

Agile Systems Engineering

Bruce Powel Douglass, Ph.D. Unleash the Labs
Chief Evangelist
IBM Rational
Bruce.Douglass@us.ibm.com

Graham Bleakley Ph.D, Unleash the Labs
IBM Rational
graham.bleakley@uk.ibm.com

IBM Software

Innovate2012

The Premier Event for Software and Systems Innovation

Next  NOW!

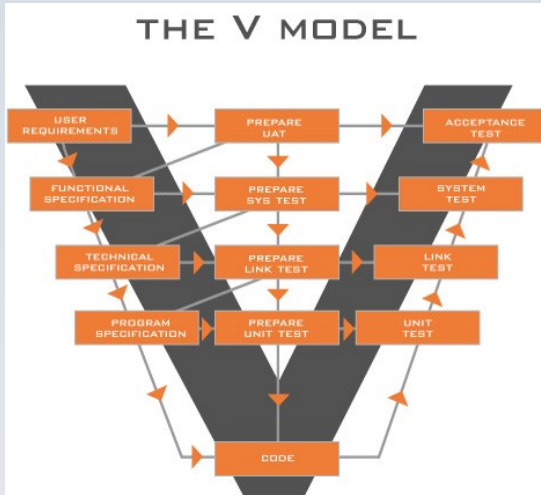
State of the Practice for Systems Development

- Systems and Software 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 to beat out many defects
 - Are difficult to maintain over the long haul
- Additional standards constraints (eg DO-178C, ARP4761, ISO26262, IEC 62304, AUTOSAR, DoDAF) add to the challenge
 - Tooling Selection
 - Dependability engineering
 - Safety
 - Reliability
 - Security
 - System certification

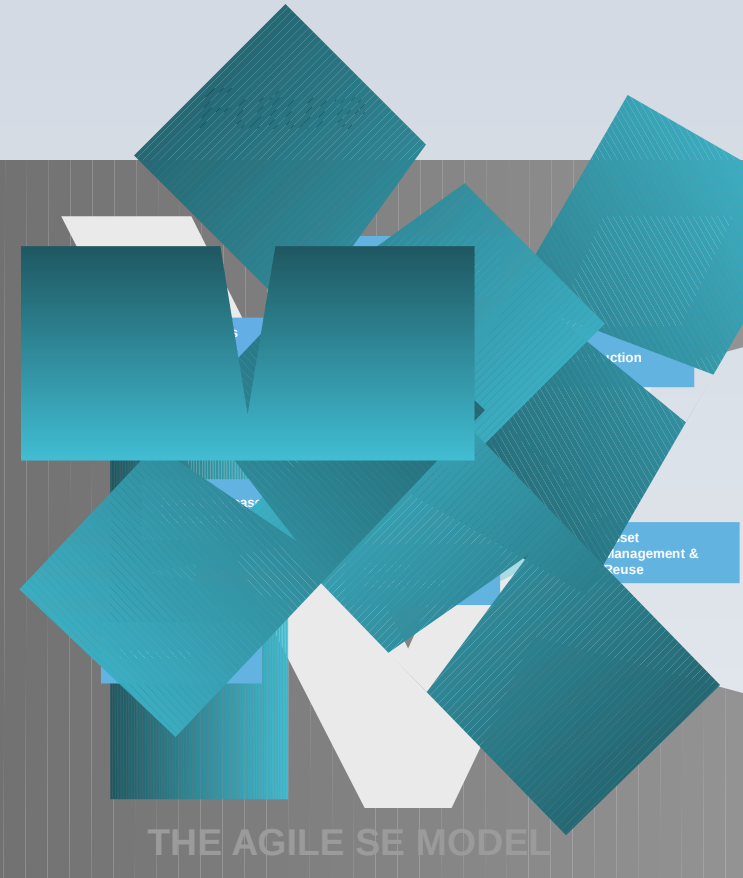


Modern Processes and Practices are Evolving

Past



Model-Based Engineering
Defect Avoidance
Defensive Design
Continuous Integration
Risk Management
Project Governance
Dynamic Planning



Moving from waterfall “ballistic” planning to incremental, adaptive approach

Key Concepts for Agility

- Improve quality through *continuous feedback*
 - Verification (*do it right*)
 - Analysis
 - Review
 - Testing via execution or simulation
 - Customer feedback (*meet the need*)
 - Correctness
 - Appropriateness
 - Usability
 - Defensive Design
- Efficiency through
 - Concentrate on high-value tasks
 - Avoid rework
 - Paying attention to how you're doing against goals
 - Project retrospective
 - Risk management
- Planning
 - Don't plan beyond the fidelity of the information you have
 - Plan enough but not more than that
 - Adjust plans based on "truth on the ground" (metrics)

Primarily build *executable things*
Verify them continuously
Validate them with the customer early & often

Active and continuous risk mitigation

Dynamic planning

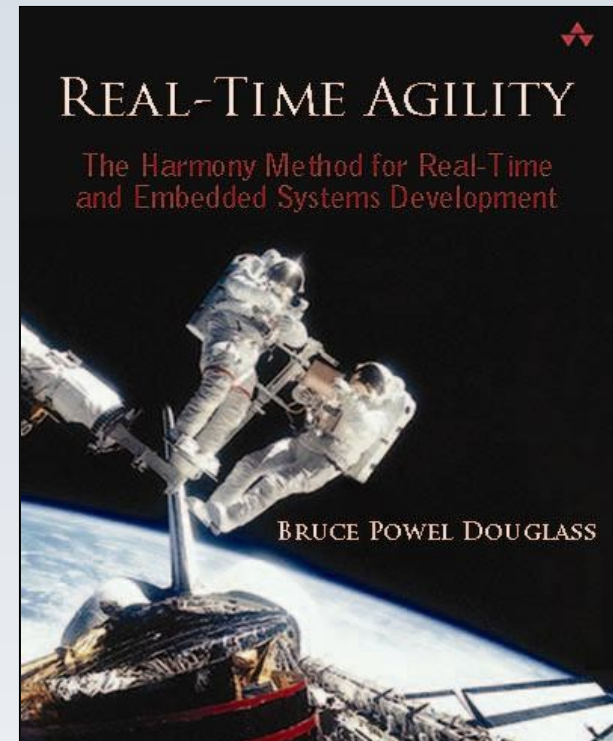
What does “agile” mean for Systems Engineering?

- Do what you need to do, no more and no less
 - This depends heavily on industry, regulation, and business environment
 - 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
- Verify continuously
 - With Q/A activities
 - With testing
 - With customer
- Outcome contains textual specifications but linked to executable & *verifiable* specifications
- Use dynamic planning to adjust project plans based on “truth on the ground”
 - Use goal-based metrics (KPIs) to track project progress
 - Continuously track progress against plan. Adjust planning frequently
- Safety, Reliability, Security
 - Not “done once” but continuously assessed
- Model-based hand off to downstream engineering

Best Practices for Agile Systems Engineering

- High-fidelity model-based engineering (Hi-MBE)
- Incremental functional analysis with use cases
- Executable requirements modeling with SysML / UML
- Test-driven development of system specifications
- Integrated safety and reliability analysis
- Model-based handoff to downstream engineering
- Automated document generation from model artifacts

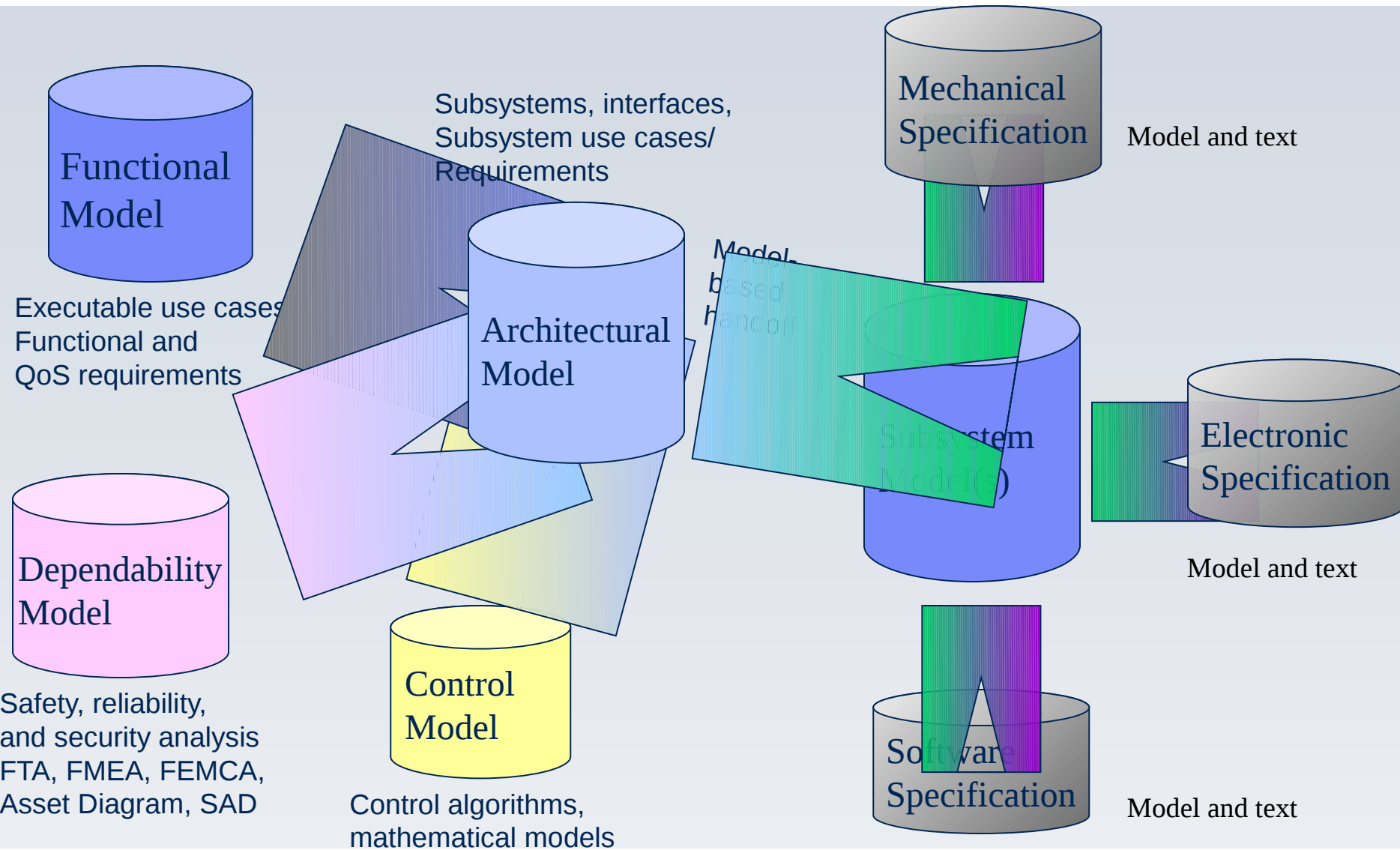
Note: a key difference between agile SW and agile SE is that the *outcome* of SE is *specifications* and the *outcome* of SW is *implementation*



But Why High-Fidelity Modeling???

- Hi-MBE brings to engineering
 - Precision
 - Verification via executability or formal methods
 - Stakeholder/Analysis-relevant viewpoints at any desired level of abstraction e.g.
 - Functionality
 - State-based behavior
 - Algorithmic/control behavior
 - Structure and Architecture
 - Integration of engineering work, e.g.
 - Functional requirements
 - Dependability analysis
 - Safety
 - Reliability
 - Cyberphysical security and Information Assurance
 - Architectural structure, behavior, and allocation
 - Control modeling and analysis

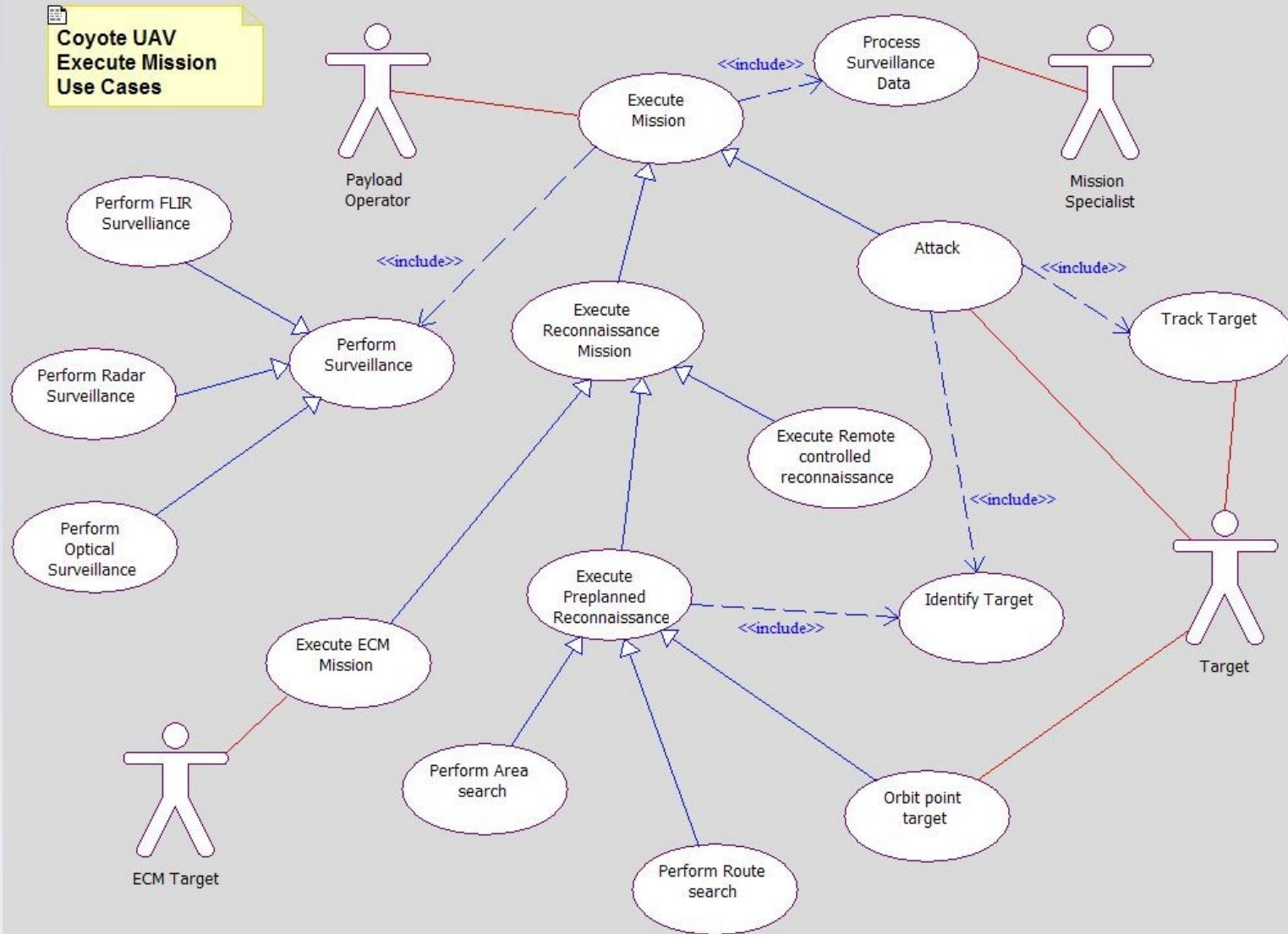
Models and Viewpoints in Model-Based Systems Engineering



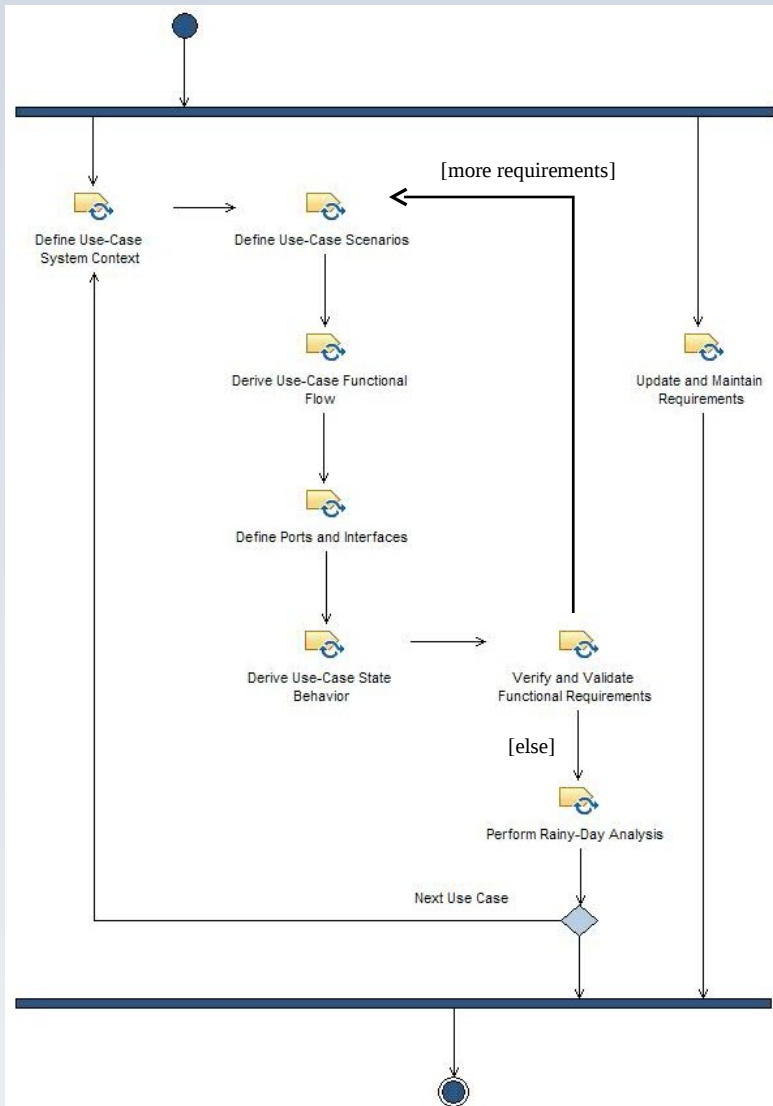
Use Cases for Systems Engineering

- A system-level use case
 - Can be thought of as a comprehensive user story
 - Is cross-discipline and usually results in mechanical, electronic and software implementation
 - Is a coherent set of requirements clustered around a systems capability or usage
 - Has bi-directional traceable links to system requirements
 - While focusing on functionality, *must* also include quality of service requirements and constraints, such as
 - Safety
 - Reliability
 - Security
 - Performance (worst-case, throughput, average case, etc)
 - Maintainability
 - Parametric requirements (heat, weight, recurring cost, etc)
- Three primary work flows for “detailing the use case”
 - Scenario-driven
 - Activity-driven
 - State-driven
 - It should be noted that the use case state machine is really the normative specification of the system with respect to that system capability

Detailing Use Cases with Statecharts and Scenarios



Scenario Driven Use Case Construction / Validation



Making it agile

1. Incrementally specify
 1. Use cases/User stories
 2. Specification nanocycles
2. Continuously verify
 1. Build executable models
3. Frequently validate
 1. Demonstrate to customer
4. Identify missing requirements
 1. Unspecified state transitions
 2. Unspecified scenarios
5. Incrementally add traceability
6. Annotate with qualities of service
7. Integrate with other specifications
 1. Other use cases
 2. Non-functional requirements
 1. Safety
 2. Security
 3. Reliability

Executable Requirements?

- A key problem is ensuring the quality of requirements for complex systems
- This is *not just* a matter of making good stand alone “shall” statements
 - Truly, most requirements specifications have poorly-specified “shall” statements that
 - Are ambiguous
 - Lack precision
 - Inappropriately include design detail
 - But the *hard part* is the subtle interactions of requirements
 - Don’t cover all cases, situations, or operational environments
 - Don’t discuss system error and fault responses
 - Missing data specification
 - Missing performance properties
 - Inconsistent requirements
- The “state of the practice” for requirements is protracted manual review of textual documents
 - The problem is this is error-prone and hugely expensive
- The “state of the art” is to create executable requirements models that *demonstrate* the system black-box behavior
 - The point is to *clearly and unambiguously specify the **control and data transformations** (and their constraints) to be performed by the system in all relevant situations, conditions, environments, and operational scenarios*

Executable Requirements

- Most common workflow
 - Start with textual specifications
 - Incrementally construct use case behavioral model
 - Test Driven Development
 - Nanocycle Iteration
 - Constrain with quality of service requirements
 - Continuous trace linking to textual requirements
 - Continuous safety, reliability, security assessment
- Outcome
 - Better requirements as workflow mitigates against
 - Incorrect requirements
 - Missing functionality
 - Missing QoS & performance requirements
 - Inconsistent requirements
 - Ambiguous requirements
 - Validation with the customer via execution of the use case
 - Model-based handoff to downstream engineering
 - Traceability supports impact analysis, change management, and safety certification

Validation of requirements through “what if” executions

- Virtual prototype / Panel graphics support
 - Ideal communications aid for validation of customer requirements
 - Clearly shows data and control transformations specified by requirements

The screenshot displays a software development environment with three main windows:

- Windows Internet Explorer:** Shows a web browser at `http://localhost:90/` displaying a virtual prototype of a "Bluetooth Headset" interface. The interface includes a "Builder[0]" component and links for "Bluetooth Headset" and "Nokia 6210".
- Rhapsody in C++:** The main IDE window showing the project structure on the left and statecharts on the right.
 - Entire Model View:** A tree view showing the project hierarchy, including packages like "AnalysisPkg", "PresentationPkg", "InterfacePkg", "MobilePhonePkg", and "HeadsetPkg".
 - Statechart of : Button - Builder[0]->itsHeadset->itsButton:** A statechart with states "idle", "debounce", and "pressed". Transitions include "evPress", "evRelease", and "tm(100)".
 - Statechart of : Headset - Builder[0]->itsHeadset:** A statechart with states "off", "disconnected", "connected", and "connecting". Transitions include "evLongPress", "evActive", and "evConnect".
- Call Stack and Event Queue:** Located at the bottom of the IDE, showing the current execution context.

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* and less about *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
 - 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

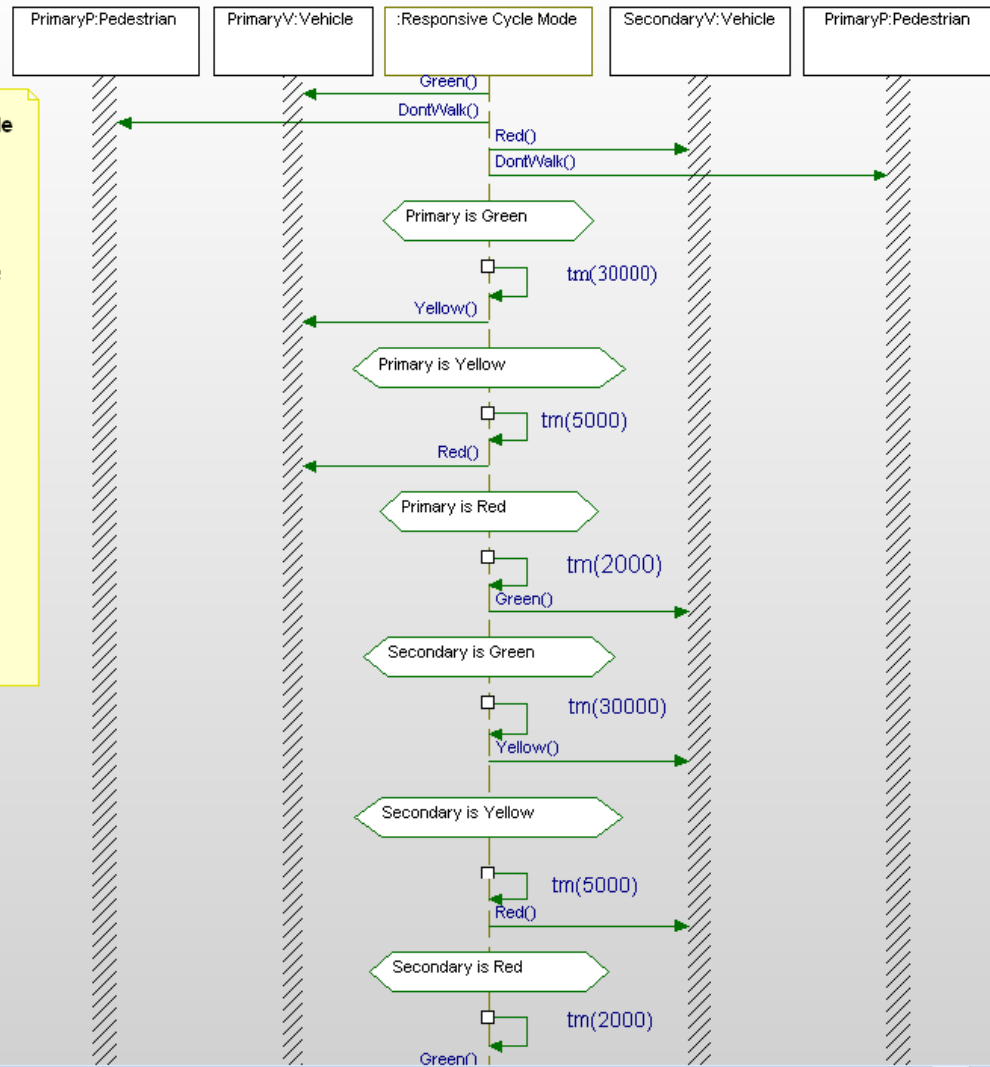
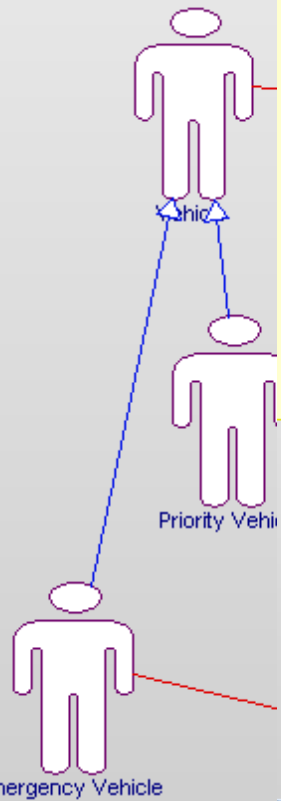
Example: Traffic Light Control Fixed Cycle Mode Spec

RoadRunner Traffic Li
System Use Cases

Use Case: Responsive Cycle Mode

Scenario 1: No Traffic

Preconditions:
 Mode: Responsive Cycle Time
 Primary and secondary roads parameters set the same
 Road directions: DUAL
 Turn lanes: TRUE
 Turn lane mode: SIM
 Pedestrian lights: TRUE
 Green Time 30
 Yellow Time 5
 Red Delay Time 2
 Walk time 20
 Wam Time 10
 Green Turn Time 20
 Green Yellow Time 5
 Primary light is Green.
 Secondary light is Red.



Example: Traffic Light Control Fixed Cycle Mode Spec

RoadRunner Traffic Light System Use Cases

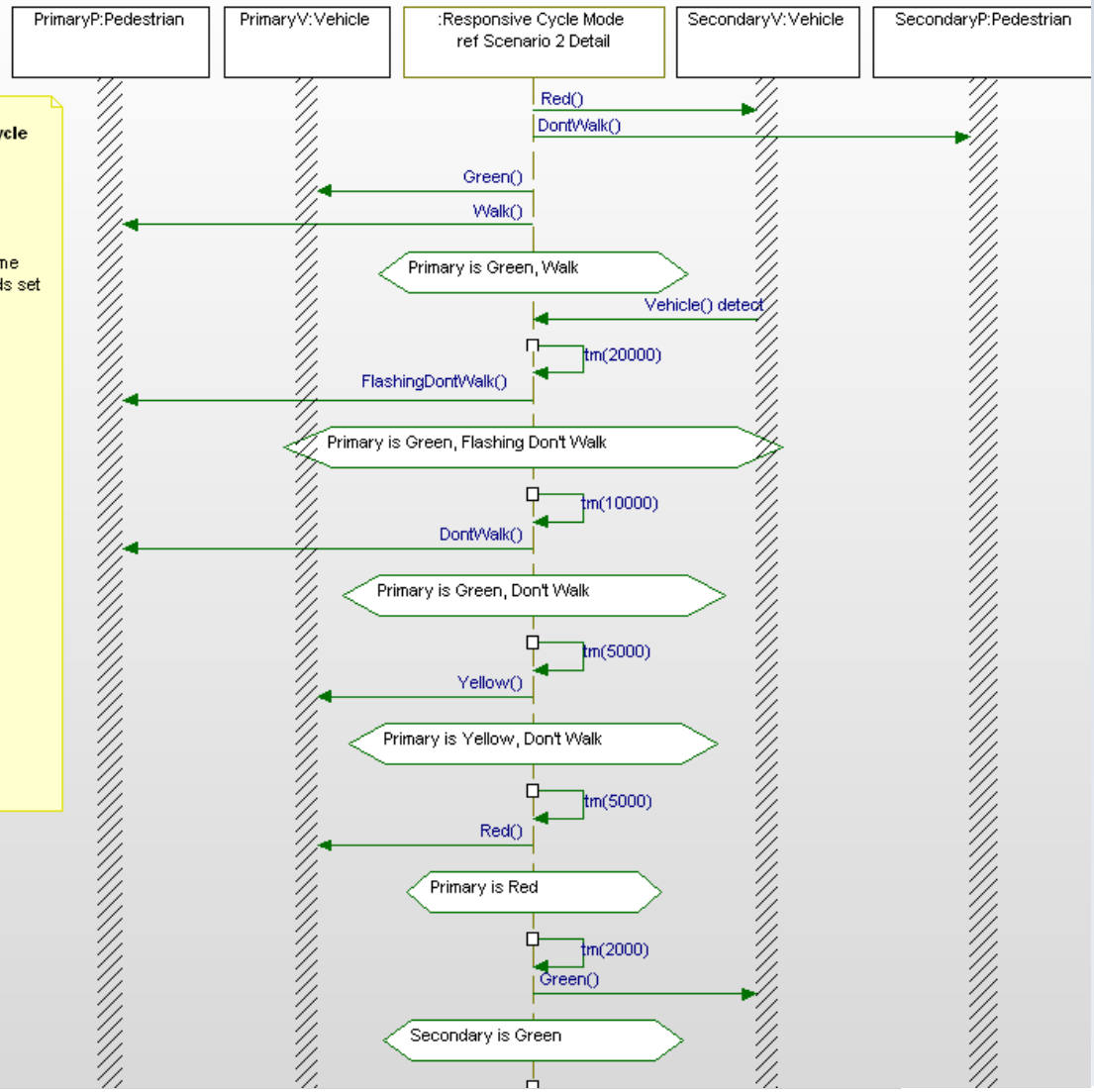
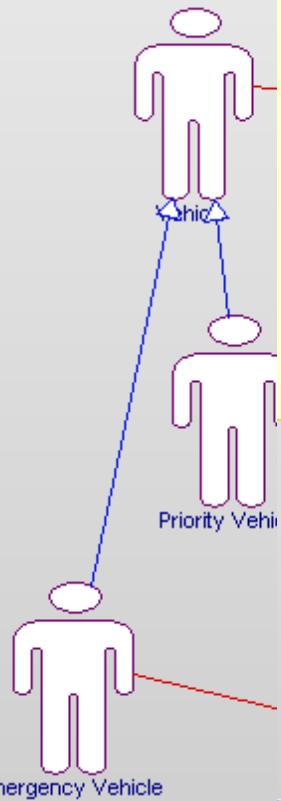
Use Case Mode

Scenario

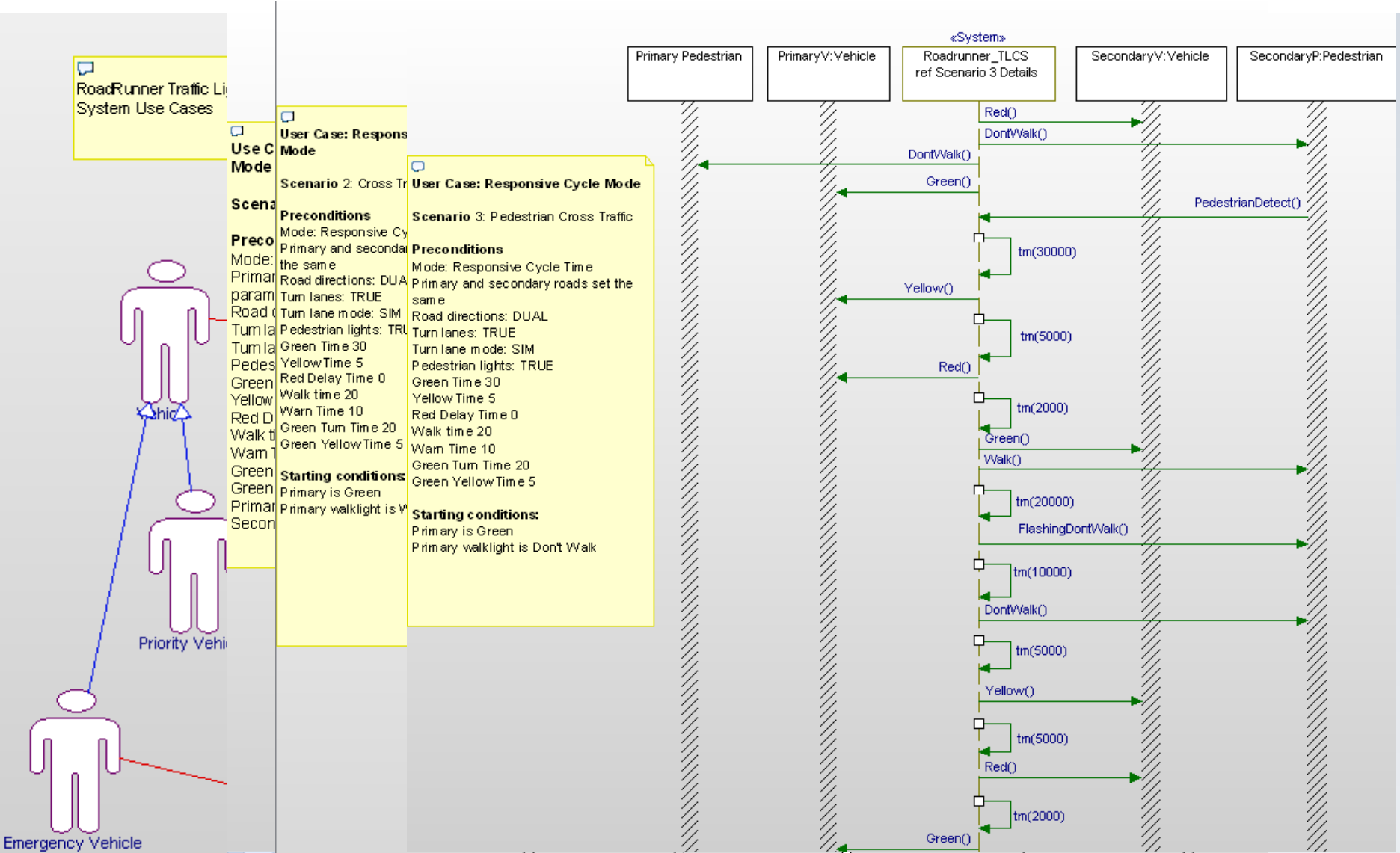
Preconditions

Starting conditions

User Case: Responsive Cycle Mode
Scenario 2: Cross Traffic
Preconditions
 Mode: Responsive Cycle Time
 Primary and secondary roads set the same
 Road directions: DUAL
 Turn lanes: TRUE
 Turn lane mode: SIM
 Pedestrian lights: TRUE
 Green Time 30
 Yellow Time 5
 Red Delay Time 0
 Walk time 20
 Warn Time 10
 Green Turn Time 20
 Green Yellow Time 5
Starting conditions
 Primary is Green
 Primary walklight is WALK



Example: Traffic Light Control Fixed Cycle Mode Spec



Example: Traffic Light Control Fixed Cycle Mode Spec

RoadRunner Traffic Light System Use Cases

User Case: Responsive Cycle Mode

Scenario 2: Cross Traffic

Preconditions
 Mode: Responsive Cycle Mode
 Primary and secondary roads the same
 Road directions: DUAL
 Turn lanes: TRUE
 Turn lane mode: SIM
 Pedestrian lights: TRUE
 Green Time 30
 Yellow Time 5
 Red Delay Time 0
 Walk time 20
 Warn Time 10
 Green Turn Time 20
 Green Yellow Time 5

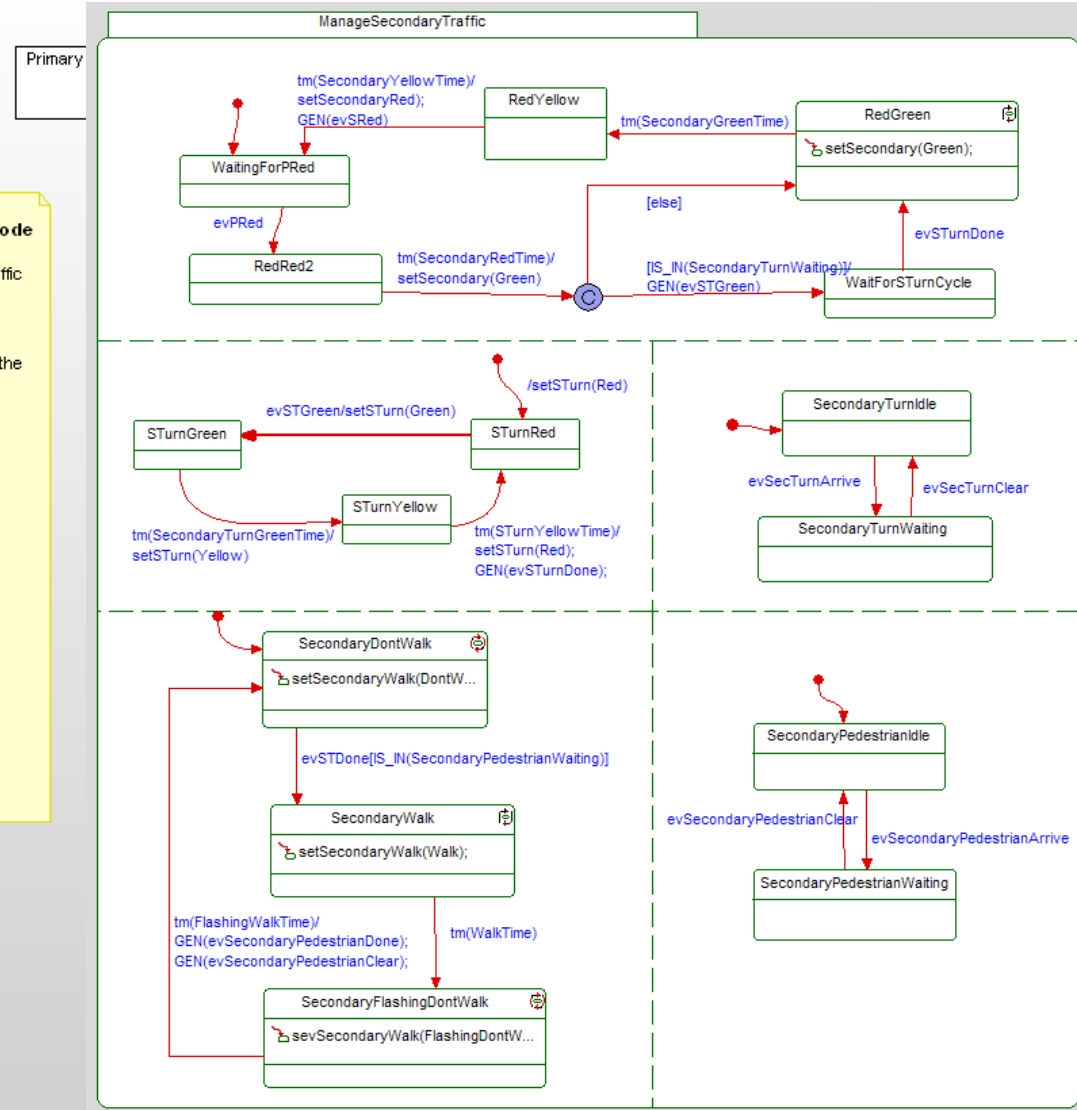
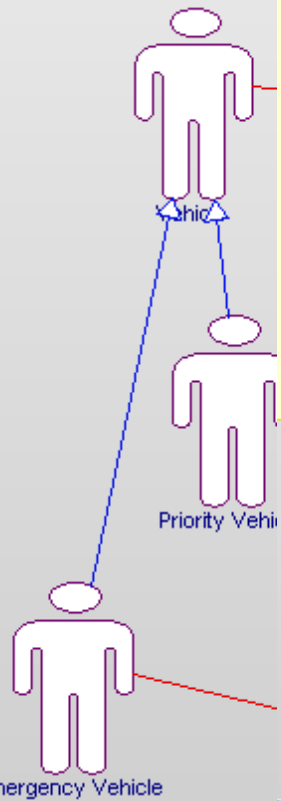
Starting conditions
 Primary is Green
 Primary walklight is V

User Case: Responsive Cycle Mode

Scenario 3: Pedestrian Cross Traffic

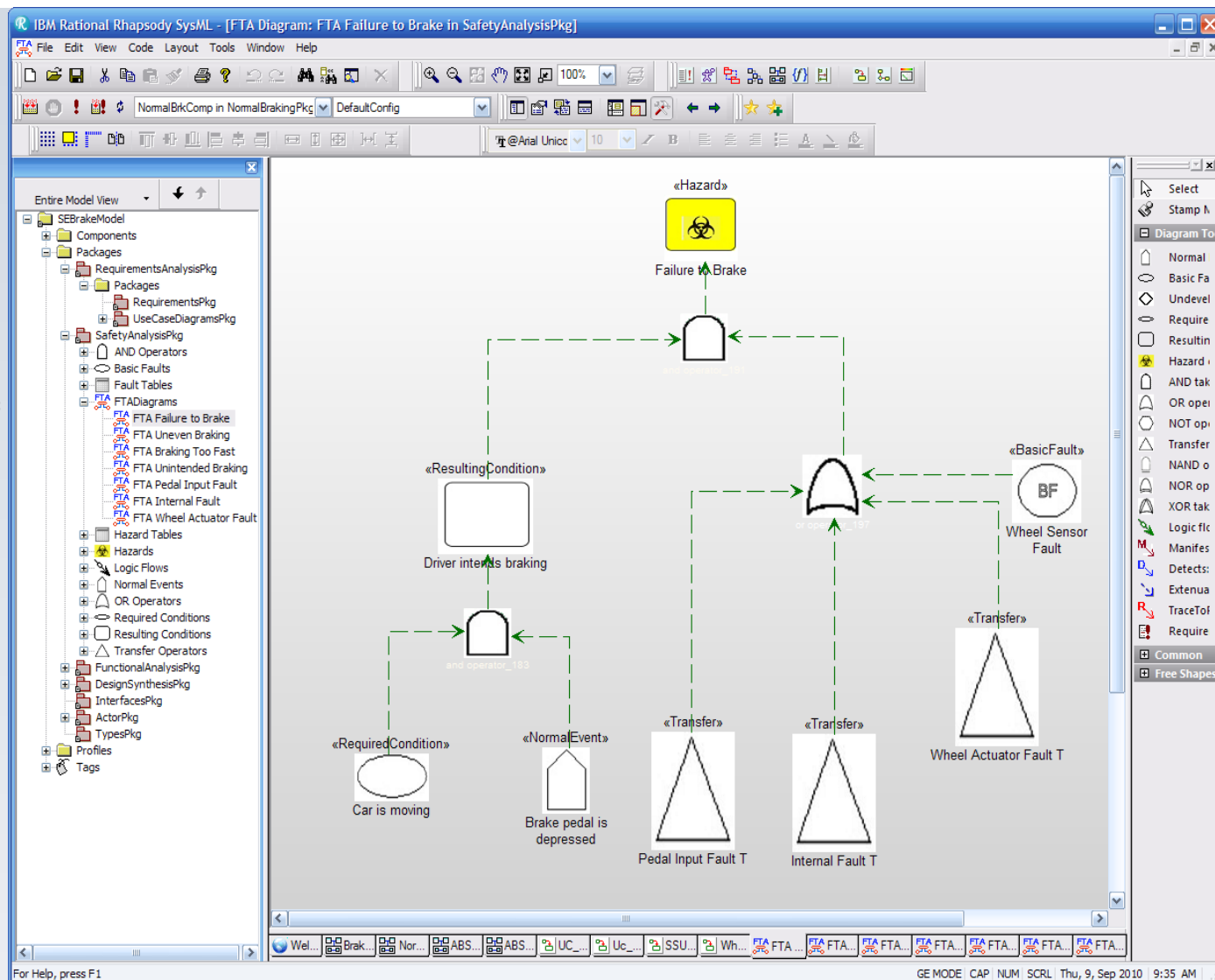
Preconditions
 Mode: Responsive Cycle Mode
 Primary and secondary roads set the same
 Road directions: DUAL
 Turn lanes: TRUE
 Turn lane mode: SIM
 Pedestrian lights: TRUE
 Green Time 30
 Yellow Time 5
 Red Delay Time 0
 Walk time 20
 Warn Time 10
 Green Turn Time 20
 Green Yellow Time 5

Starting conditions
 Primary is Green
 Primary walklight is Dont Walk

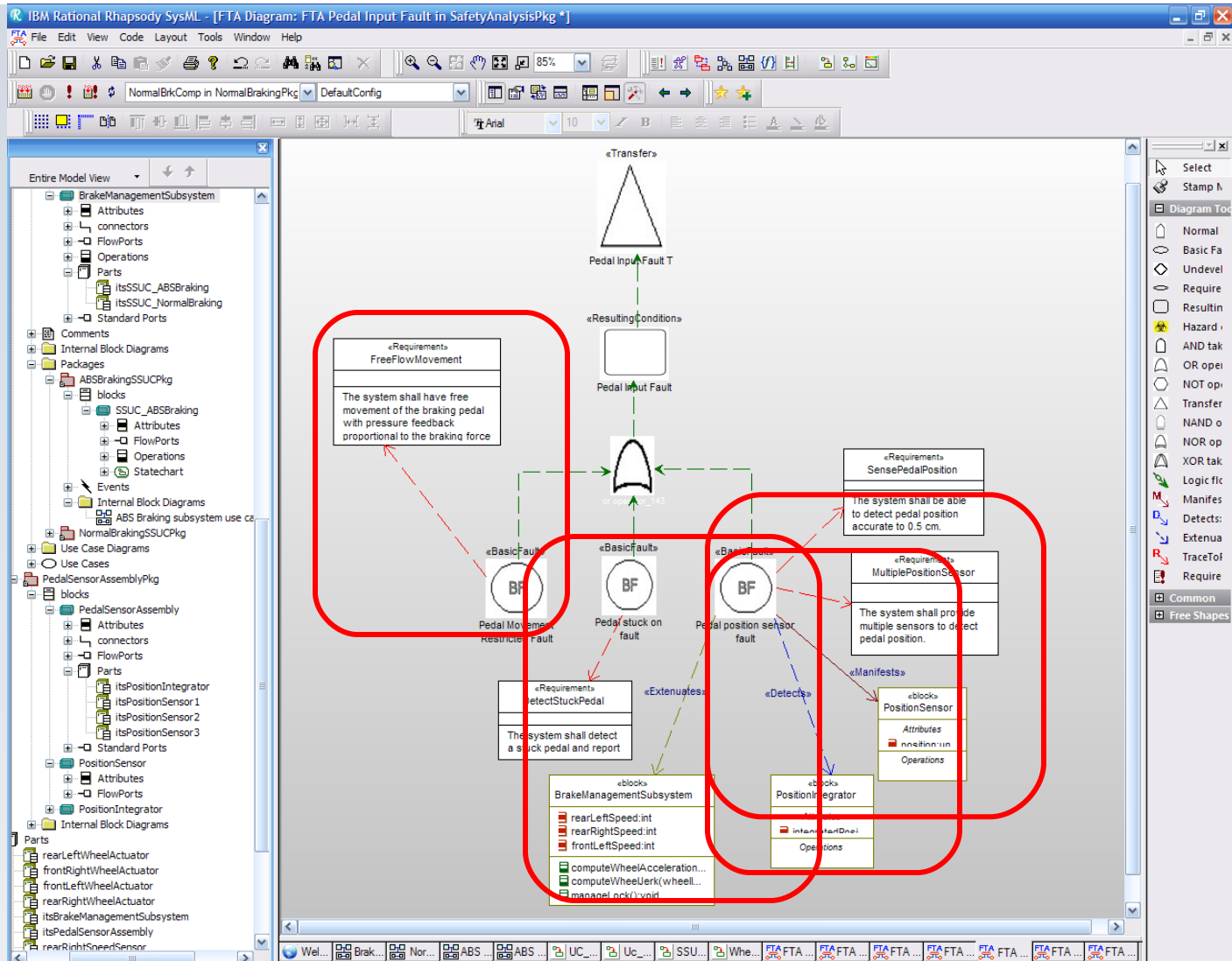


Integrated Safety and Reliability Analysis

- Fault Tree Analysis (FTA) connects *hazards* with logical combinations of events, conditions, errors, and faults
- Allows you to identify
 - Effects of combinations of conditions and events on safety
 - Safety measures
 - Safety requirements
 - DO 178B/C
 - IEC 61508
 - ISO 26262
 - Impacts of architectural, technological, and design choices on safety



Integrated Safety and Reliability Analysis



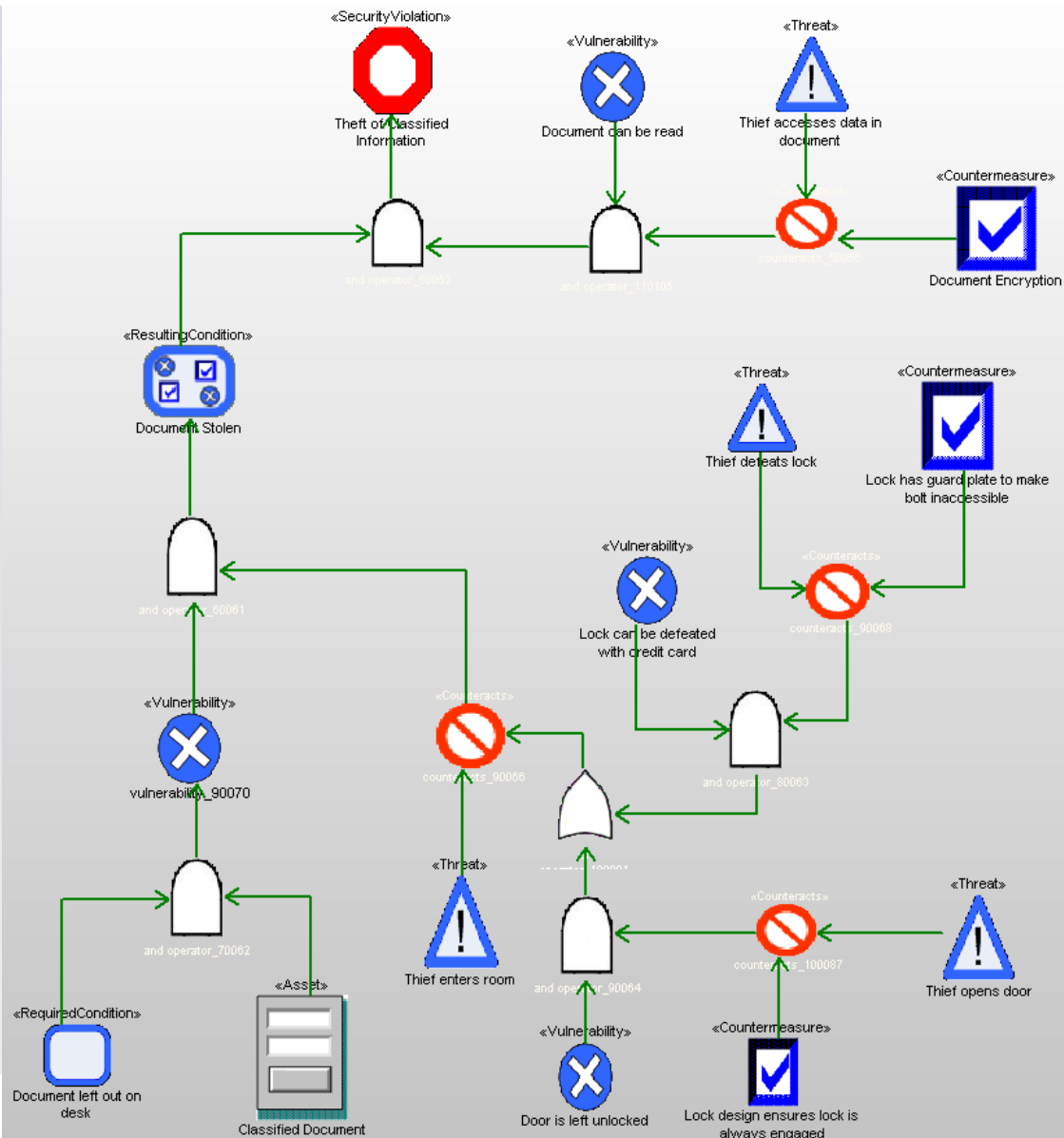
For Help, press F1

GE MODE | CAP | NUM | SCRL | Thu, 9, Sep 2010 | 9:51 AM

Model-Based Threat Analysis

• **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



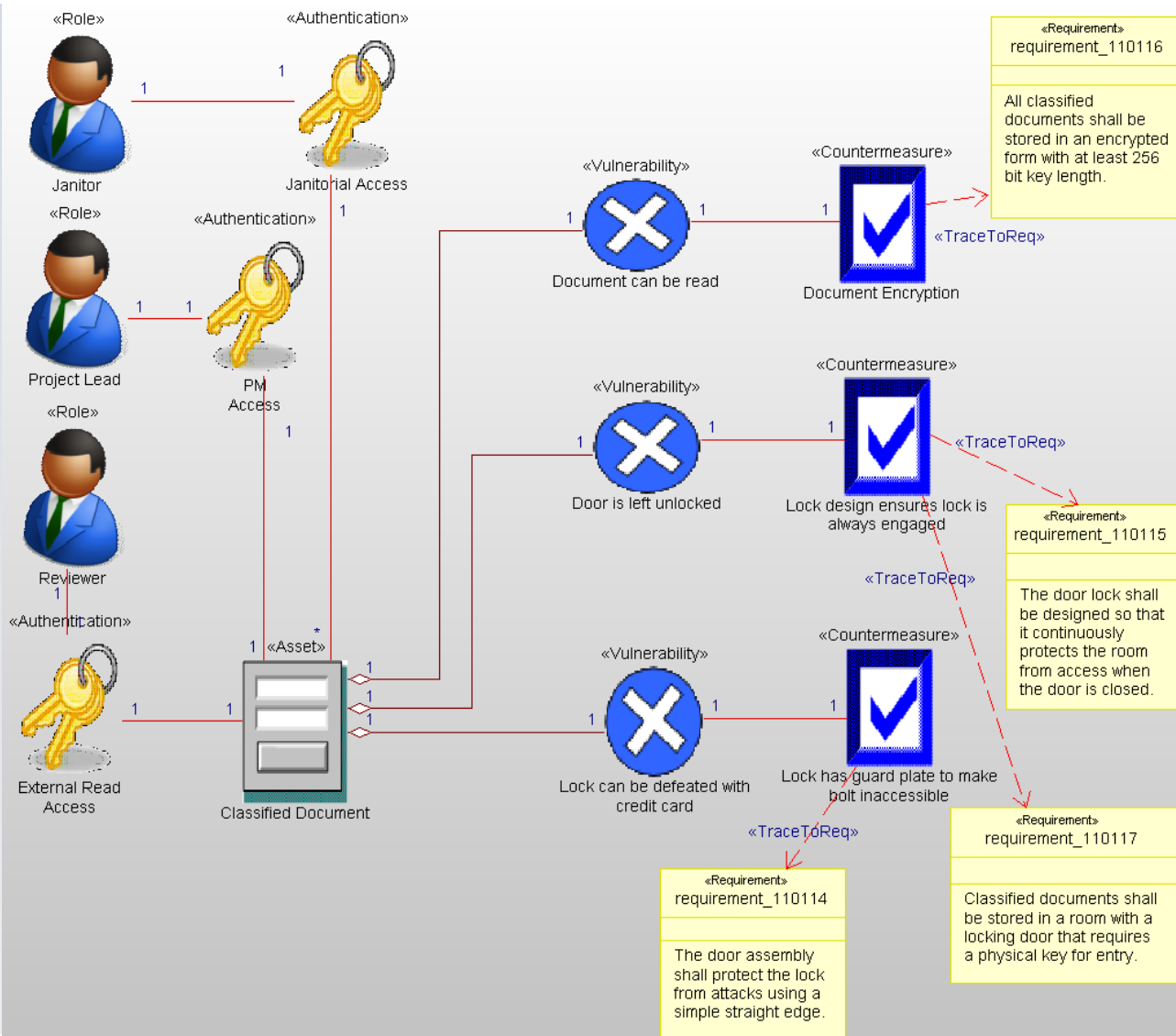
Model-Based Threat Analysis

- An **Asset Diagram** looks at the semantic relations between roles, authentication, vulnerabilities, and countermeasures. It is a way of representing the security-relevant design elements.

- Here it is shown with traceability links to requirements

- Assets are things you want to protect, e.g.

- Physical
- Informational
- Currency
- Resource
- Security



Auto-generation of dependability-relevant summary data

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

To: Class, SafetyMeasure Scope: DesignModel

AlarmManager	GasFlowSensor	Pump	PressureSensor	SpO2Sensor	GasValve	PumpController	O2Sensor	PowerSupplyRegulator
Gas Supply Fault					GasValve			
Ventilator Pump Fault		Pump						
Ventilator Parameter Setting wrong						PumpController_0		

To: Class, SafetyMeasure Scope: DesignModel

GasFlowSensor	PressureSensor	PumpController	GasMixer	PowerSupplyRegulator	Battery	ProtectedCRCClass	CO2Sensor
---------------	----------------	----------------	----------	----------------------	---------	-------------------	-----------

To: Requirement Scope: RequirementsAnalysis

	REQ_BCM_09	REQ_BCM_11	REQ_VD_03	REQ_VD_04	REQ_VD_06	REQ_SpO2_01	REQ_VD_08	REQ_VD_10	REQ_VD_11
Gas Supply Fault			REQ_VD_03	REQ_VD_04	REQ_VD_06		REQ_VD_08		
Breathing Circuit Leak			REQ_VD_03	REQ_VD_04	REQ_VD_06				
Ventilator Pump Fault					REQ_VD_06				
Ventilator Parameter Setting wrong									
Ventilator Computation Incorrect									
Esophageal Intubation									
Patient disconnect from Breathing Circuit									
Power Supply Fault									
O2 Supply Fault									
Redundant computational Channel fails									
Ventilator Parameter Limiting Fails									
Ventilator Parameter CRC check fails									
Backup Power Fails									
SpO2 Sensor Fault									
Breathing Circuit O2 Sensor Fault									
Expiratory Limb CO2 sensor fault									

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 s	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

• Traceability improves your ability to make your safety/security case

Dependability metadata guides

- System requirements
- Downstream engineering work
- Regulatory approval submissions

Where am I | Tree Sets

Harmony/SE

- Introduction to IBM® Rational® Harmony™
- Getting Started with Harmony
- Core Principles
- Harmony for Systems Engineering
- Disciplines
- Domains
- Roles
- IBM® Rational® Tools
- References
- About IBM® Rational® Harmony™
- 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](#)

Harmony Systems Engineering Workflows

IBM Rational Harmony for Systems Engineering

Glossary | Index | Feedback | About

Print

Where am I | Tree Sets

Harmony/SE

- Introduction to IBM® Rational®
- Getting Started with Harmony
- Core Principles
- Harmony for Systems Engineering
- Disciplines
- Domains
- Roles
- IBM® Rational® Tools
- References
- About IBM® Rational® Harmony™
- IBM® Rational® Harmony™ for Systems Engineering

Harmony for Systems Engineering Lifecycle > Requirements Analysis > Perform Initial Safety and Reliability Analysis

Task: Perform Initial Safety and Reliability Analysis

This task creates performs initial safety and reliability analysis and captures the results in a hazard analysis document.

Expand All Sections Collapse All Sections

The purpose of this task is to identify and clarify the initial safety and reliability issues of the system for the purpose of identifying relevant safety and reliability requirements.

Back to top

Relationships

Roles	Main:	Additional:	Assisting:
	<ul style="list-style-type: none"> Safety Czar 	<ul style="list-style-type: none"> Reliability Czar 	

Inputs	Mandatory:	Optional:	External:
	<ul style="list-style-type: none"> System Requirements Specification 	<ul style="list-style-type: none"> None 	<ul style="list-style-type: none"> None

Outputs	
	<ul style="list-style-type: none"> Failure Modes and Effect Analysis (FMEA) Fault Tree Analysis (FTA) Hazard Analysis

Back to top

Main Description

The hazard analysis is a key document that captures hazards, risks, faults, and control measures together. The control measures mitigate the risks and so must be captured as requirements on the system to manage the risks to an acceptable level.

Back to top

Steps

Expand All Steps Collapse All Steps

- Create initial hazard analysis
- Identify hazards
- Quantify risks

Back to top

Properties

Key Considerations

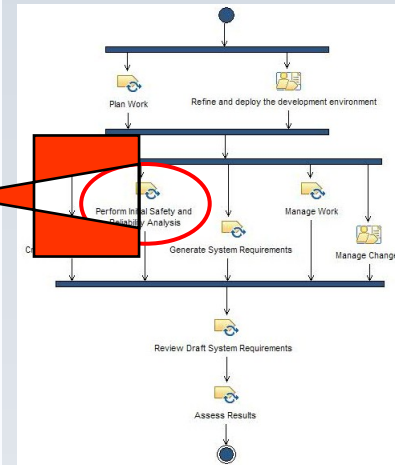
More Information

Guidelines	
	<ul style="list-style-type: none"> Fault Tree Analysis Guideline Hazard Analysis Guideline

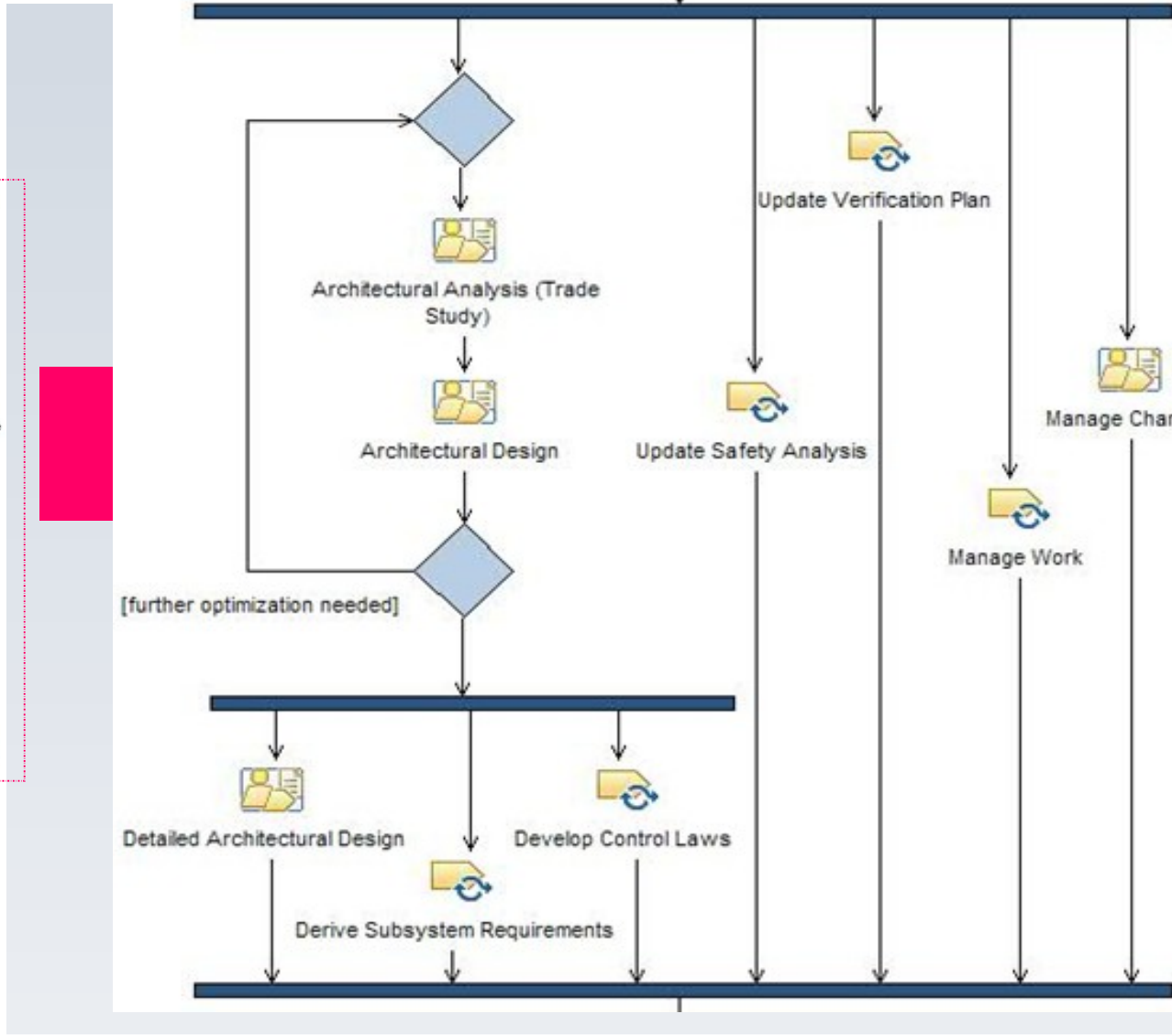
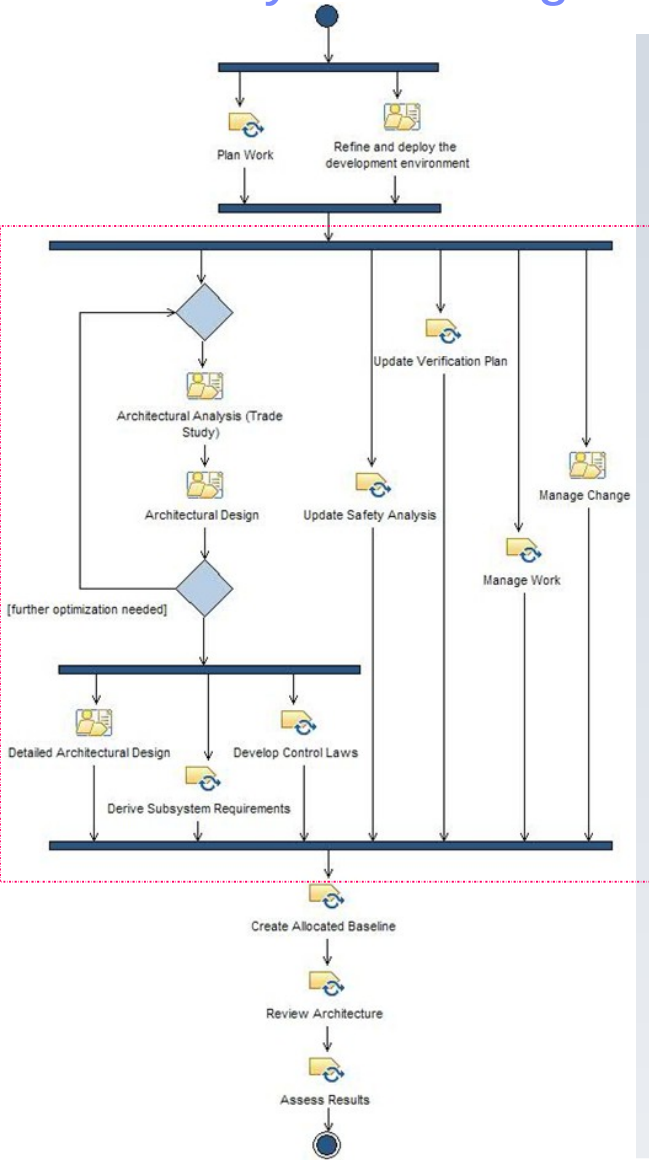
Back to top

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

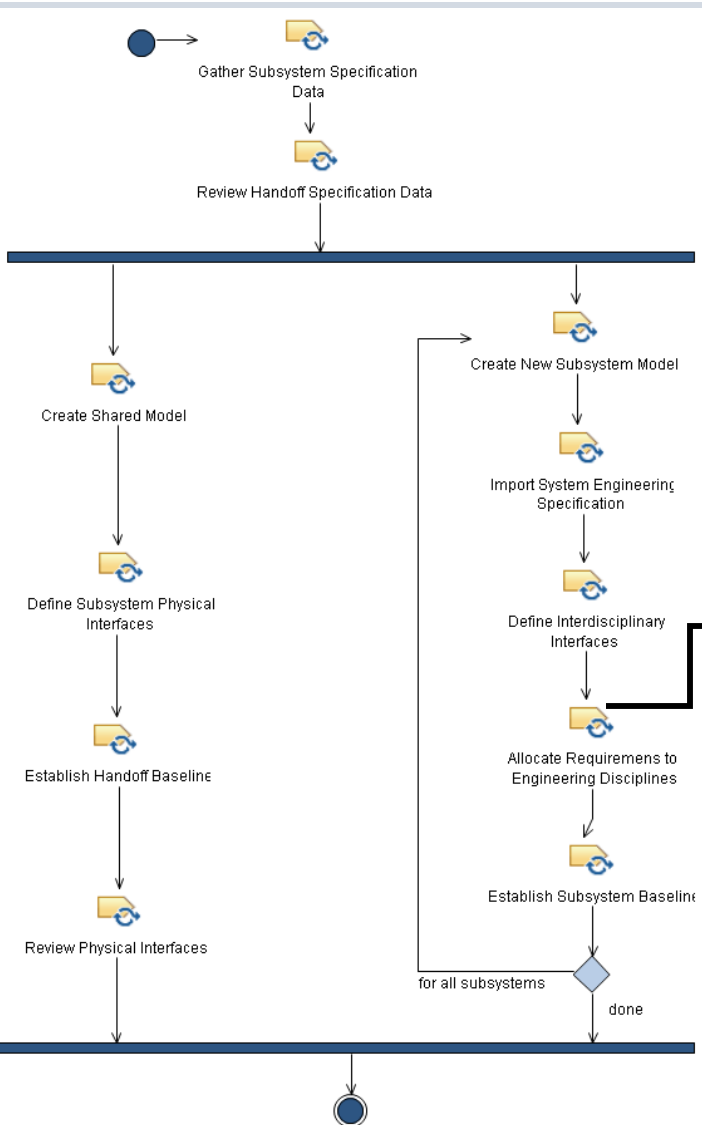
Systems Engineering: Requirements Analysis



Harmony/SE: Design Synthesis



Model-Based Hand-off to Downstream Engineering



Task: Allocate Requirements to Engineering Disciplines

This task takes the requirements allocated to a subsystem as a whole and allocates them to the different engineering disciplines involved (e.g. software, electronics, mechanical, optical, hydraulic).

Expand All Sections Collapse All Sections

Purpose

The purpose is to clearly delineate the required contributions of different engineering disciplines to the engineering development of a subsystem by allocating the requirements allocated to the subsystem by the system engineering team.

Back to top

Relationships

Roles	Primary Performer: <ul style="list-style-type: none"> Control Engineer Developer Electrical Engineer Mechanical Engineer Miscellaneous Engineer 	Additional Performers: <ul style="list-style-type: none"> Architect Reliability Czar Safety Czar
Inputs	Mandatory: <ul style="list-style-type: none"> Requirements Traceability Subsystem Model Systems Requirements Specification 	Optional: <ul style="list-style-type: none"> Failure Modes and Effect Analysis Fault-Tree Analysis Hazard Analysis
Outputs	<ul style="list-style-type: none"> Requirements Traceability Subsystem Requirements Specification 	
Process Usage	<ul style="list-style-type: none"> handoff_cp > Allocate Requirements to Engineering Disciplines 	

Back to top

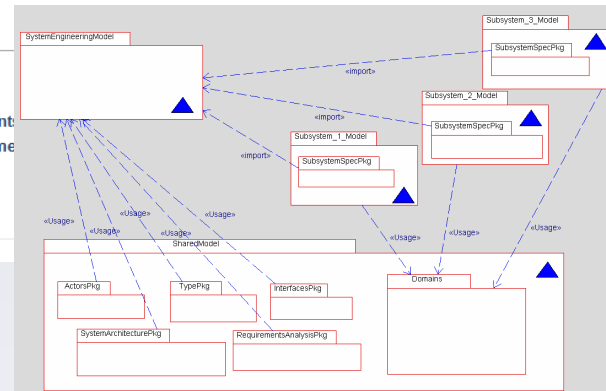
Main Description

A subsystem team is usually comprised of engineers within different disciplines, such as software, digital electronics, analog electronics, hydraulic, pneumatic, control, and mechanical. Once the subsystem specification is handed off, certain of the requirements will belong to one discipline or the other. Other requirements will require decomposition into derived requirements, allocating portions of a subsystem-level requirement to different discipline. This is particularly true of quality-of-service requirements.

Back to top

Steps

- Review subsystem requirements
- Allocate single-discipline requirement
- Decompose multi-discipline requirement
- Allocate derived requirements
- Update traceability record
- Review allocations



Back to top

Collapse All Steps

Back to top

Summary

- Systems Engineering capability can be greatly enhanced with two key technologies
 - Use of SysML/UML Modeling to capture system
 - Behavior (executable use cases)
 - Structure (architecture)
 - Data and performance modeling
 - Model-based hand off to downstream engineering
 - Automatic generation of documentation from model-based work products
 - Agile methods employing
 - Incremental construction of use cases
 - Test Driven Development nanocycle-level iteration
 - Incorporating dependability analysis with the use case development
- Harmony best practice workflows embody agile for embedded systems engineering
 - Rational provides the services
 - R&D Capability Assessment
 - “Agile Systems Engineering” and “Agile Embedded Software Development” courses
 - Rapid Deployment Package (training, mentoring, frequent design reviews by modeling experts)
 - Rational has the tools
 - **Rational DOORS** provides requirements management with great traceability
 - **Rational Rhapsody** provides tooling for best practice realization
 - **Rational Method Composer** manages process and practice definitions
 - **Rational Team Concert** provides a project enactment & governance environment
 - **Rational Quality Manager** manages test cases and procedures

QUESTIONS

www.ibm.com/software/rational





www.ibm.com/software/rational

© Copyright IBM Corporation 2012. All rights reserved. The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way. IBM, the IBM logo, Rational, the Rational logo, Telelogic, the Telelogic logo, and other IBM products and services are trademarks of the International Business Machines Corporation, in the United States, other countries or both. Other company, product, or service names may be trademarks or service marks of others.

