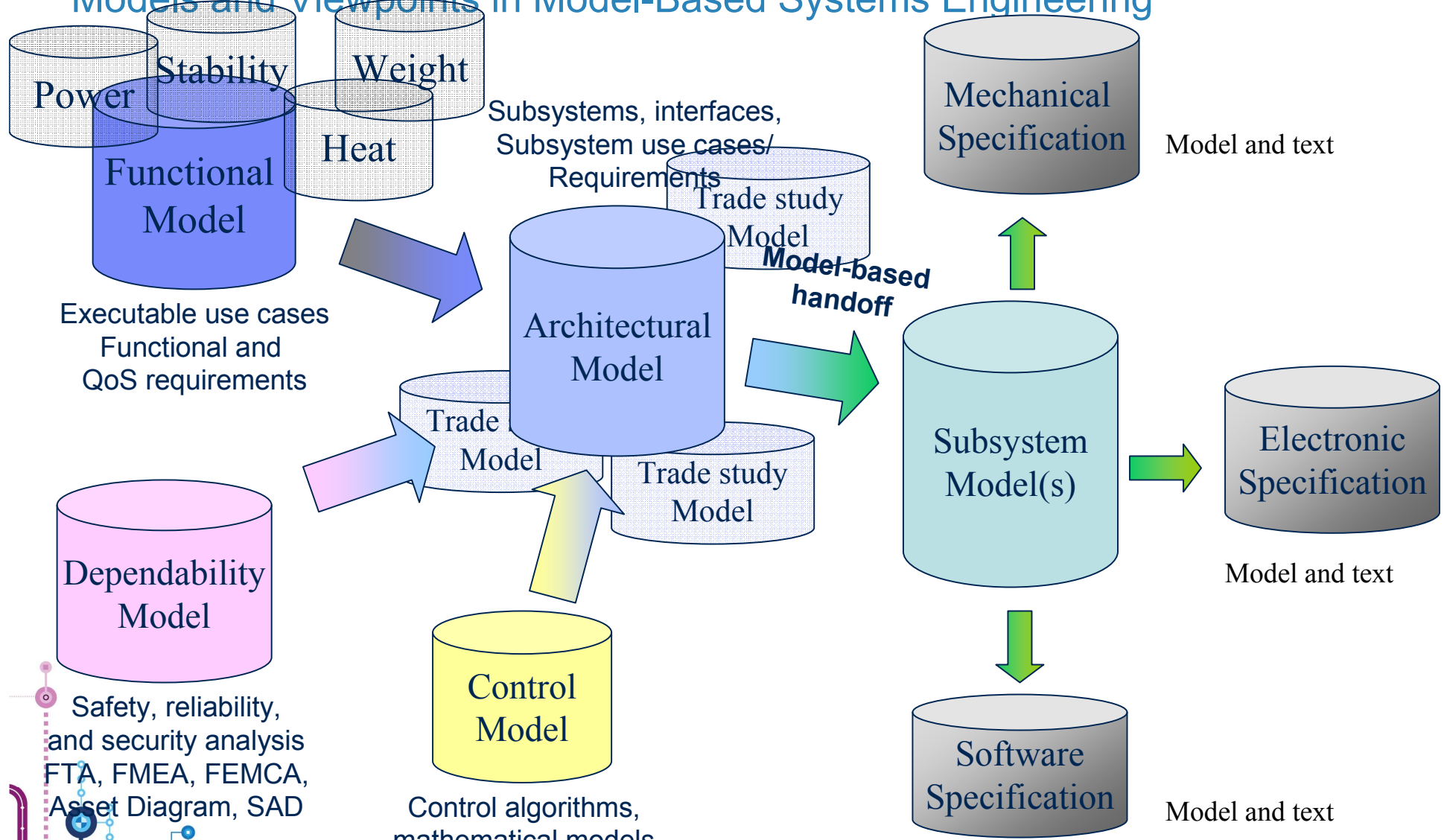


Advantages of MBSE

- Precision
 - Models constructed in formal (or semi-formal) languages are more precise than text
 - Recommendation: Link description informal text to precise, formal models
- Verification
 - Models can be executed, simulated, or (formally) analyzed
 - Requirements models
 - Architecture models
 - Dependability models
 - Control models
- Improved Handoff from systems engineering to downstream engineering
 - Precise models are less likely to be misinterpreted
 - If systems and software engineers use the same modeling languages, then no translation is required
- Improved understanding of architecture
- Improved visualization of functional, structural, and behavioral aspects
- Decreased design learning time



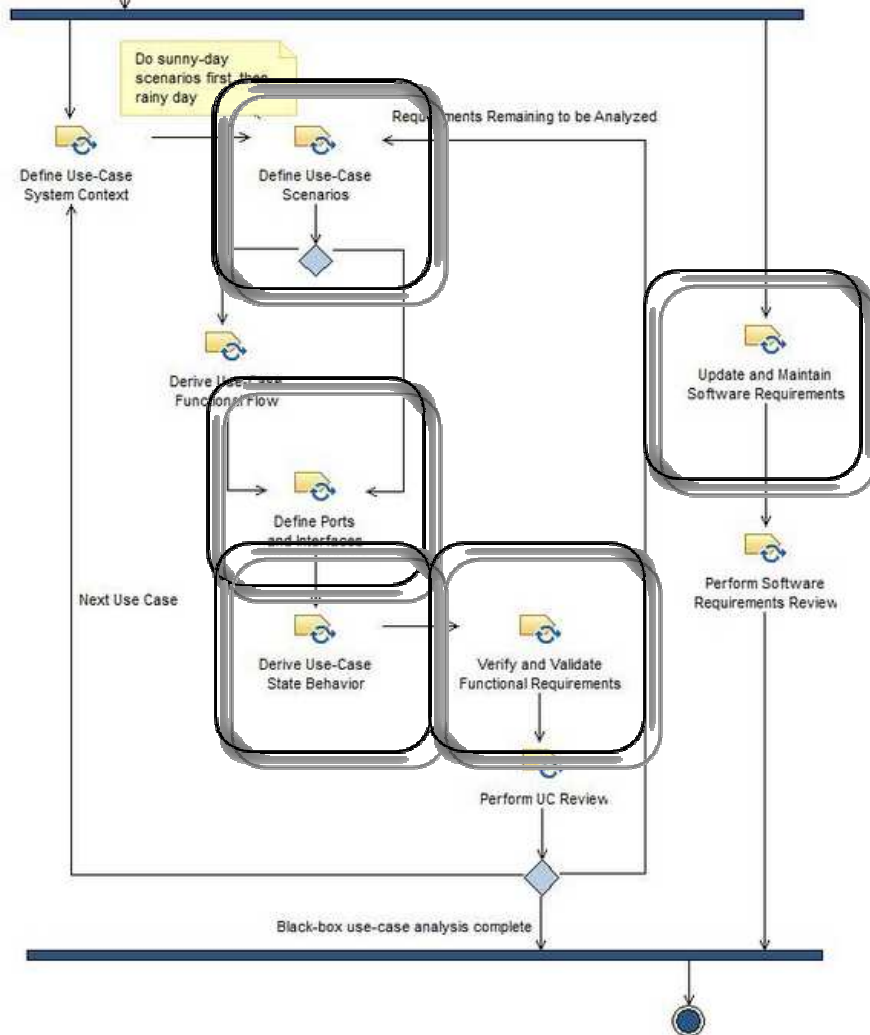
Models and Viewpoints in Model-Based Systems Engineering



#IBMSymposiumSystemes



Scenario Driven Use Case Construction / Validation



Making it Agile

Loop

Loop

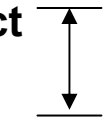
Conceptualize requirement aspect

Incrementally augment model

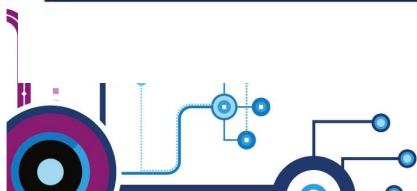
Verify

Repeat until all requirements added

Repeat for all use cases

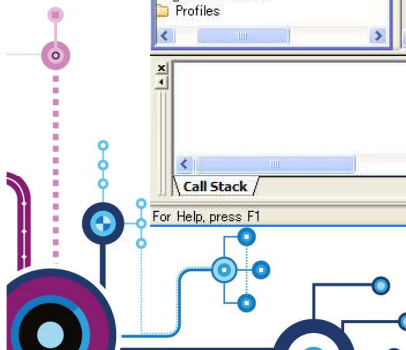


< 1 hr



Requirements Verification Using Rhapsody and Simulink

The image displays a multi-windowed software development environment. The main window is IBM Rational Rhapsody SysML, showing a statechart for a cruise control system. The statechart includes states like 'initializing', 'disengaged', 'engaged', 'idle', 'accelerating', 'adjusting', 'decelerating', and 'holding'. Transitions are labeled with events and conditions, such as 'reqCruise' and '[accPosn * MAX_SPEED >= desiredSpeed]'. A 'ready' state is highlighted with a purple border. To the right, a Simulink window shows a block diagram of the 'CruiseControlSystem' with inputs for 'desiredSpeed' and 'actualSpeed', and outputs for 'thrust'. Below the statechart, a 'PropFeedback' block diagram shows a control loop with gain blocks (Kp, Ki), integrators, and a force limit. Two 'Scope' windows show plots of 'actualSpeed' over time, with the top plot showing a ramp up to a steady state and the bottom plot showing a similar ramp.



Test-Driven Development isn't just for software anymore

- The principle behind TDD is to develop and apply test cases as you develop a system to demonstrate that it is correct
 - This is done in parallel with the system development and *not ex post facto*
 - This is about *defect avoidance* not so much *defect identification and repair*
- TDD applies to the development of complex system use case models
 - During the nanocycle of a use case's development
 - Make small incremental changes (e.g. add a state, or a couple of actions, or a transition or two)
 - Identify what is the desired behavior of the system that you've specified *so far*
 - Execute that incomplete use case model to ensure that it is correct
 - Repeat until all requirements for the use case and all scenarios defined for the use case have been met in the normative specification
- TDD may be realized in SE Models
 - By “instrumenting the actors” – specifying behavior of the actors to perform tests
 - Tooling implementing the UML Profile for Test (e.g. Test Conductor™ and Automatic Test Generator™)
 - Manually writing test scripts



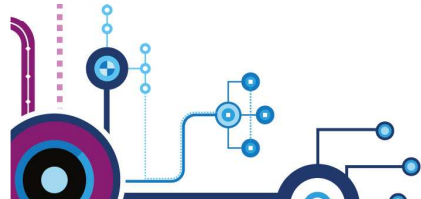
Traceability

- Traceability serves a number of purposes
 - It allows *impact analysis* – what is the impact if I change this element?
 - It allows for *coverage analysis* – are all elements realized?
 - It allows for *consistency analysis* – are these different elements in different work products consistent and compatible with each other?

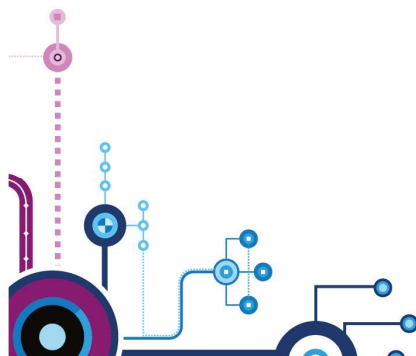
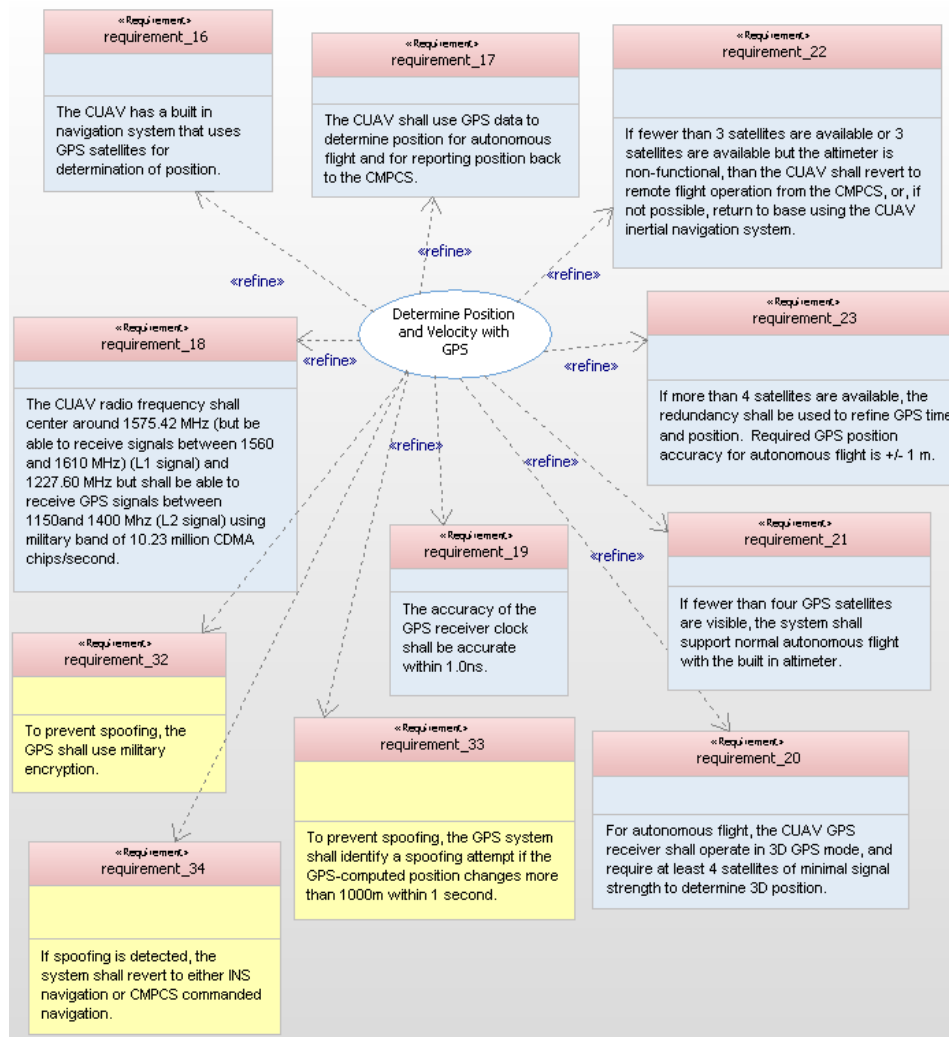
| | | Design / Implementation Elements | | | | |
|--------------|----|----------------------------------|----|----|----|----|
| | | D1 | D2 | D3 | D4 | D5 |
| Requirements | R1 | x | | | | x |
| | R2 | | | | | |
| | R3 | | x | | | |
| | R4 | | | | x | |

← Unimplemented requirement

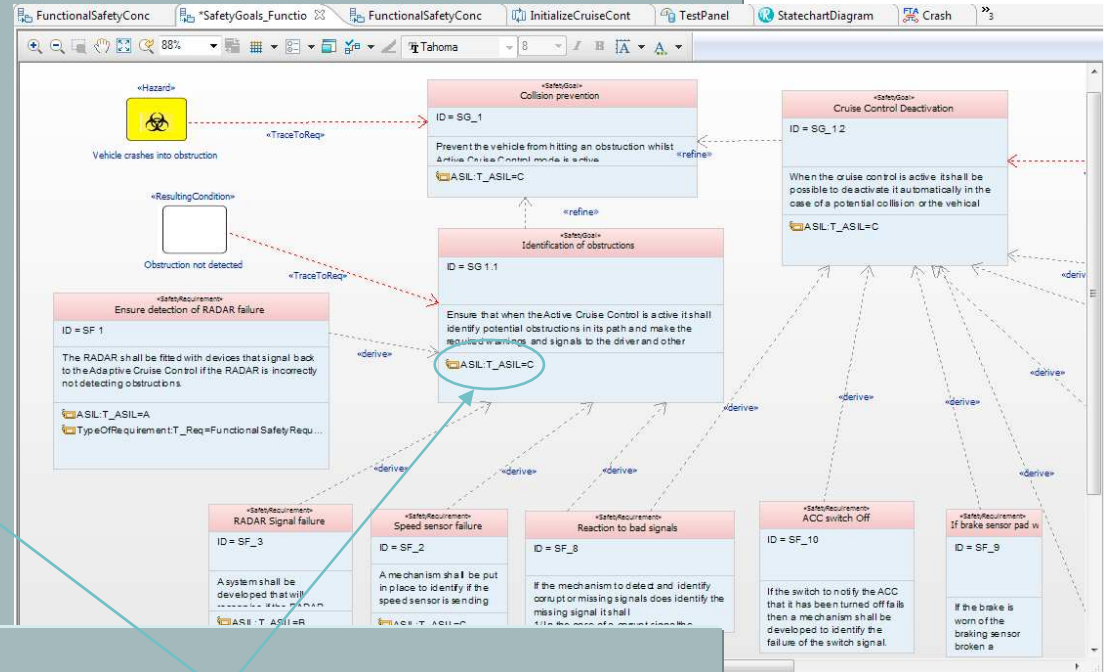
↑ Gold plating?



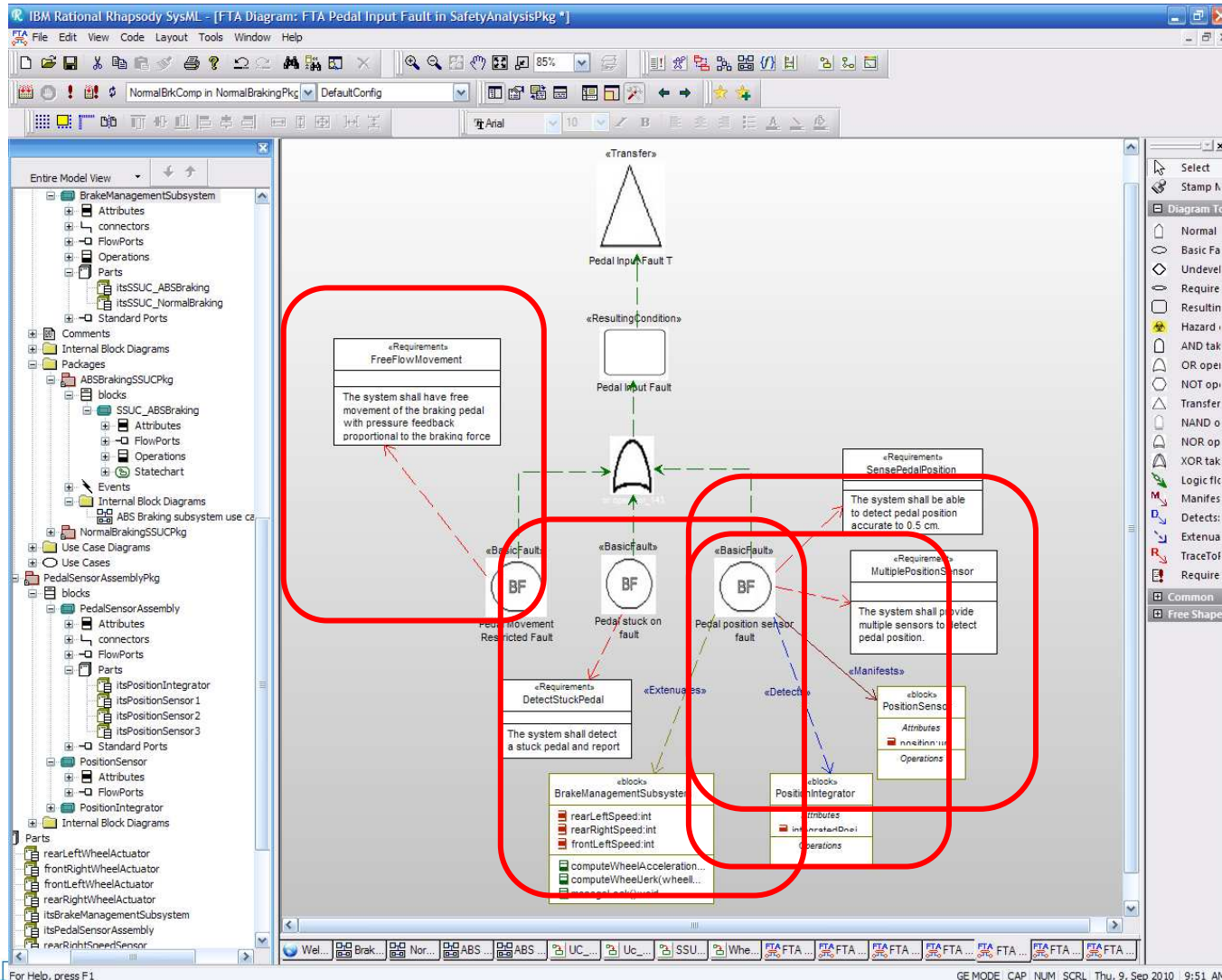
Traceability in Models



Important to Relate Safety Information Through Lifecycle

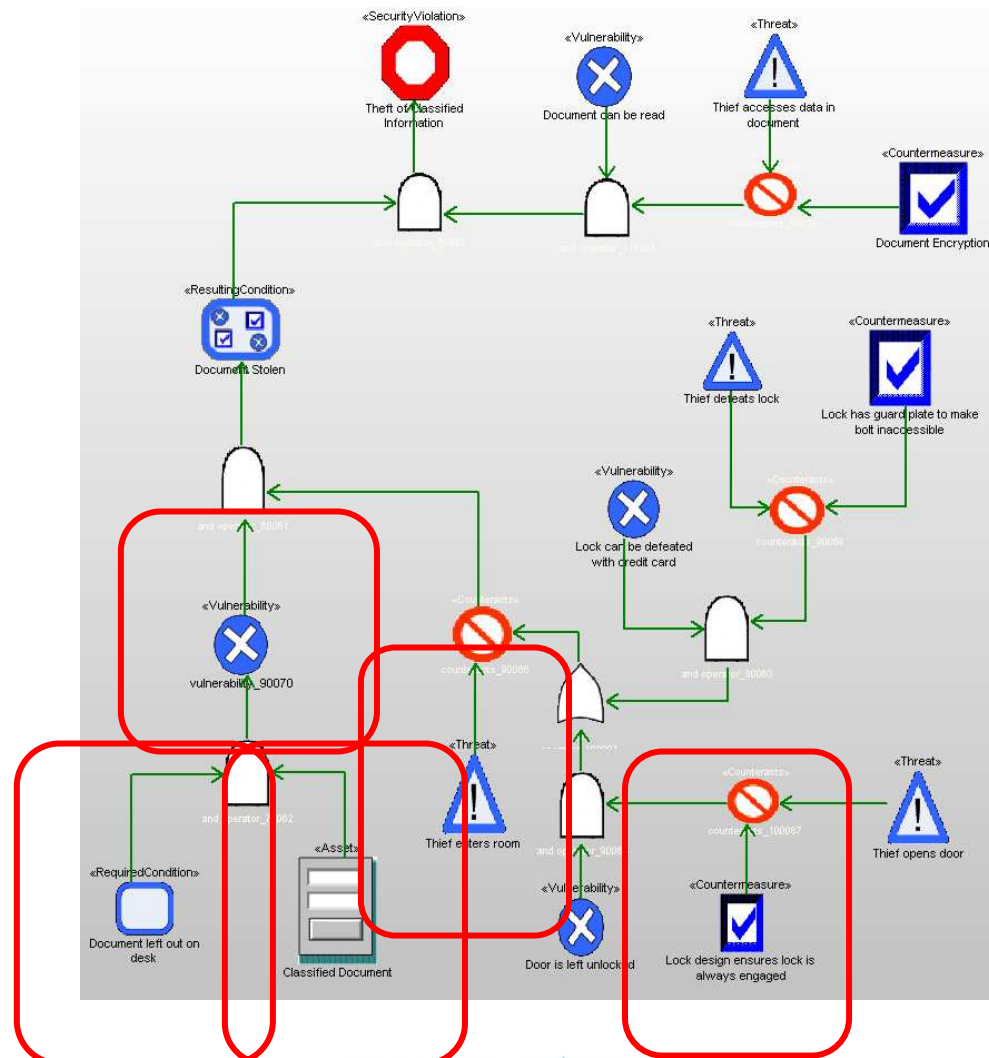


Integrated Dependability Analysis: UML Fault Tree Analysis Profile



Integrated Dependability Analysis: UML Security Analysis Profile

- Security Analysis Diagram (SAD) is like a Fault Tree Analysis (FTA) but for security, rather than safety
 - It looks for the logical relation between assets, vulnerabilities, attacks, and security violations
 - Permits reasoning about security
 - What kind?
 - How much?
 - Risk assessments



Auto-generation of documents (summary data)

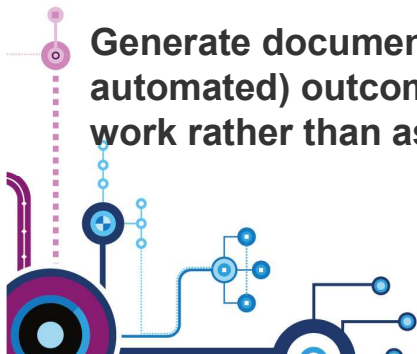
Fault Source Matrix, Fault Detection Matrix, Fault-Requirement Matrix, FMEA, Hazard Analysis...

The screenshot displays a software interface for system analysis. On the left, there are two hierarchical tree views. The top one is titled 'From: Basic Fault, Normal Event, Required Condition, Undeveloped Fa...' and lists various fault types like 'Gas Supply Fault', 'Ventilator Pump Fault', etc. The bottom one is titled 'From: Basic Fault, Hazard, Resulting Condition, Transfer Operator, Undeveloped Fa...' and lists more specific conditions like 'Breathing Circuit Leak', 'Ventilator Parameter Setting wrong', etc. The main area shows a table with columns for 'To: Class, Safety/Measure' and 'Scope: DesignModel'. Below this, there's a table for 'To: Requirement' and 'Scope: RequirementsAnalysis' with columns for various requirement IDs (REQ_BCM_09, REQ_BCM_11, REQ_VD_03, etc.).

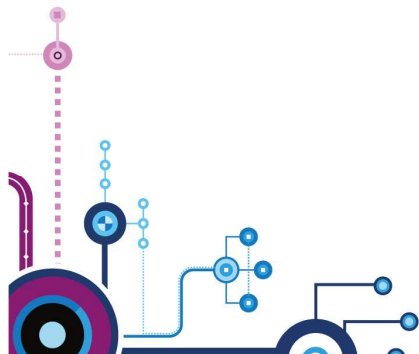
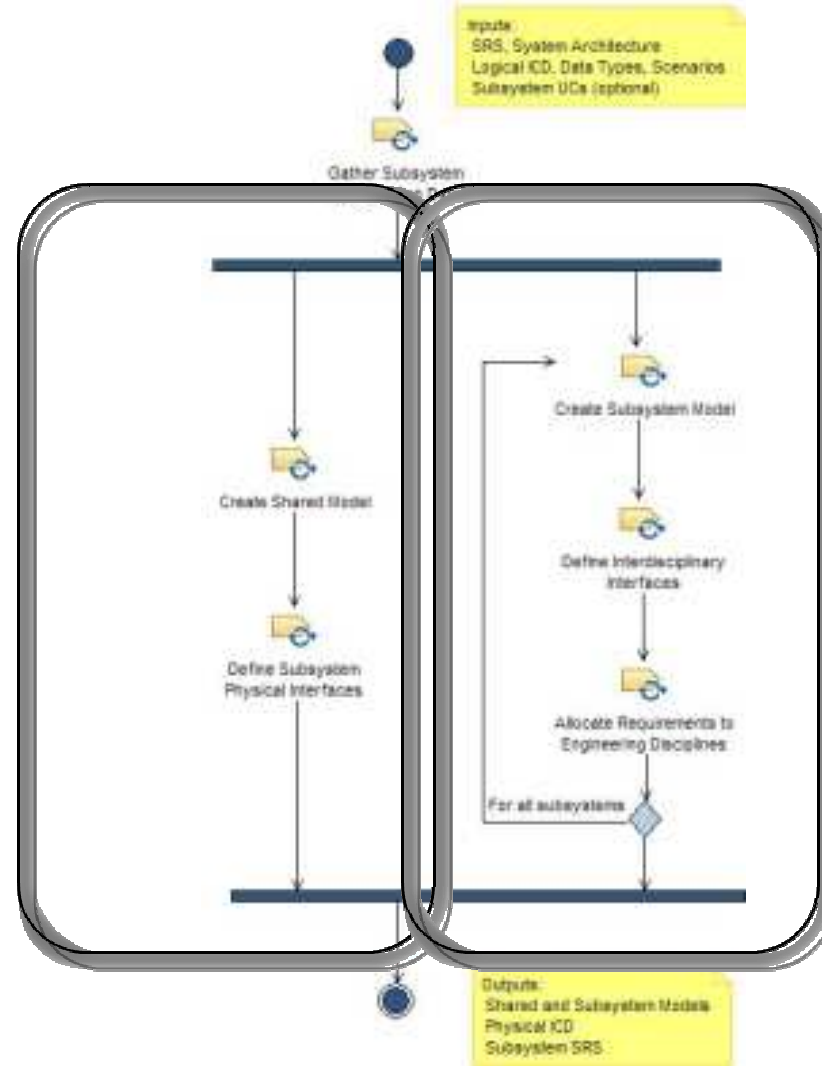
Traceability improves your ability to make your safety/security case

Generate documents are a natural (and automated) outcome of engineering work rather than as a separate activity

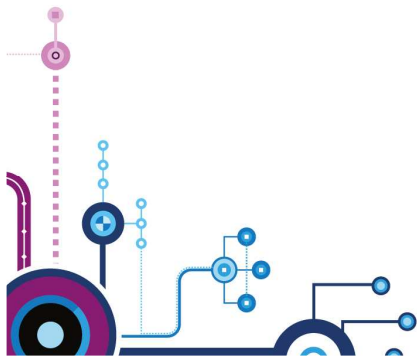
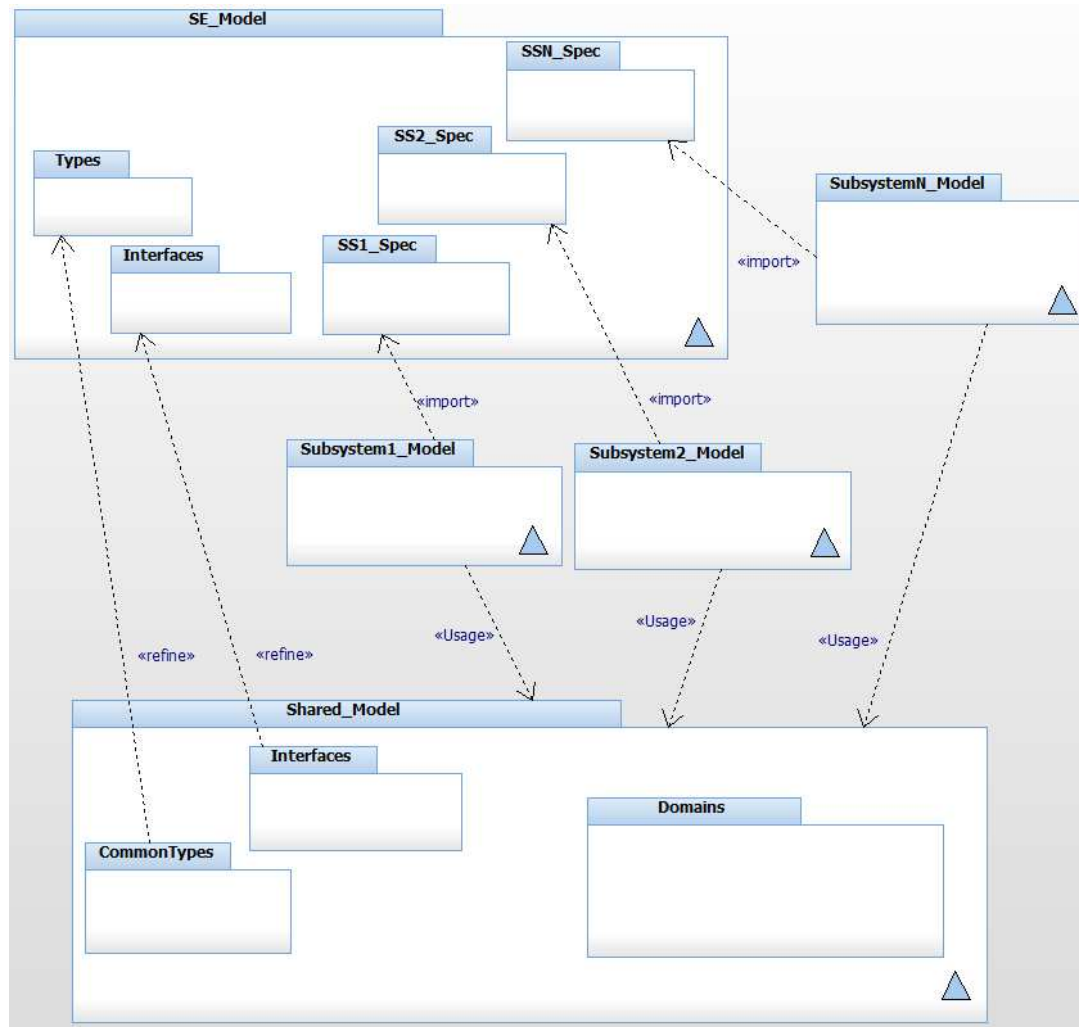
| Hazard | Description | Fault tolerance time | Fault tolerance time units | Probability | Severity | Risk | Safety integrity level |
|-------------------------|---|----------------------|----------------------------|-------------|----------|----------|------------------------|
| Hypoxia | The hypoxia hazard occurs when the brain and other organs receive insufficient oxygen. In a normal 21% O ₂ environment, death or irreversible injury occurs after five minutes of no oxygen. If the patient is breathing 100% for a significant period of time, this time is about 10 minutes. | 5 | minutes | 1.00E-02 | 8 | 8.00E-02 | 3 |
| Overpressure | Overpressure can damage the lungs. This is an especially severe trauma, possibly fatal, to neonates. | 200 | millisecond | 1.00E+04 | 4 | 3.00E+04 | 3 |
| Hyperoxia | Hyperoxia problems are usually limited to neonates, where it can cause blindness. | 10 | minutes | 1.00E+05 | 4 | 4.00E+05 | 4 |
| Inadequate anesthesia | Inadequate anesthesia leads to patient discomfort and memory retention of the surgical procedures. This is normally not life threatening but can be severely discomforting. | 5 | minutes | 1.00E+04 | 2 | 2.00E+04 | 2 |
| Over anesthesia | Over anesthesia can lead to death. | 3 | minutes | 1.00E+03 | 4 | 4.00E+03 | 4 |
| Anesthesia leak into ER | Anesthesia leak can lead to short or, in smaller doses, to long-term poisoning of medical staff. | 10 | minutes | 1.00E+05 | 5 | 4.00E+05 | 5 |



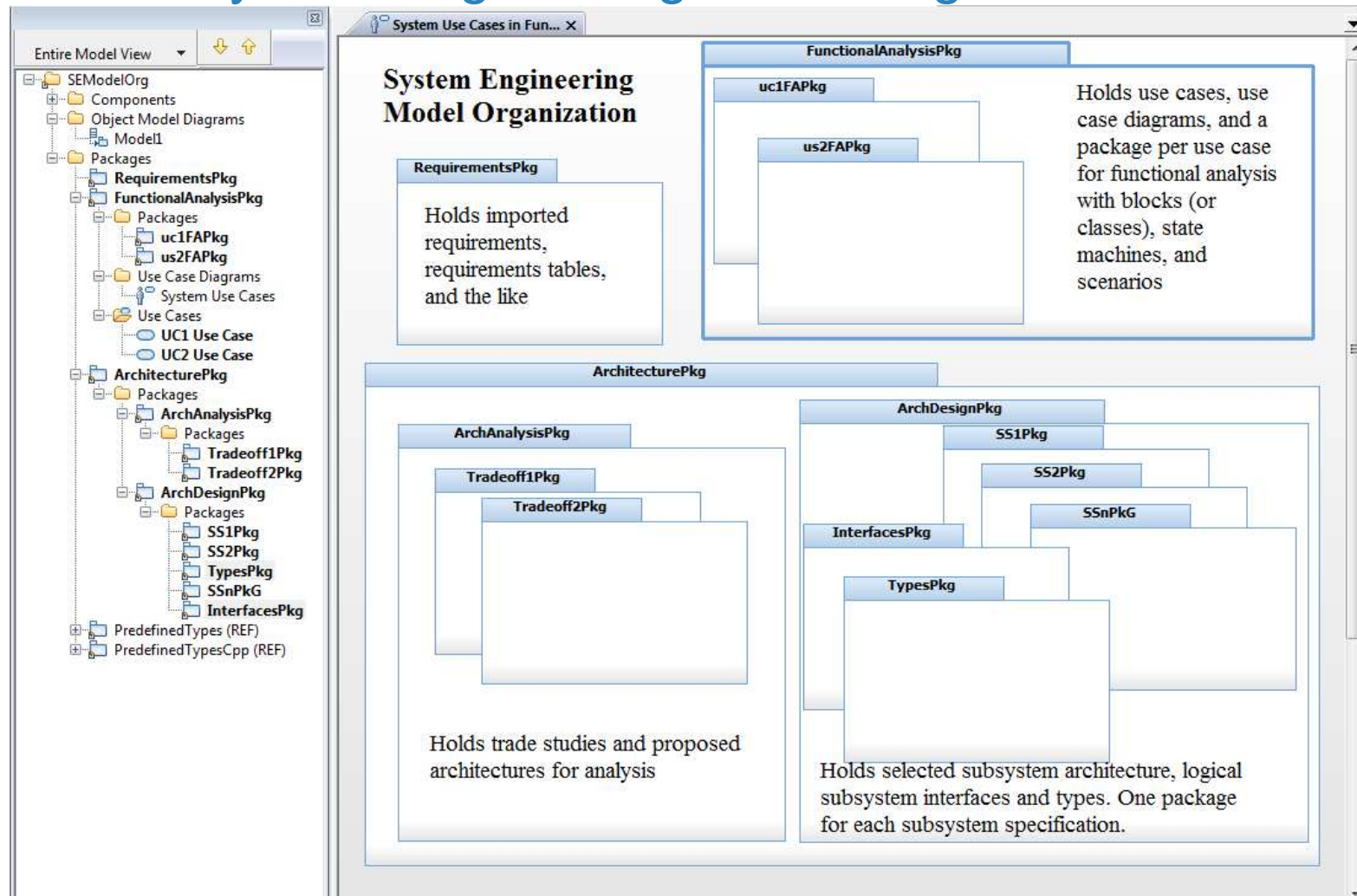
Model-Based Hand-off to Downstream Engineering



Canonical Model Organization



Canonical System Engineering Model Organization



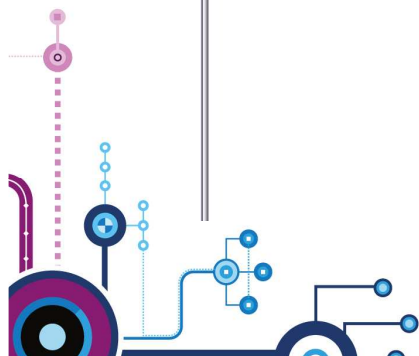
System Engineering Model Organization

RequirementsPkg
Holds imported requirements, requirements tables, and the like

FunctionalAnalysisPkg
uc1FAPkg
us2FAPkg
Holds use cases, use case diagrams, and a package per use case for functional analysis with blocks (or classes), state machines, and scenarios

ArchitecturePkg
ArchAnalysisPkg
Tradeoff1Pkg
Tradeoff2Pkg
Holds trade studies and proposed architectures for analysis

ArchDesignPkg
SS1Pkg
SS2Pkg
SSnPkg
InterfacesPkg
TypesPkg
Holds selected subsystem architecture, logical subsystem interfaces and types. One package for each subsystem specification.



Summary

- Systems Engineering capability can be greatly enhanced with two key technologies
 - MBSE - Use of SysML/UML Modeling to capture system
 - Functionality and Qualities of service (executable use cases)
 - Structure (architecture)
 - Model-based hand off to downstream engineering
 - Automatic generation of documentation from model-based work products
 - Agile methods employing
 - Incremental construction and verification of models
 - Test Driven Development nanocycle-level iteration
 - Incorporating dependability analysis with the SE workflow
 - Incremental traceability
- Harmony best practice workflows can be employed in an agile way
 - Process guidance – linked guidance to performance of tasks and creation of work products
 - Project Planning – create project plans with Harmony process templates in Rational Team Concert
 - Project Governance – monitor KPIs in project dashboards

