



IBM Software Group

Model-Based Ada Development for DO-178C Projects and Agile Methods

Bruce Powel Douglass, Ph.D.

Chief Evangelist

Global Technology Ambassador

IBM Rational

Bruce.Douglass@us.ibm.com

Twitter: [@BruceDouglass](https://twitter.com/BruceDouglass)

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

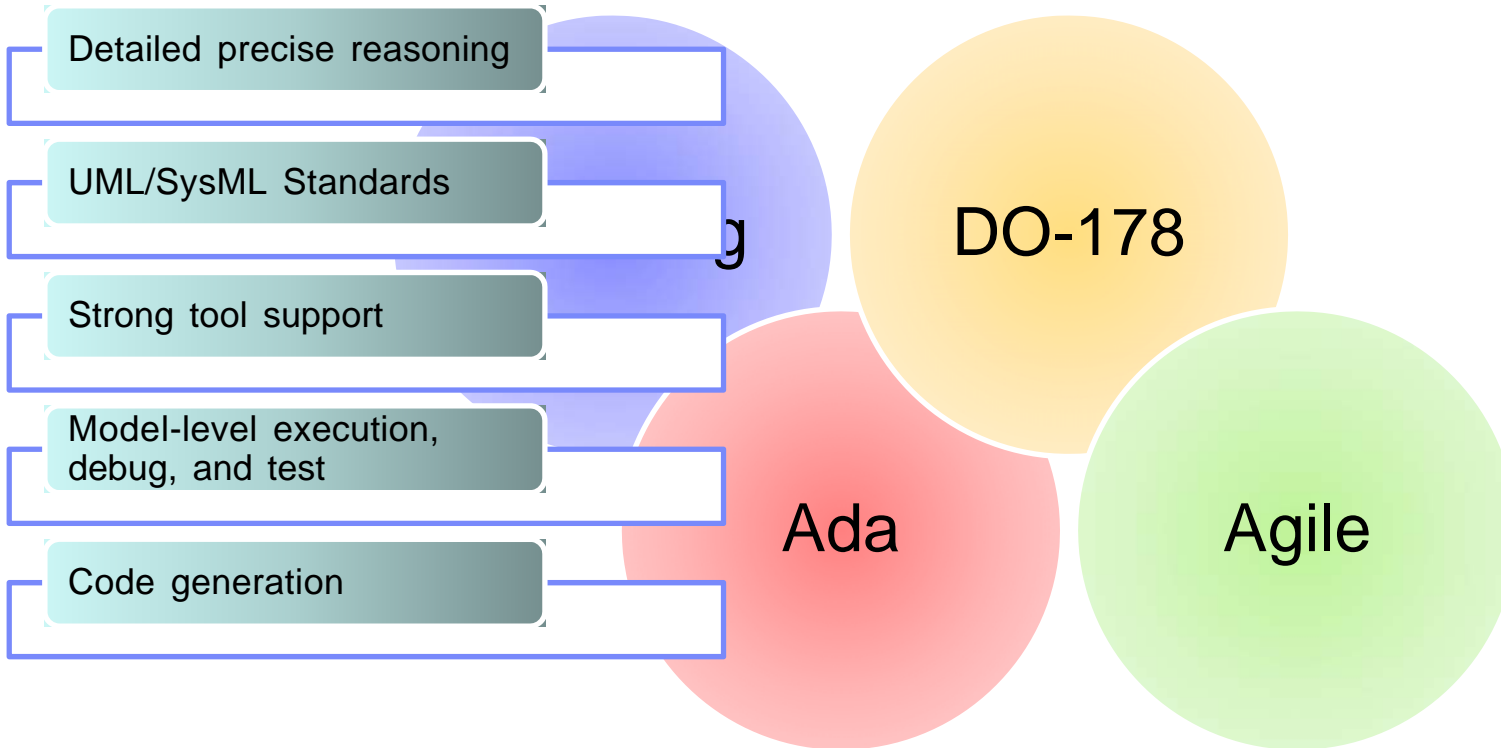
IBM: ibm.co/brucedouglass



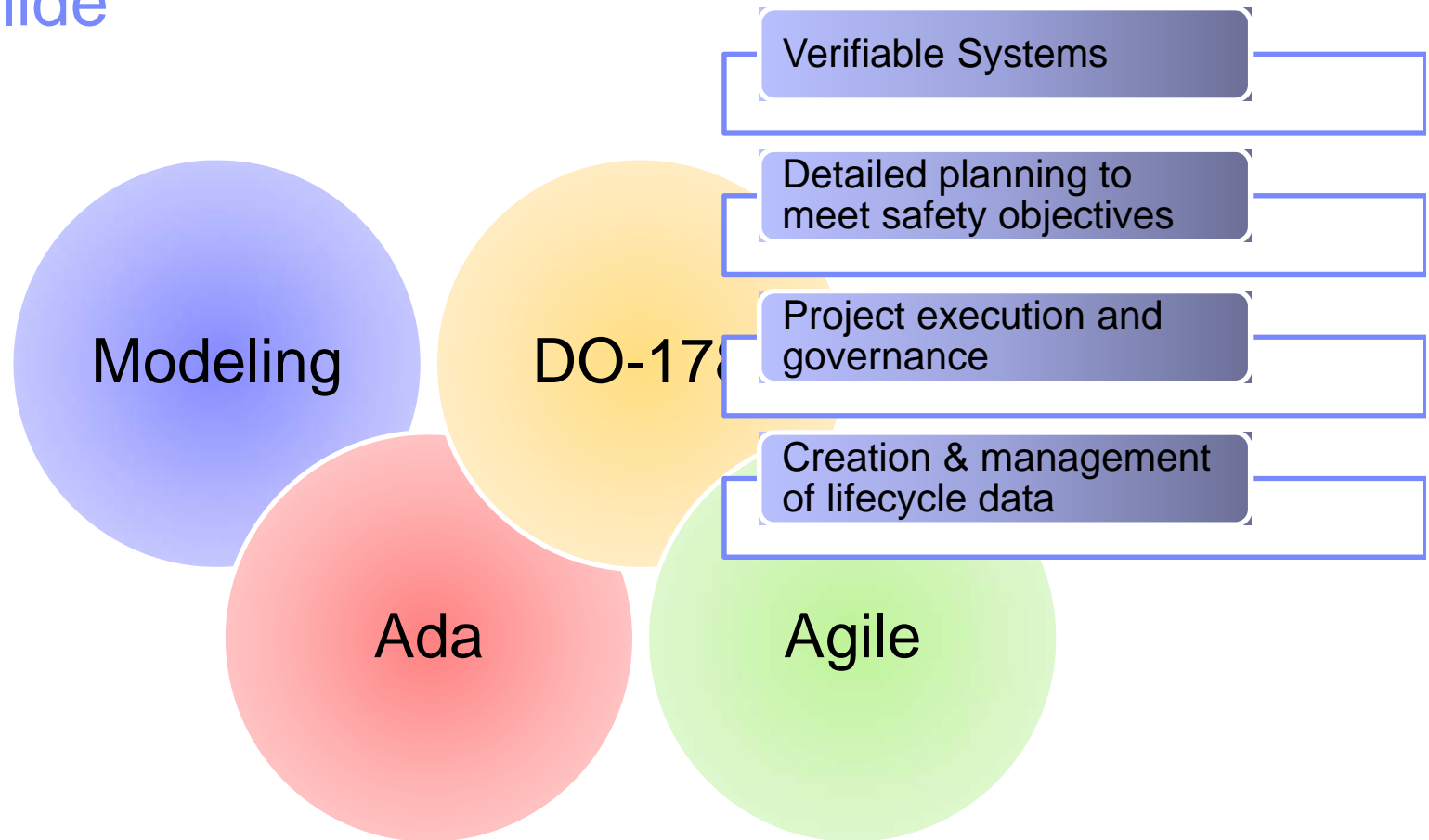
Rational. software

© 2013 IBM Corporation

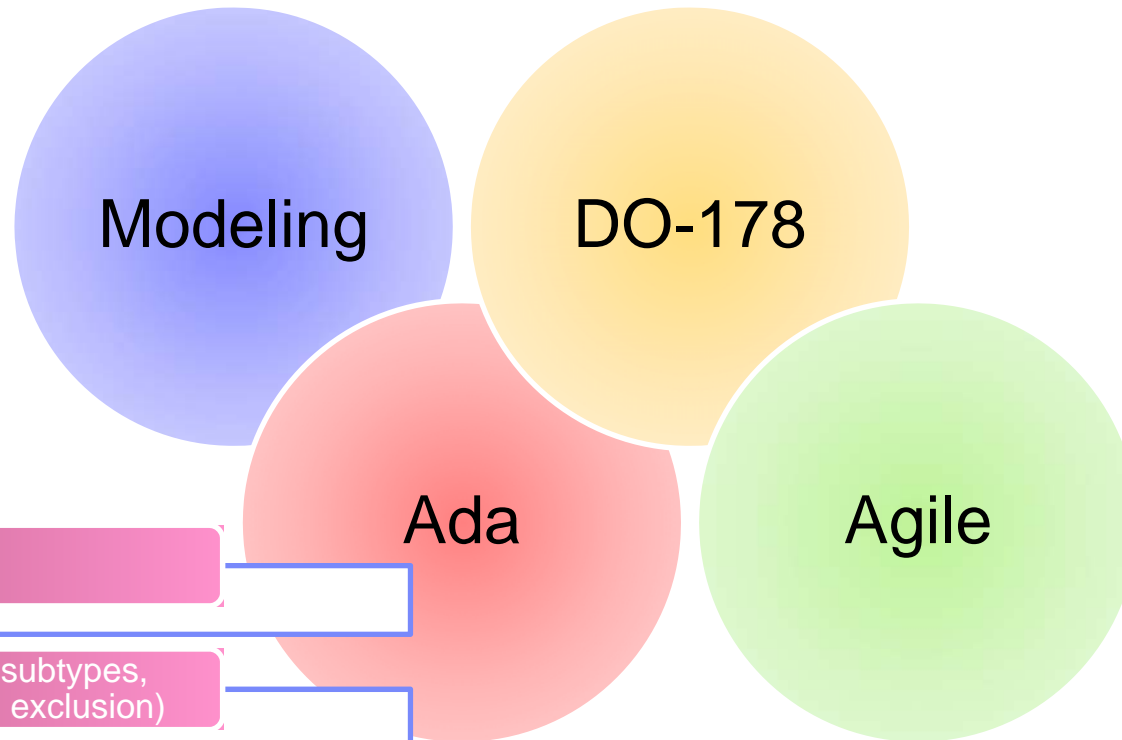
Worlds Collide



Worlds Collide



Worlds Collide



Type safety

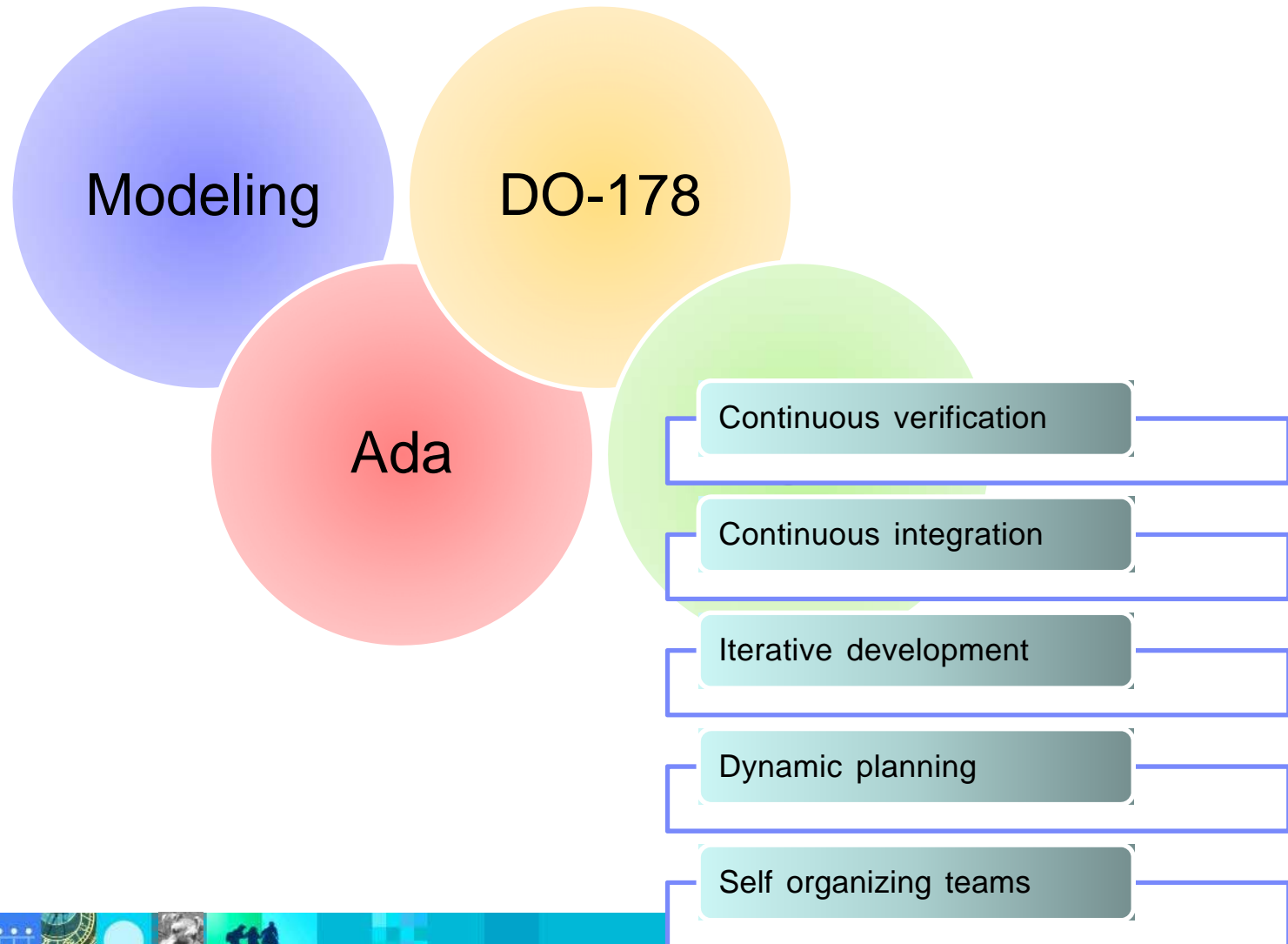
Modern (e.g. OO, subtypes, task types, mutual exclusion)

Targeted towards embedded real-time systems

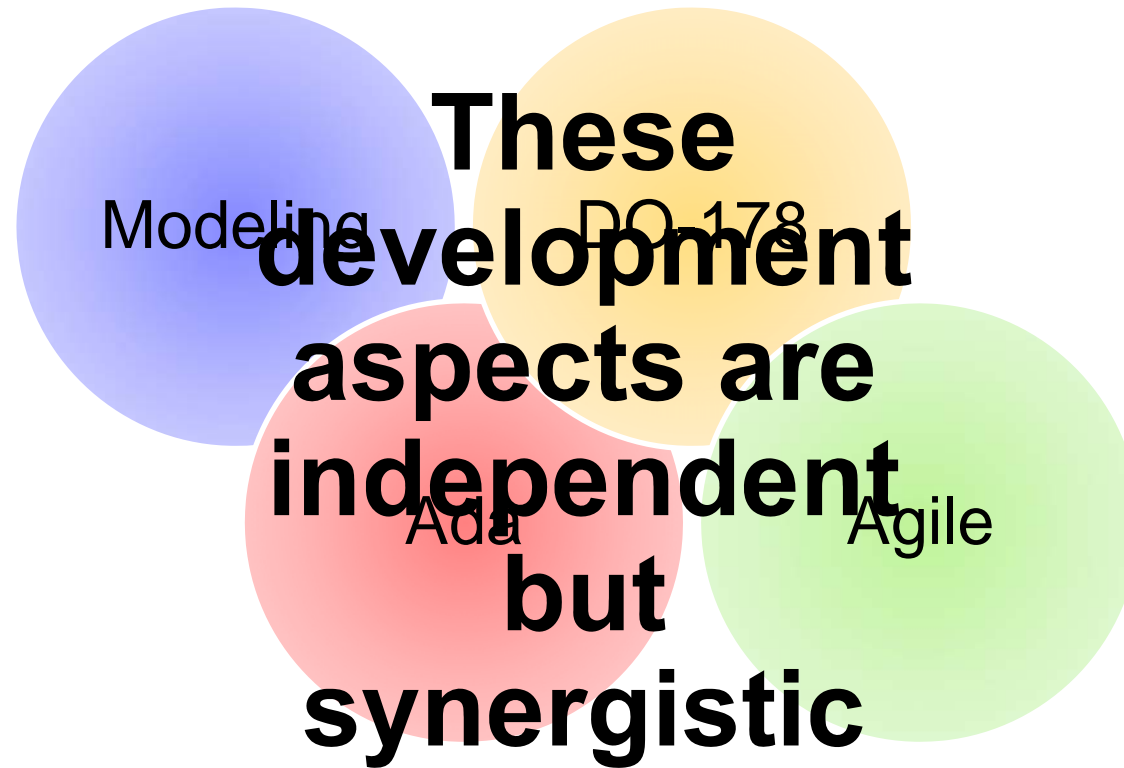
Run-time checking



Worlds Collide



Just to be clear ...



Advantages of (good) modeling

Understandability

Reasoning

Modeling

Architecture

Test cases

Verifiability

Design

Requirements

Consistency of models and code

Code generation

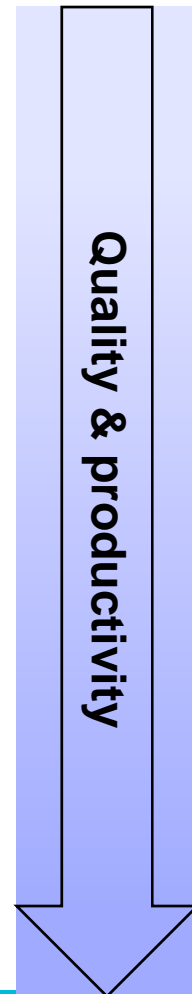
Precision

Handoff



UML Maturity Model Index (UMMI)

Level	Benefit	Focus	Technologies
0 Code Based Development	0%	Manually writing code	Editor, compiler
1 Visualization	5%	Visualizing code structures	Reverse engineering
2 Structural Modeling	15%	Class and block modeling of structure	Class and block diagrams
3 Behavioral Modeling	30%	State and algorithmic modeling	State, sequence and activity diagrams
4 Executing	70%	Model-based verification	Model execution, code generation, model-based debugging
5 Optimizing	100%	Agile and Engineering Best Practices	Model-based testing, nanocycle execution, test driven development, continuous integration



Model Execution

The screenshot displays the IBM Rational Rhapsody Developer for C++ interface. The main window shows a statechart titled "Statechart of : FCM_Turns_uc - FCMBuilder_Turn.itsFCM_Turns_ucF...". The statechart includes states such as "AllStop", "ManagingTraffic", "RedRed", "WaitingForTurn", "GreenRed", "YellowRed", "TurnRed", "TurnGreen", "TurnYellow", "PTurnNoWait", "PTurnWait", "STurnRed", "STurnGreen", "STurnYellow", "STurnNoWait", and "STurnWait". Transitions are labeled with events and actions, such as "inPRIMARY_RED_TIM", "outPRIMARY_RED_TIM", "inSEC_GREEN_TIM", "outSEC_GREEN_TIM", etc.

On the right side, a sequence diagram titled "Sequence Diagram: Animated Animated FCM Turn Template_0*" is visible, showing a series of vertical lifelines for objects like "FCMBuilder...".

At the bottom left, a console window displays the following output:

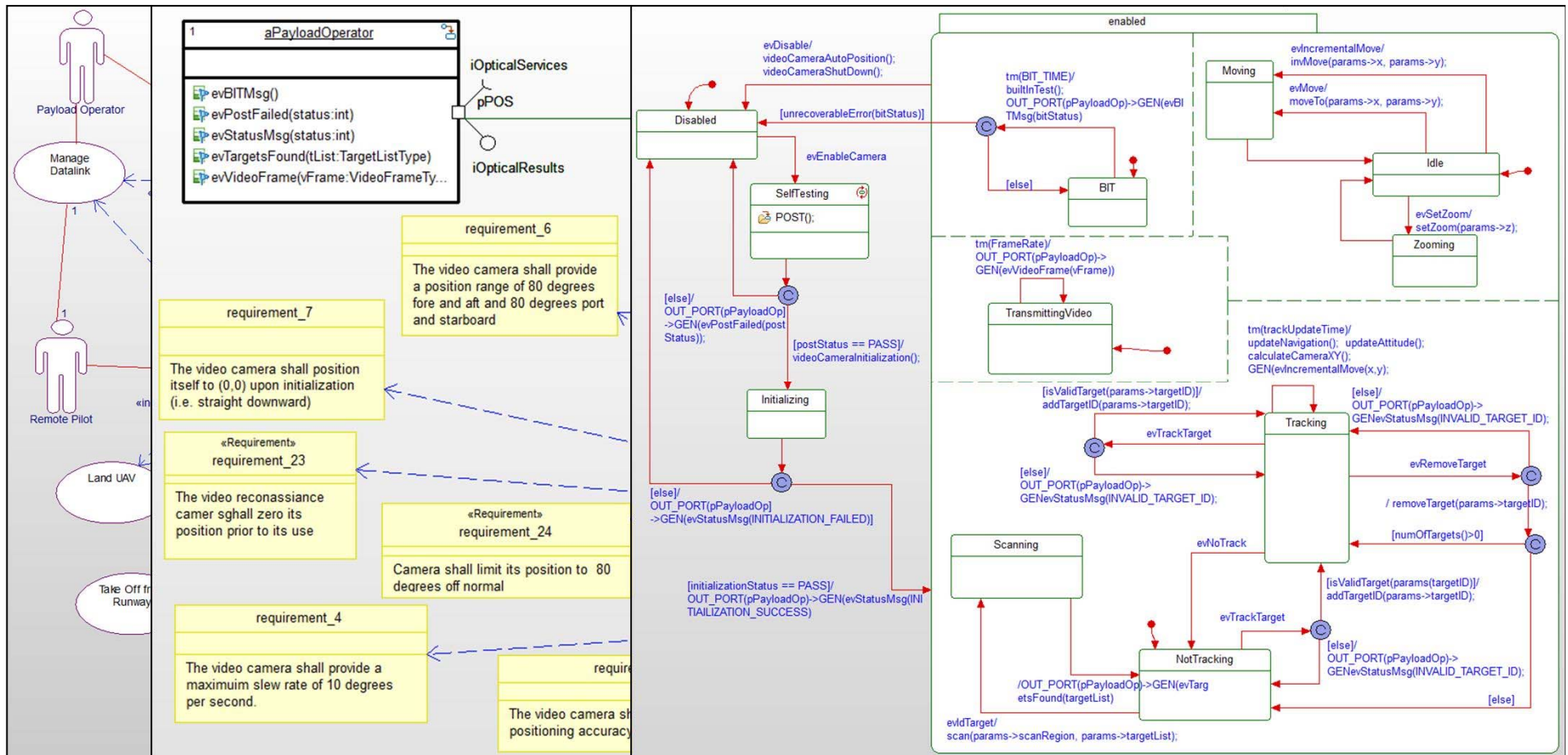
```

set ped lane id 0
Initialized Secondary Through
set ped lane id 2
Initialized Primary Through
set lane id 0
Initialized Primary Turn
set lane id 1
Initialized Secondary Through
set lane id 2
Initialized Secondary Turn
set lane id 3
Ped Primary Through=don't walk
Ped Secondary Through=don't walk
Primary Through=Red
Primary Turn=Red
Secondary Through=Red
Secondary Turn=Red
    
```

The bottom right corner of the application window shows the "Event Queue" and "Results" panels, along with the system date and time: "Tue, 12, Feb 2013 1:07 PM".

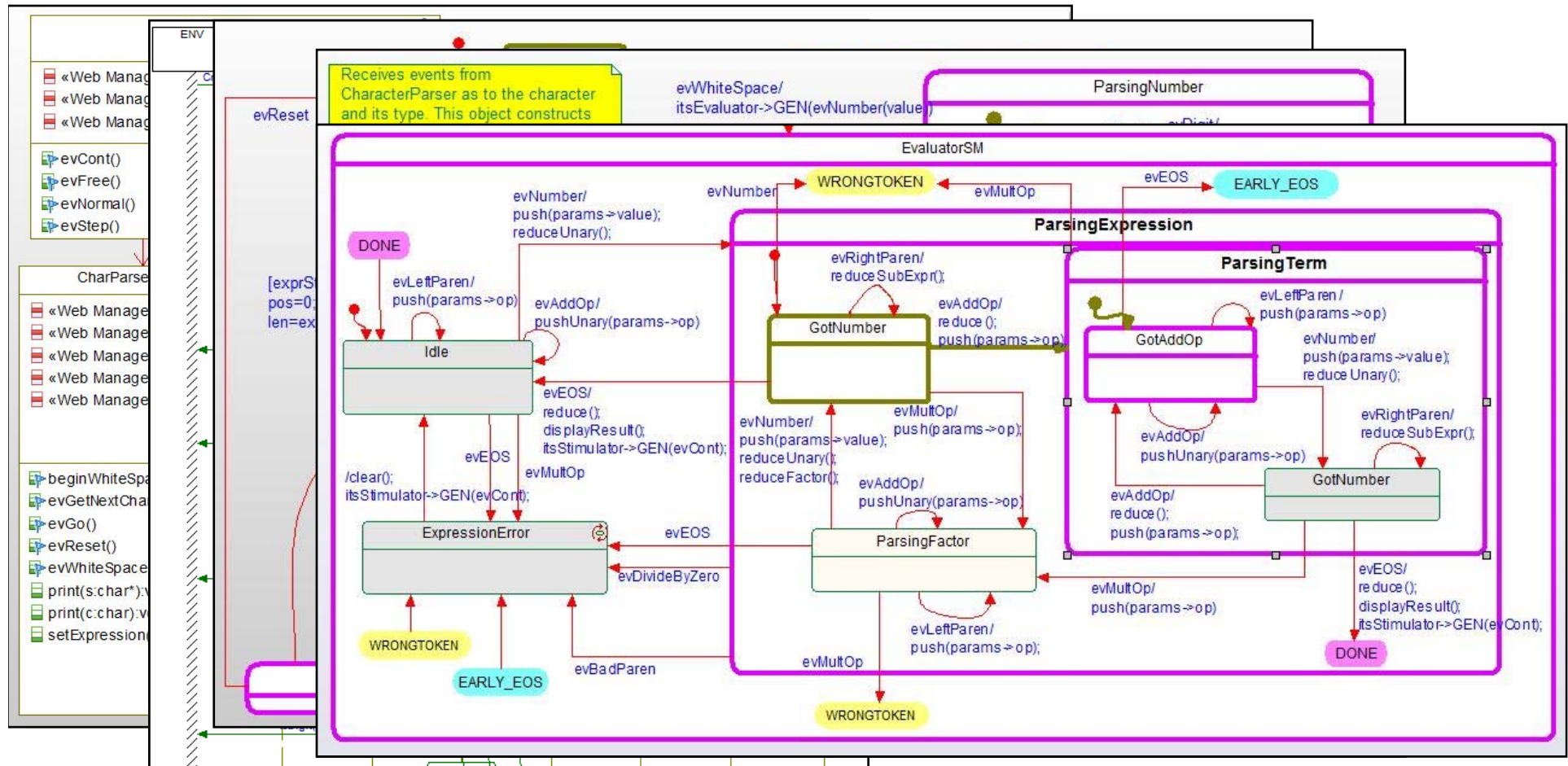
Model Execution supports *Model-Based Verification*

- Requirements Models
 - ▶ Requirements specify a systems input-output control and data transformations



Model Execution supports *Model-Based Verification*

- Design Models
 - ▶ Design specifies the (internal) structural elements and their behavior for implementation



Code Generation from Design Models

The screenshot displays the IBM Rational Rhapsody Developer for Ada environment. The interface is divided into several panes:

- Project Browser (Left):** Shows the project structure for 'Dishwasher', including components like 'Front_Panel', 'Real_Time', and 'text_io'.
- Object Model Diagram (Top Middle):** Shows a class hierarchy where 'Front_Panel' is associated with 'Dishwasher'.
- Statechart of: Dishwasher* (Bottom Middle):** Displays a state transition diagram for the Dishwasher component.
- Dishwasher.ads (Middle Right):** Contains Ada code defining types and procedures:


```

472 type OMAnim_Dishwasher_Set_Rinse_Time_t is
473 record
474 baseOper: RhpAnim.ARCOperation;
475 value : Integer;
476 end record;
477
478 Type OMAnim_Dishwasher_Set_Rinse_Time_vtbl_t
479 record
480 invoke: system.address ;
481 unserializeArguments: system.address;
482 end record;
483
484
485 procedure Dishwasher_Set_Rinse_Time_Unserial
486 me: in out OMAnim_Dishwasher_Set_Rinse_Tim
487 argValues: System.Address;
488 Position : System.Address;
489 pragma Export(C, Entity => Dishwasher_Set_Ri
490 External_Name=>"Dishwasher_Set_Rinse_Time_
491
492 function Dishwasher_Set_Rinse_Time_invoke (
493 me: OMAnim_Dishwasher_Set_Rinse_Time_t;
494 theInstance: System.Address) return System
495 pragma Export(C, Entity => Dishwasher_Set_Ri
496 External_Name =>"Dishwasher_Set_Rinse_Time
497
498 OMAnim_Dishwasher_Set_Rinse_Time: OMAnim_Dis
499
500 OMAnim_Dishwasher_Set_Rinse_Time_vtbl: OMAni
501 (Dishwasher_Set_Rinse_Time_invoke'address,
502 Dishwasher_Set_Rinse_Time_Unserialize_Argum
503
504 type OMAnim_Dishwasher_Set_Wash_Time_t is
505 record
506 baseOper: RhpAnim.ARCOperation;
507 value : Integer;
508 end record;
509
510 Type OMAnim_Dishwasher_Set_Wash_Time_vtbl_t
511
            
```
- Dishwasher.adb (Right):** Contains the implementation code for the procedures defined in the .ads file:


```

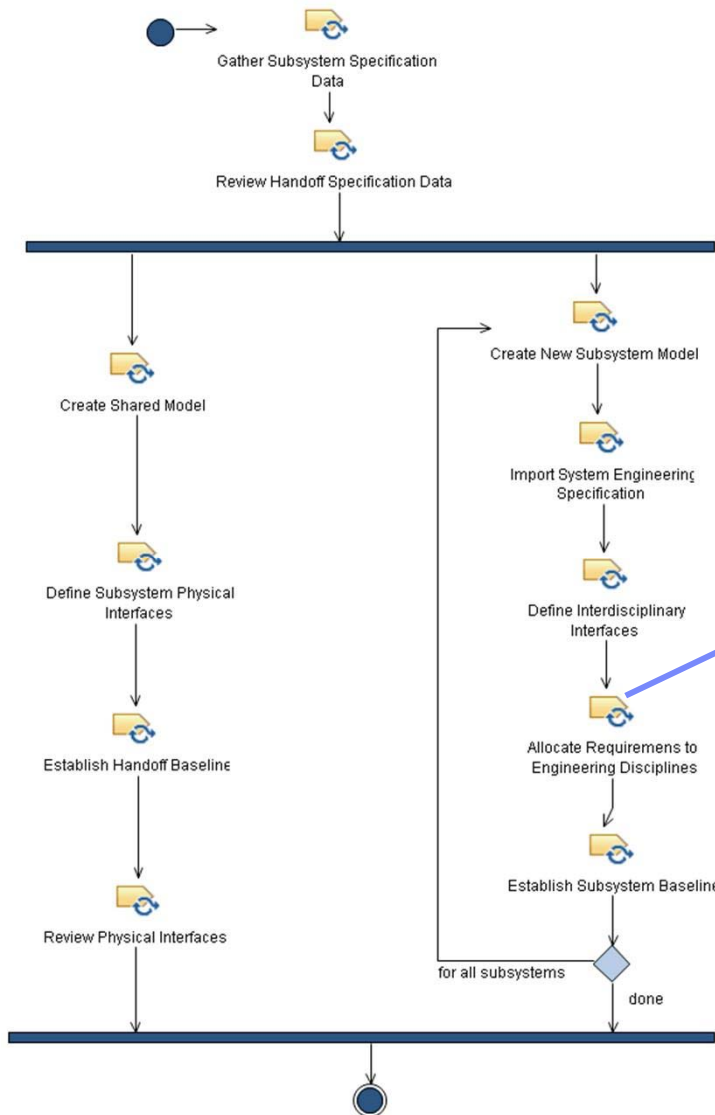
0437
0438 procedure Anim_Active_Serialize_States (
0439 Anim_State : RhpAnim.State_Acc_T) is
0440 begin
0441 RhpAnim.Add_State(Anim_State, Anim_Sta
0442 Anim_Active_Mode_Serialize_States(th
0443 Anim_Active_Running_Serialize_States
0444 Anim_Active_Service_Serialize_States
0445 end Anim_Active_Serialize_States;
0446
0447 procedure Anim_Active_Mode_Serialize_Sta
0448 Anim_State : RhpAnim.State_Acc_T) is
0449 begin
0450 RhpAnim.Add_State(Anim_State, Anim_Sta
0451 case this.Active_Mode_sub_state is
0452 when Active_Mode_Intense =>
0453 Anim_Active_Mode_Intense_Serialize
0454 when Active_Mode_Quick =>
0455 Anim_Active_Mode_Quick_Serialize_S
0456 when others =>
0457 null;
0458 end case;
0459 end Anim_Active_Mode_Serialize_States;
0460
0461 procedure Anim_Active_Mode_Intense_Seria
0462 Anim_State : RhpAnim.State_Acc_T) is
0463 begin
0464 RhpAnim.Add_State(Anim_State, Anim_Sta
0465 end Anim_Active_Mode_Intense_Serialize_S
0466
0467 procedure Anim_Active_Mode_Quick_Seriali
0468 Anim_State : RhpAnim.State_Acc_T) is
0469 begin
0470 RhpAnim.Add_State(Anim_State, Anim_Sta
0471 end Anim_Active_Mode_Quick_Serialize_Sta
0472
0473 procedure Anim_Active_Running_Serialize_
0474 Anim_State : RhpAnim.State_Acc_T) is
0475 begin
0476 RhpAnim.Add_State(Anim_State, Anim_Sta
0477 case this.Active_Running_sub_state is
            
```

Specifications & Models handed off to software from SE

- Any system specification or design model handed off to software should contain
 - ▶ Requirements from which model was developed
 - ▶ Model configuration items (CIs) (files or data representing the model)
 - ▶ Modeling standards describing the modeling techniques
 - ▶ Model element libraries
 - ▶ Model and system interfaces description
 - ▶ Configuration index of model CIs
 - ▶ Modeling development environment and user's manuals
 - ▶ Any data from V&V activities performed as system level the may be used to satisfy verification objectives



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

Expand All Steps Collapse All Steps

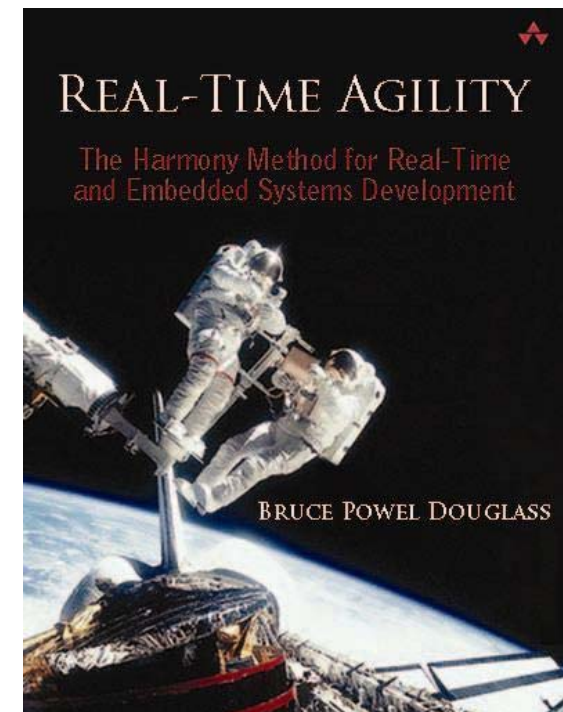
- Review subsystem requirements
- Allocate single-discipline requirements
- Decompose multi-discipline requirements
- Allocate derived requirements
- Update traceability record
- Review allocations

Back to top

Agile for Embedded Real-Time Systems

- Embedded is *different* than IT
 - ▶ More constrained
 - ▶ Often safety-critical
 - ▶ HW/SW co-design
 - ▶ Handed off to manufacturing not end users
 - ▶ More difficult to test
 - ▶ Far more difficult to update in the field
- Harmony process applied agile methods to embedded
 - ▶ Iteration-centric model-based development
 - ▶ Includes practices for
 - Test Driven Development
 - High-fidelity modeling
 - Continuous integration
 - Dynamic planning
 - Quality assurance
 - Continuous safety/reliability/security assessment

Agile



Rational. Method Composer

Search this Site:

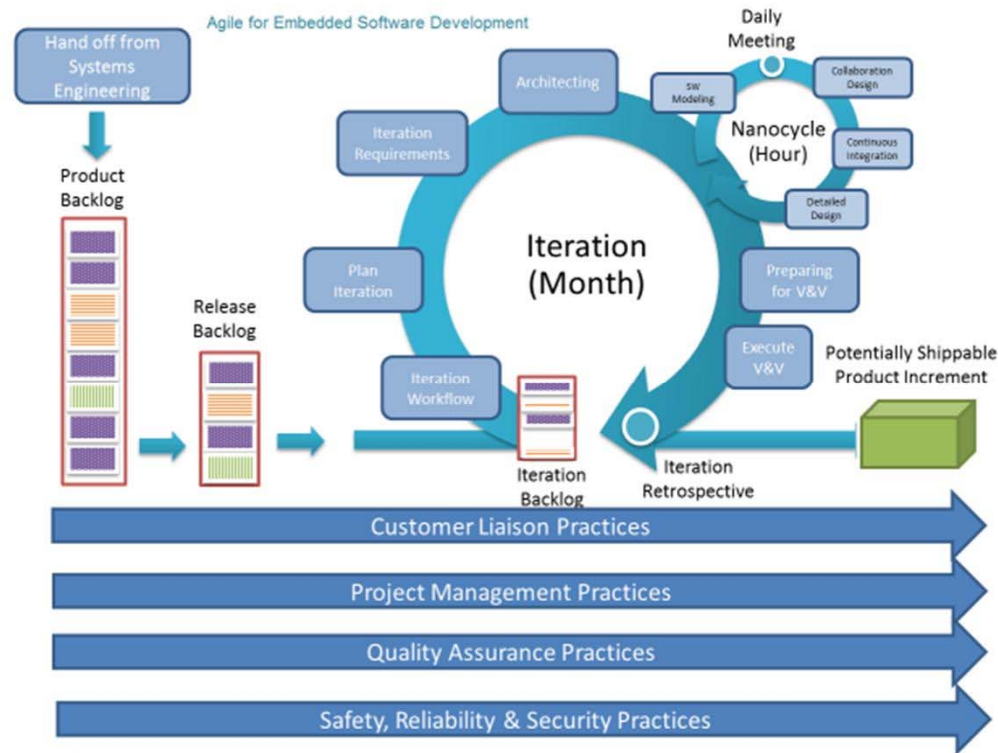
- Agile for ESW
 - Welcome to Harmony/Agile for Embedded Software Development
 - Getting Started
 - Delivery Processes
 - Harmony/Agile for Embedded Software Development
 - Handoff for Downstream Engineering
 - Define and Deploy the Development Environment
 - Develop Initial Requirements
 - Development Iteration
 - Configuration Management
 - Control Project
 - Manage Change
 - QA Audit
 - Pre-Iteration Planning
 - Development Iteration
 - Use Case / User Story Analysis
 - Plan Iteration
 - High-Fidelity Modeling
 - Architectural Design - RT
 - Collaboration Design - RT
 - Detailed Design - RT
 - Continuous Integration
 - Prepare for Verification and Validation
 - Verification and Validation
 - Guidance
 - Supporting Workflows
 - ESW Agile Practices
 - Tools

Welcome to Harmony/Agile for Embedded Software Development

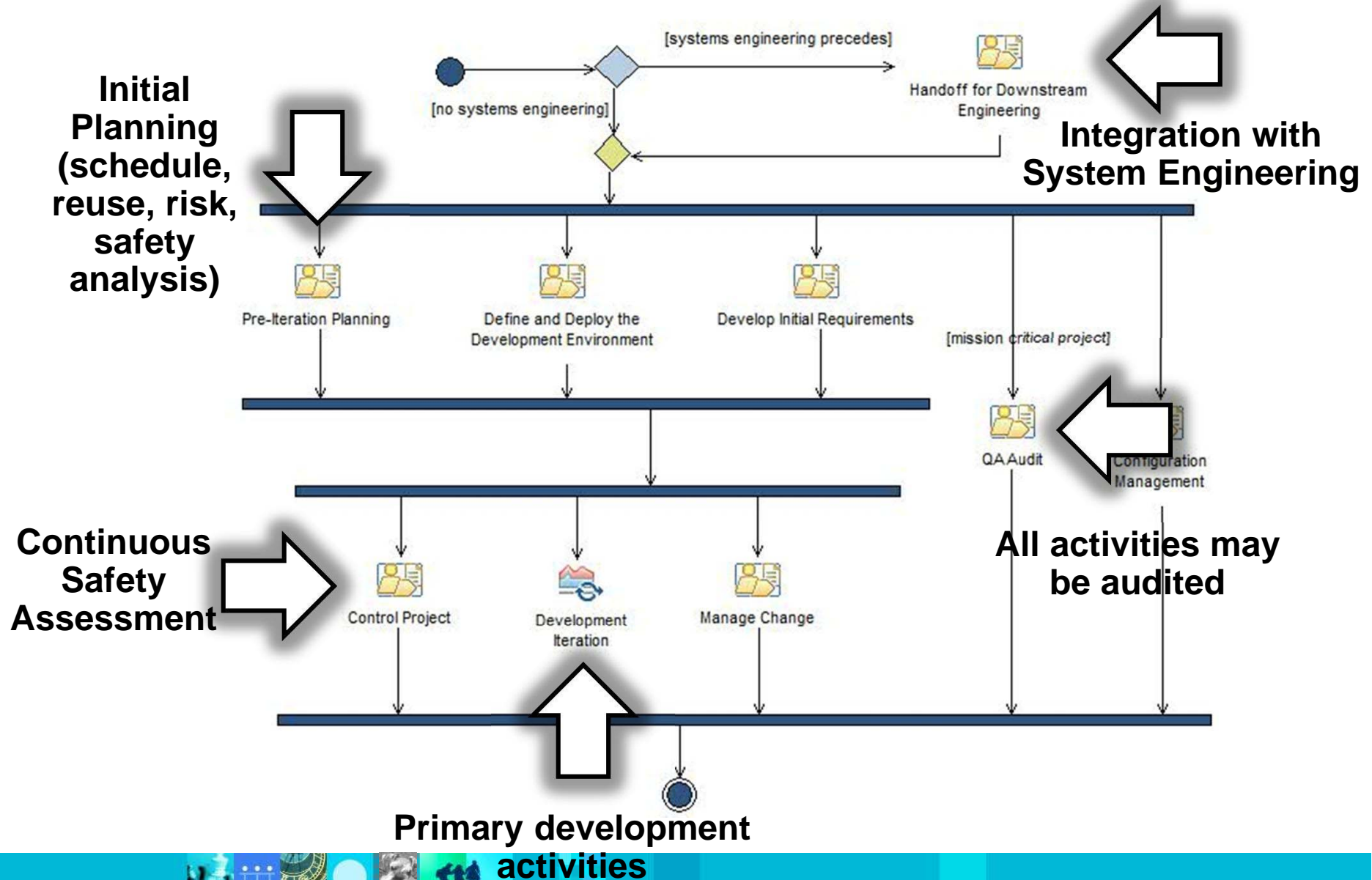
Welcome to Harmony/Agile for Embedded Software Development



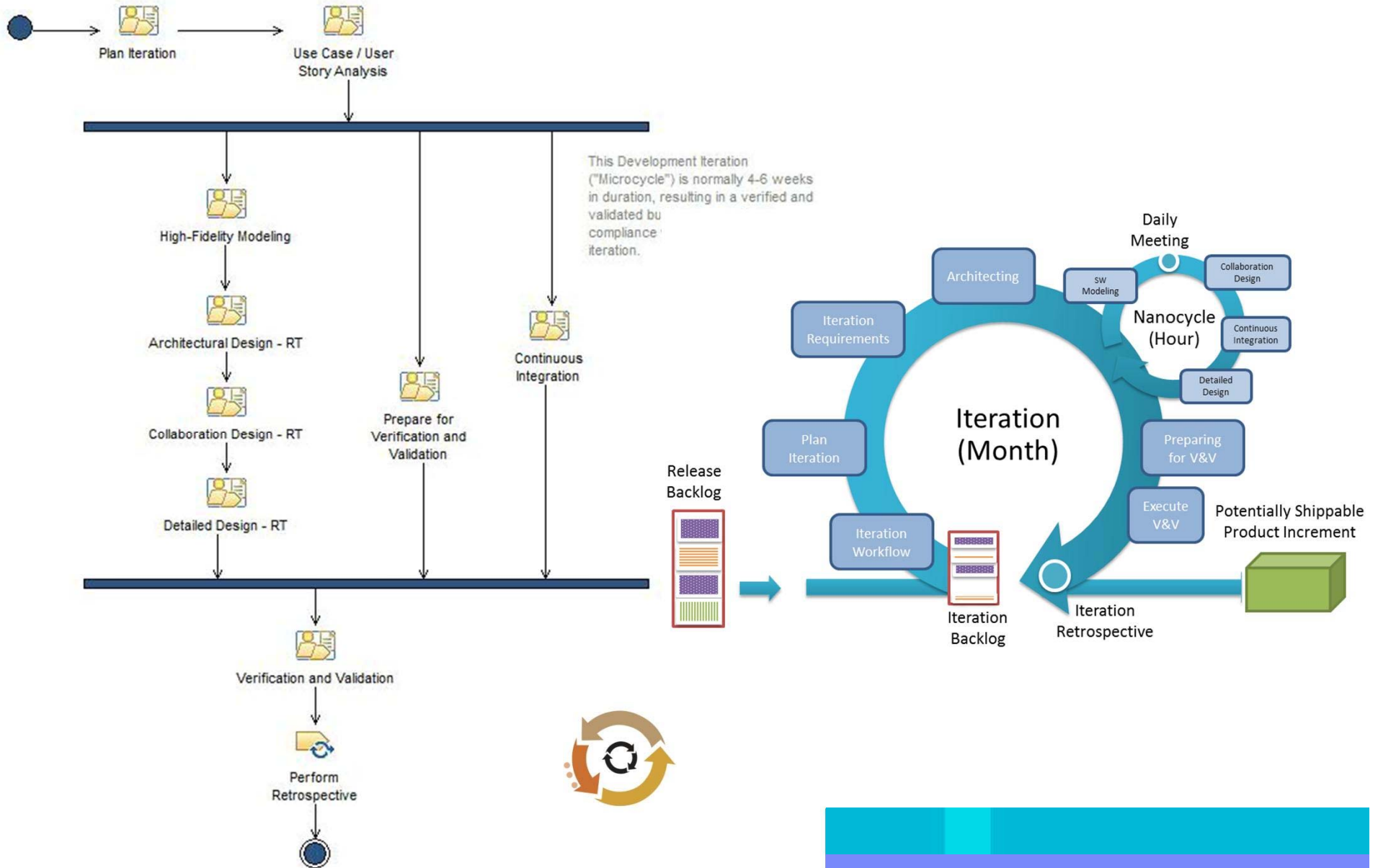
Main Description



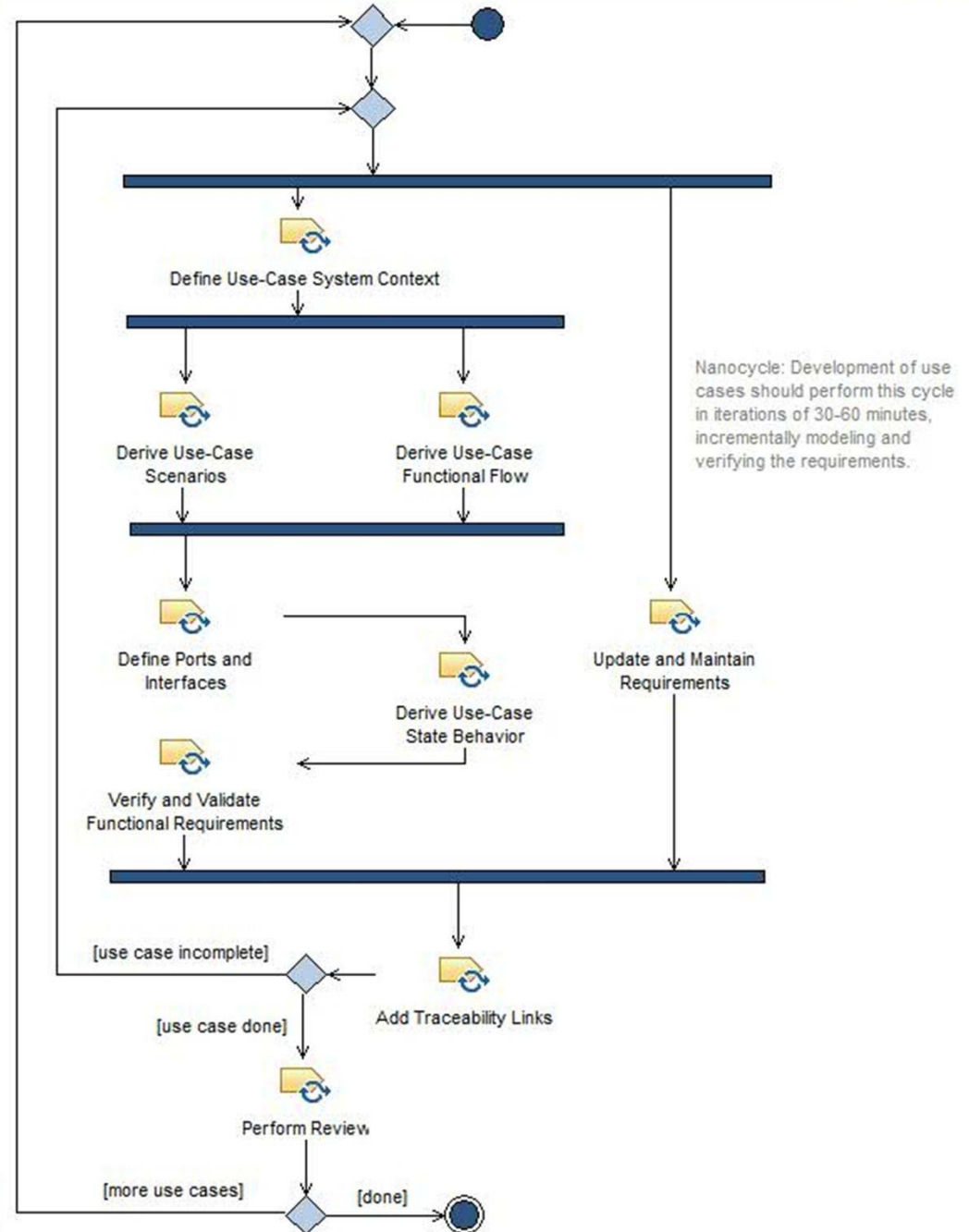
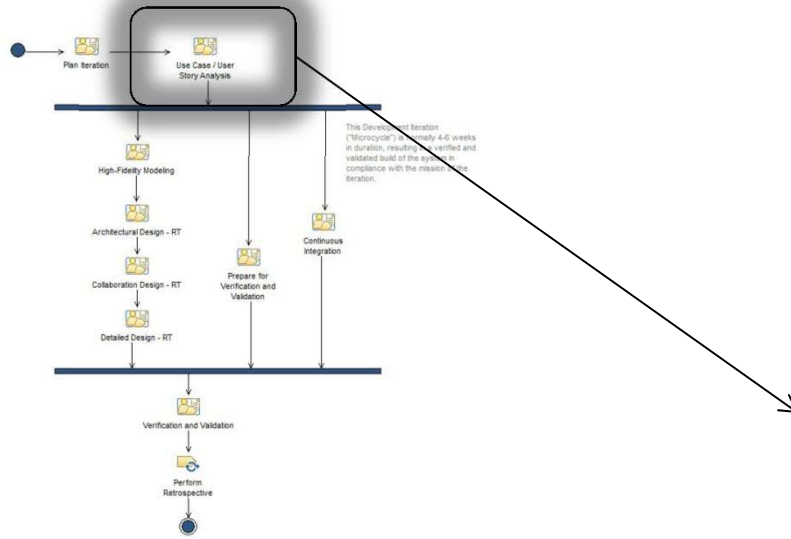
Harmony Agile Overview



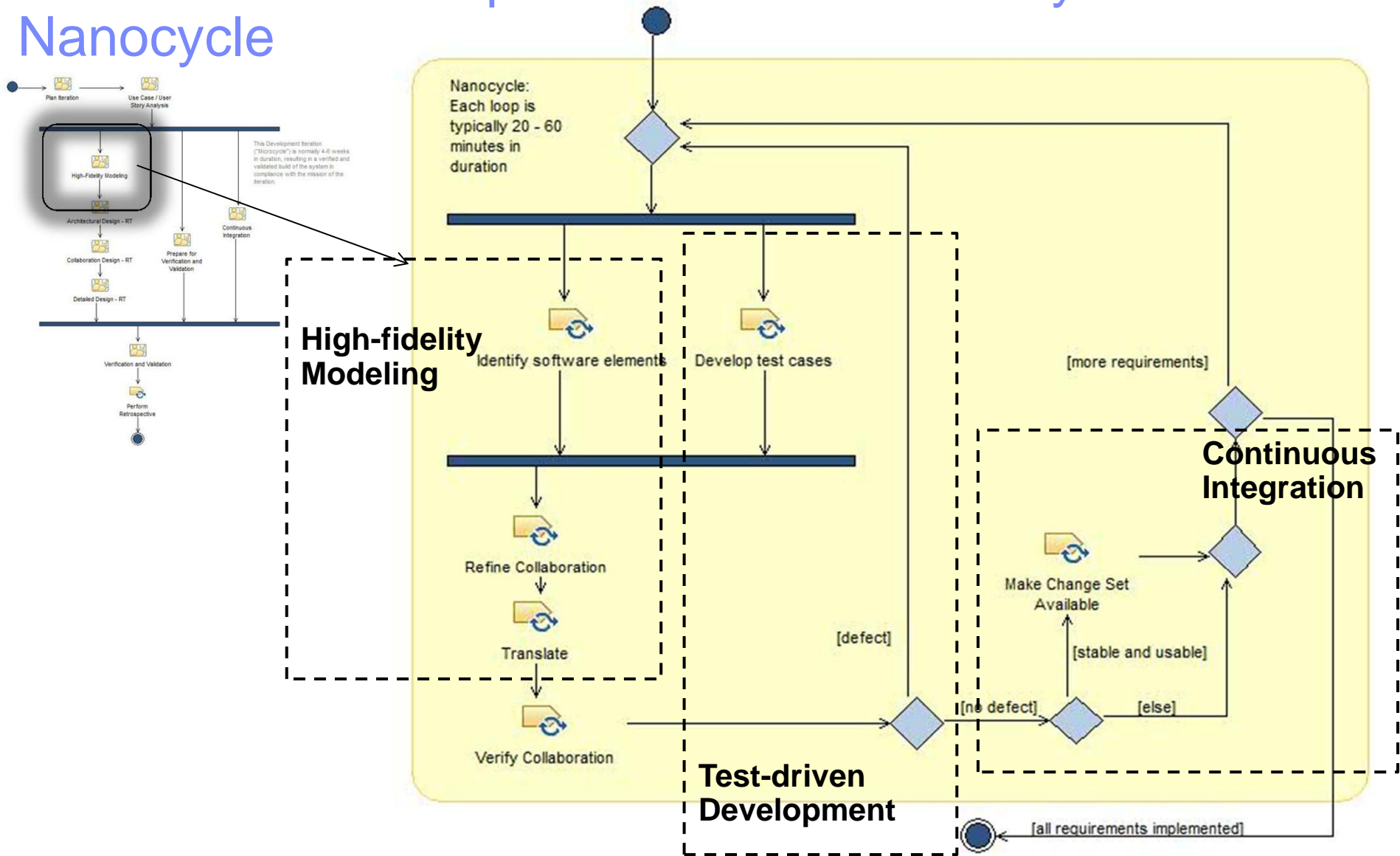
Incremental Development with Harmony



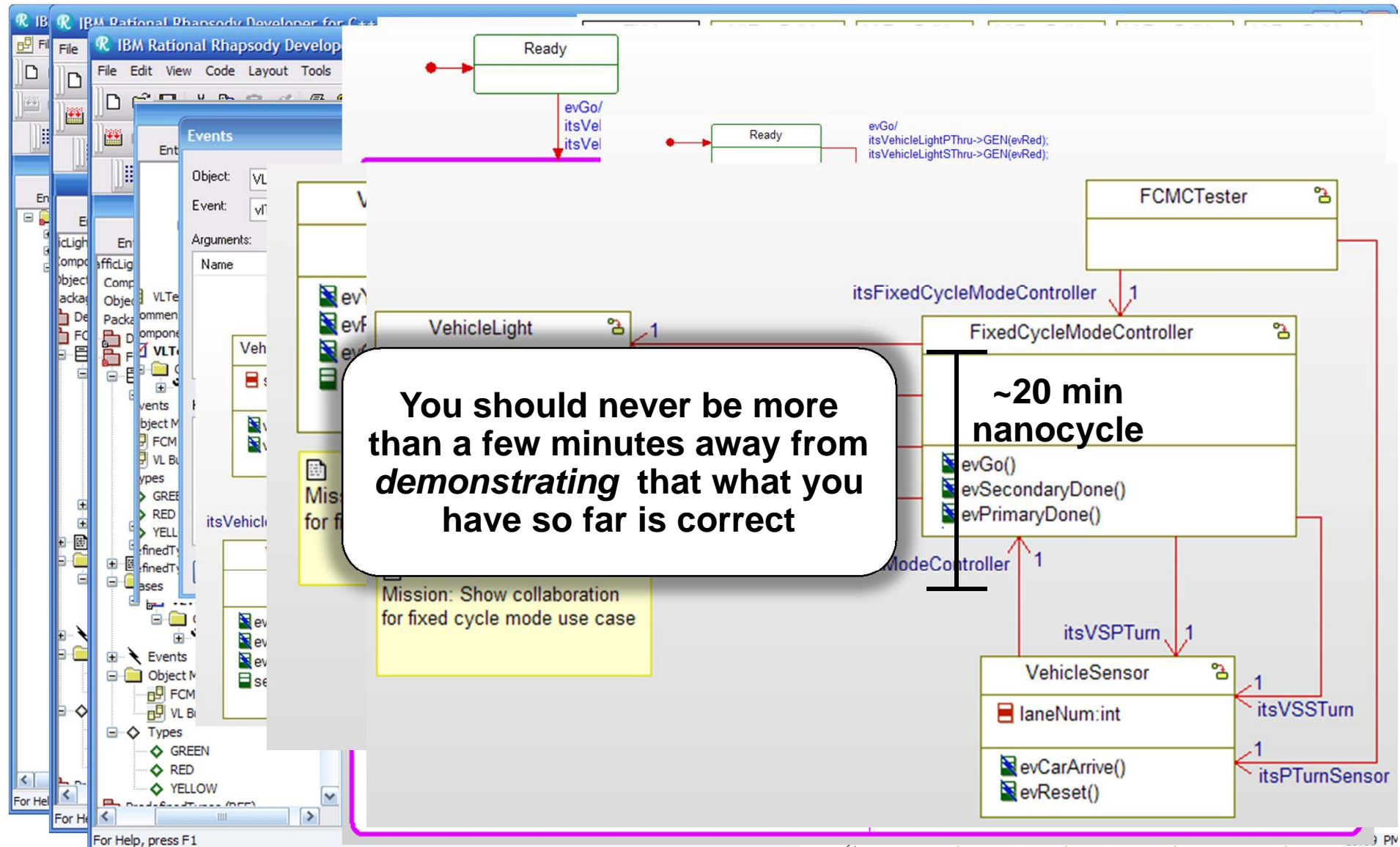
Incremental Use Case Analysis



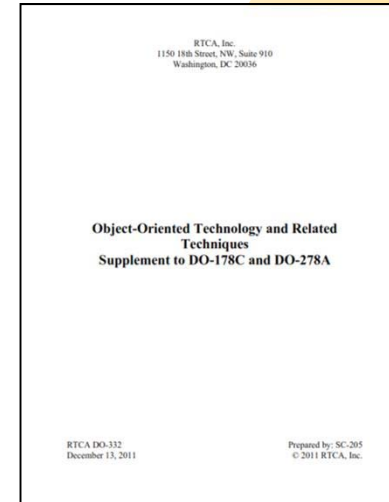
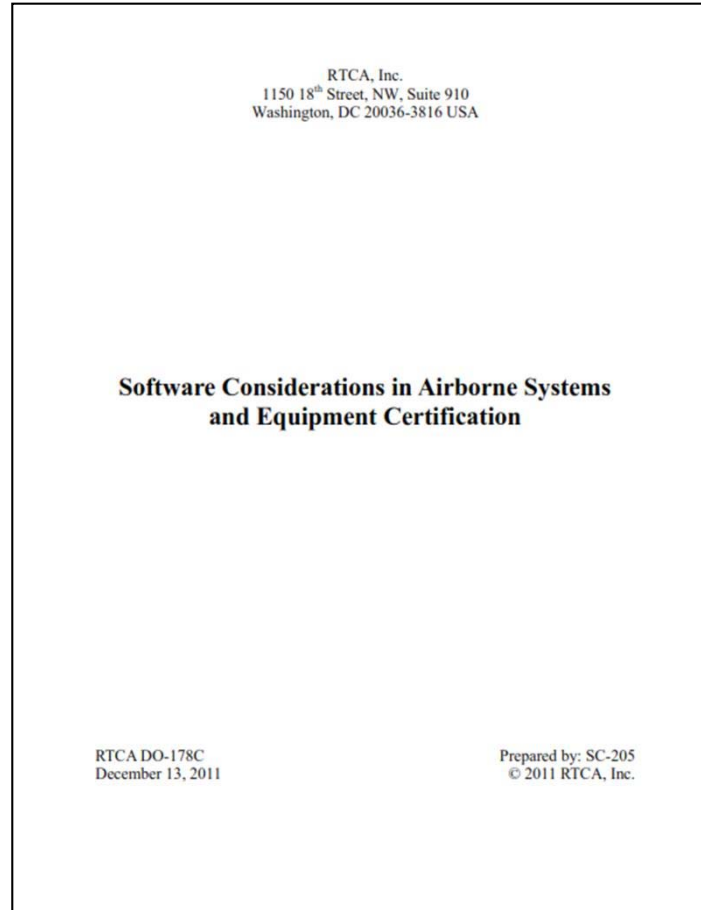
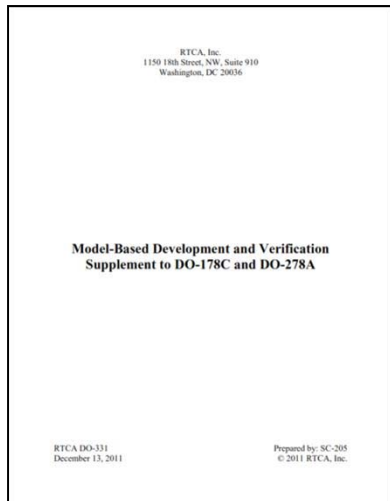
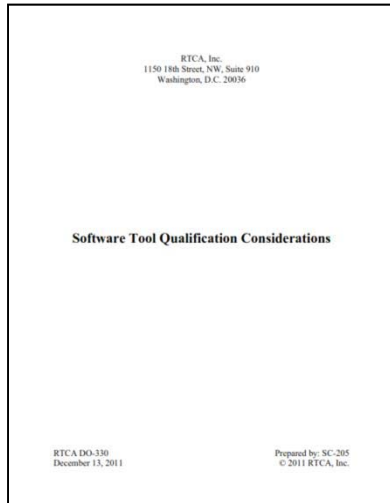
Test Driven Development and the Harmony Nanocycle



Model-Based TDD in Action



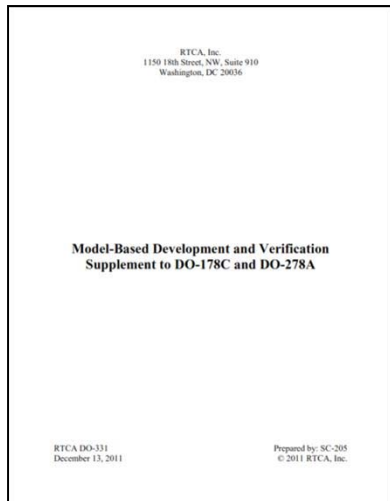
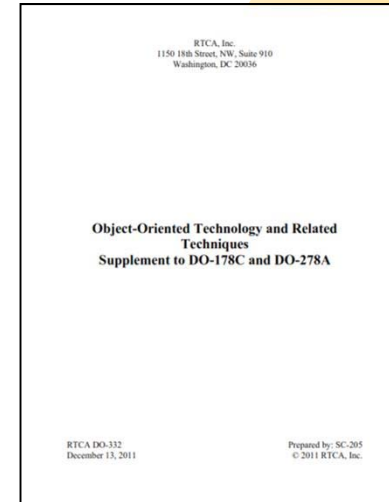
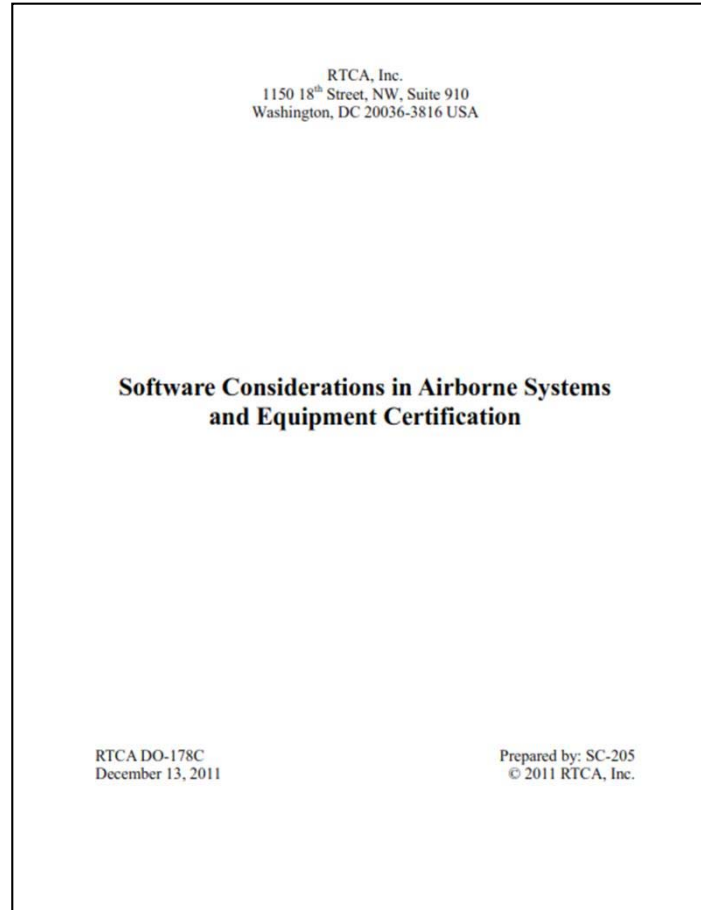
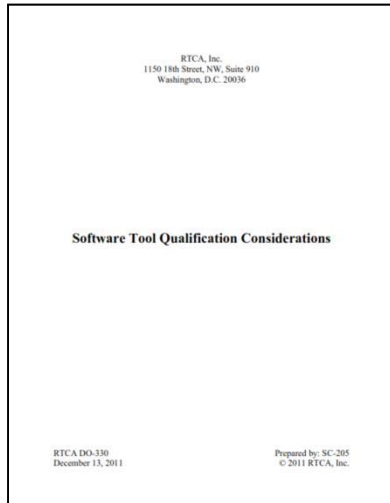
DO-331 Model Based Design Supplement to DO-178C



DO-178



DO-331 Model Based Design Supplement to DO-178C



DO-178



DO-331

- Clarifies the use of modeling in DO-178 projects
- 83 objectives total
- 12 entirely new objectives
- Identifies
 - ▶ Specification Models
 - ▶ Design Models
 - ▶ Need for identification of normative and non-normative elements
 - ▶ Guidance for use of models in DO-1788 projects

Table MB.C-1 Software Planning Process

Objective	Activity	Applicability by Assurance Level	Output		Control Category by Assurance Level									
					AL 1	AL 2	AL 3	AL 4	AL 5					
Description	Ref	Ref	AL 1	AL 2	AL 3	AL 4	AL 5	Data Item	Ref	AL 1	AL 2	AL 3	AL 4	AL 5
1 The activities of the software life cycle processes are defined.	MB.4.1.a	MB.4.2.a						PSAA	MB.11.1	①	①	①	①	①
		MB.4.2.c						SDP	MB.11.2	①	①	②	②	②
		MB.4.2.d						SVP	MB.11.3	①	①	②	②	②
		MB.4.2.g						SCM Plan	MB.11.4	①	①	②	②	②
		MB.4.2.i						SQA Plan	11.5	①	①	②	②	②
2 The software life cycle(s), including the inter-relationships between the processes, their sequencing, feedback mechanisms, and transition criteria, is defined.	MB.4.1.b	MB.4.2.j						PSAA	MB.11.1	①	①	①	①	
		MB.4.3.b						SDP	MB.11.2	①	①	②	②	
								SVP	MB.11.3	①	①	②	②	
								SCM Plan	MB.11.4	①	①	②	②	
								SQA Plan	11.5	①	①	②	②	
3 Software life cycle environment is selected and defined.	MB.4.1.c	4.4.1						PSAA	MB.11.1	①	①	①	①	
		MB.4.4.2.a						SDP	MB.11.2	①	①	②	②	
		MB.4.4.2.b						SVP	MB.11.3	①	①	②	②	
		MB.4.4.2.c						SCM Plan	MB.11.4	①	①	②	②	
		MB.4.4.3						SQA Plan	11.5	①	①	②	②	
4 Additional considerations are addressed.	MB.4.1.d	MB.4.2.f						PSAA	MB.11.1	①	①	①	①	①
		MB.4.2.h						SDP	MB.11.2	①	①	②	②	②
		MB.4.2.i						SVP	MB.11.3	①	①	②	②	②
		MB.4.2.j						SCM Plan	MB.11.4	①	①	②	②	②
		MB.4.2.						SQA Plan	11.5	①	①	②	②	②
5 Software development standards are defined.	MB.4.1.e	MB.4.2.b						SW Requirements Standards	11.6	①	①	②	②	
		MB.4.2.g						SW Design Standards	11.7	①	①	②	②	
		MB.4.5						SW Code Standards	11.8	①	①	②	②	
								SW Model Standards	MB.11.23	①	①	②	②	
6 Software plans comply with this document.	MB.4.1.f	MB.4.3.a						Software Verification Results	MB.11.14	②	②	②	②	
		MB.4.6												
7 Development and revision of software plans are coordinated.	MB.4.1.g	MB.4.2.g						Software Verification Results	MB.11.14	②	②	②	②	
		MB.4.6												



Model Based Development and Verification

- Examples of some current industry practices using Model Based Development

Table MB.1-1 Model Usage Examples

Process that generates the life-cycle data	MB Example 1	MB Example 2	MB Example 3	MB Example 4 (See Note 1)	MB Example 5 (See Note 1)
System Requirement and System Design Processes	Requirements allocated to software	Requirements from which the Model is developed	Requirements from which the Model is developed	Requirements from which the Model is developed	Requirements from which the Model is developed
					Design Model
Software Requirement and Software Design Processes	Requirements from which the Model is developed	Specification Model (See Note 2)	Specification Model	Design Model	
	Design Model	Design Model	Textual description (See Note 3)		
Software Coding Process	Source Code	Source Code	Source Code	Source Code	Source Code

Source: RTCA D0-331



Bruce's recommendation



Model Based Development and Verification

Model Simulation

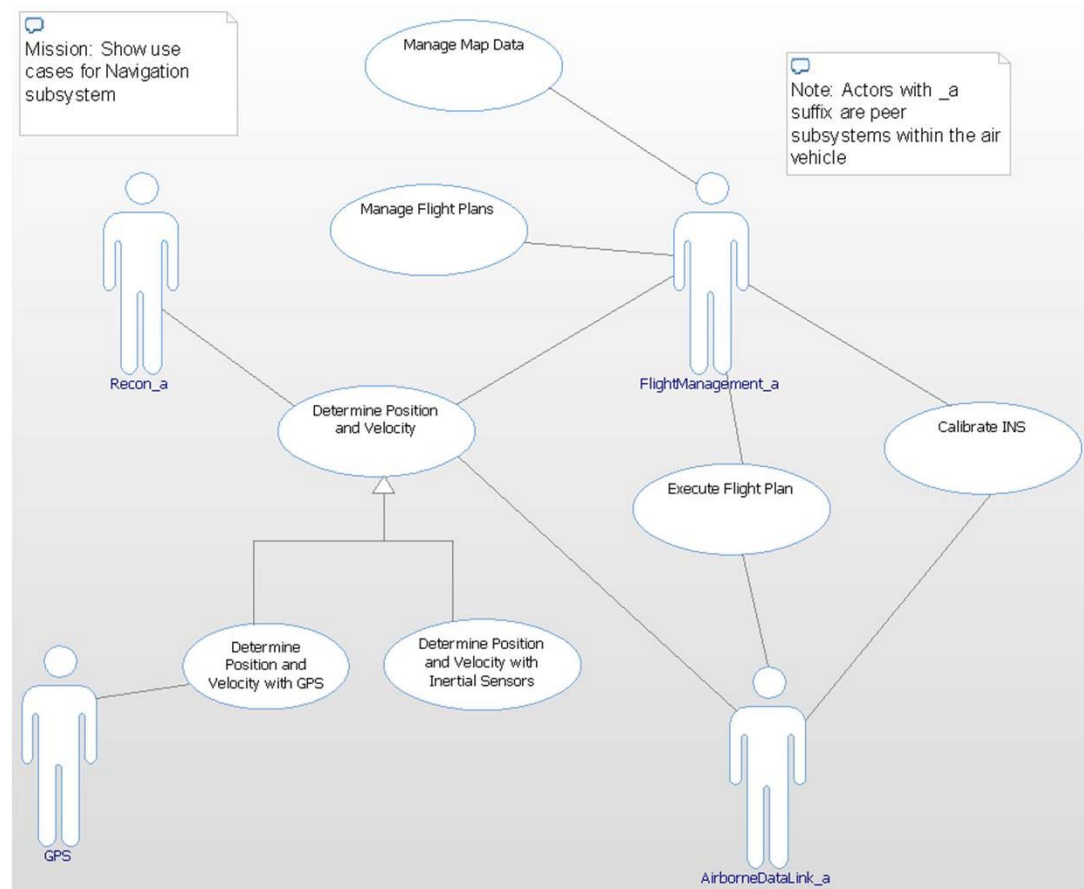
- For Specification Models or Design Models, simulation may be used in combination with reviews and analysis of requirements and architecture to satisfy some objectives of sections MB.6.3.1, MB.6.3.2, and MB.6.3.3.
- Goal is to provide repeatable evidence that the model complies with its requirements
- Cannot help meet objectives of compatibility with target computer, traceability, conformance to standards, or partitioning integrity.
- Can help satisfy:

Objectives (Compliance to:)	
System Requirements for Specification Models	MB.6.3.1.a
SW HLR for Design Models	MB.6.3.2.a, MB.6.3.2.a
HLR/LLR Accuracy and consistency	MB.6.3.1.b, MB.6.3.2.b
HLR/LLR Verifiability	MB.6.3.1.d and MB.6.3.2.d
Algorithm aspects	MB.6.3.1.g, MB.6.3.2.g
Consistency & Verifiability of SW Architecture	(may provide) MB.6.3.3.b, MB.6.3.3.d



DO-331 Specification Models

- Contain high-level requirements (HLR)
- Contains no*
 - ▶ Design
 - ▶ Low-level requirements
 - ▶ Detailed data flow
- Typically supported by multiple viewpoints
 - ▶ Use case diagram
 - ▶ Sequence diagram
 - ▶ Activity diagram
 - ▶ State machine
 - ▶ Links to text-based requirements
 - ▶ Simulation / execution data
- May *not* be used to generate code

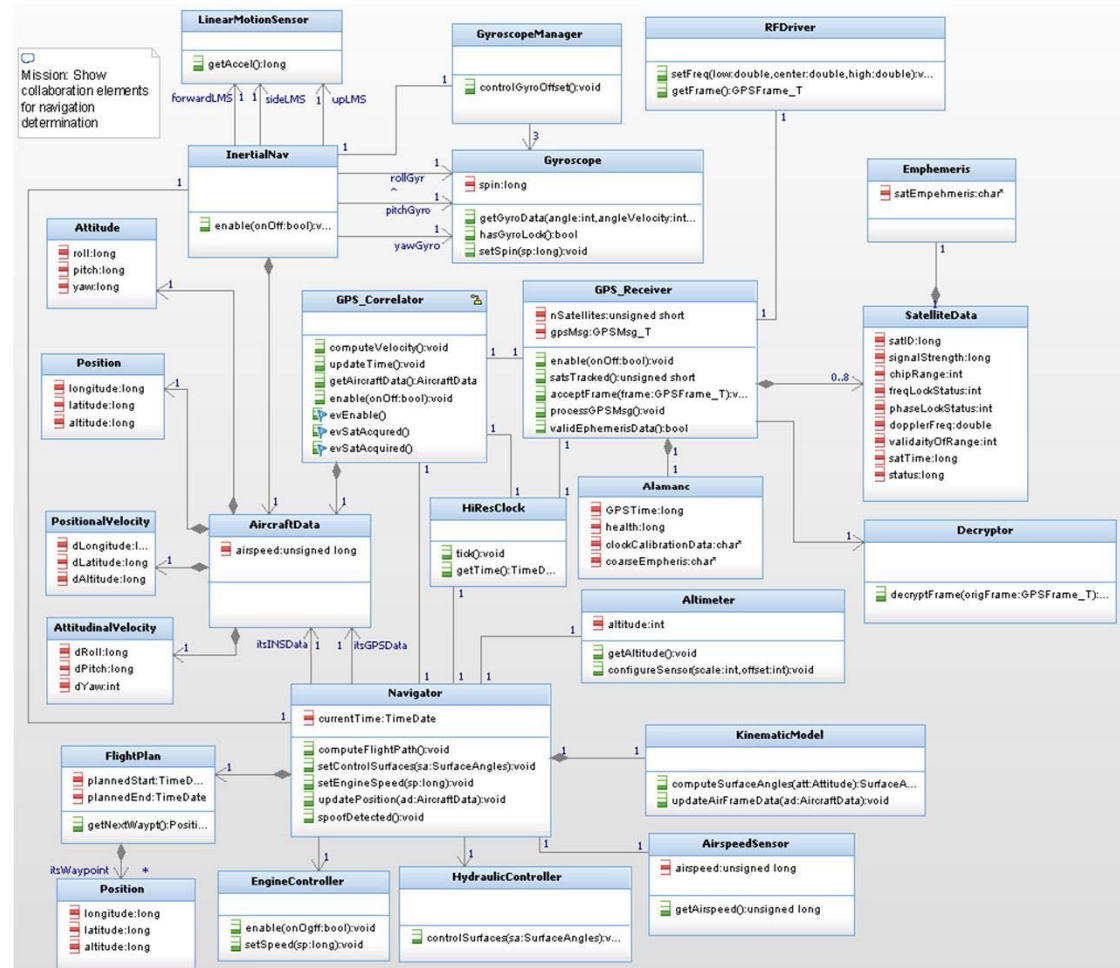


*(except to justify design constraints in HLR)



DO-331 Design Models

- May contain
 - ▶ Design
 - ▶ Low-Level Requirements (LLRs)
 - ▶ Architecture
 - ▶ Data structures
 - ▶ Detailed data flow
 - ▶ Detailed control flow
- Typically supported with multiple viewpoints
 - ▶ Class / Object / structure diagrams
 - ▶ Sequence diagrams
 - ▶ Activity diagrams
 - ▶ State diagrams
 - ▶ Source code
 - ▶ Verification data
- *May be used to generate code*



SW Modeling Standards

- DO-178 generally requires project standards and checklists for crucial work products
 - ▶ *Standards* lay out the (meta)requirements and organizational principles for work products
 - ▶ *Checklists* are used by SQA personnel to review the syntax, format, scope and completeness of the work product (semantics are dealt with by other reviewer roles and by verification)

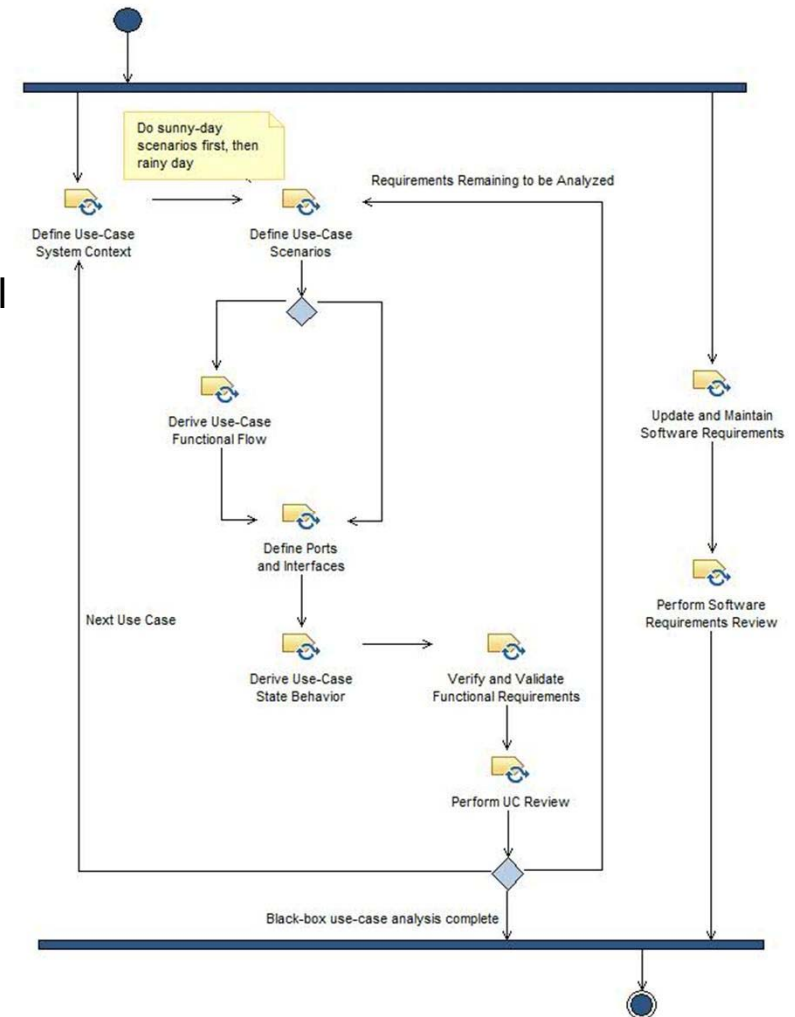
- SW Modeling standards specify
 - ▶ Modeling techniques
 - ▶ Methods for modeling
 - ▶ Modeling languages (e.g. UML) incl.
 - reference to language standard,
 - style and complexity guidelines,
 - constraints,
 - means to trace to requirements,
 - means to identify any non-normative elements,
 - rational for the suitability of the technique for the information to be expressed

Harmony Model Review Checklist	
Work Product(s) under Review	
Product / System name	
ProductName ID	
Configuration ID	
Product type	
Product(s) Owner	
Reviewer(s)	
Review date	
Standard(s) applied	
<p>Harmony-SW Modeling Standard Rev 1.3</p> <p>Harmony-SW Modeling Standards for use with MDA, UML, and Rhapsody</p> <p>Prepared by Dr. Bruce Powell Douglas, Ph.D. Chief Evangelist IBM Rational</p>	
<p>Review Checklist</p> <p>Page 1 of 52 © IBM 2012 Bruce Powell Douglas, Ph.D.</p>	



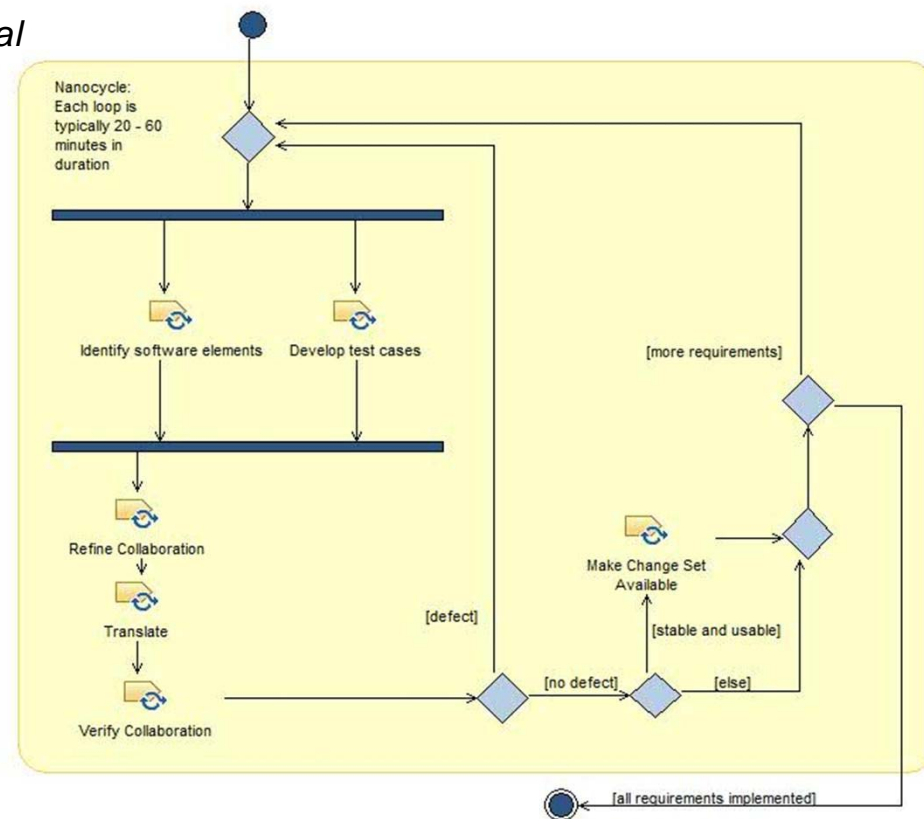
Harmony: Specification Models

- Identify use cases
 - ▶ Traceability links to requirements
 - ▶ Typically 8-20 pages of requirements
- Incrementally refine use case definition
 - ▶ Define the operational scenarios and operational contracts with sequence diagrams
 - ▶ Identify flow with activity diagrams
 - ▶ Specify normative semantics with state diagram
 - ▶ Verify with simulation / execution
 - ▶ Validate with customer at stable points
 - ▶ Repeat until all requirements represented



Harmony: Design Models

- Development proceeds incrementally with the basic premise of “make small incremental additions, verify, repeat”
- High Fidelity Modeling
 - ▶ Identify the software elements required for *functional correctness*
 - ▶ Produce a *functionally correct code base* that is verified at unit- and integration - levels
- Design optimizes at three levels of abstraction
 - ▶ Architectural – optimize the overall system with 5 key views
 - *Subsystem and Component View*
 - *Safety and Reliability View*
 - *Distribution View*
 - *Concurrency and Resource View*
 - *Deployment View*
 - ▶ Collaboration – use-case level scope
 - ▶ Detailed – individual software element scope



Wrapping up: State of the Art

