

# Agile Model-Based Systems Engineering (aMBSE)

**Bruce Powel Douglass, Ph.D.**

Chief Evangelist, Global Technology Ambassador  
IBM Internet of Things

[Bruce.Douglass@us.ibm.com](mailto:Bruce.Douglass@us.ibm.com)

Twitter: @IronmanBruce

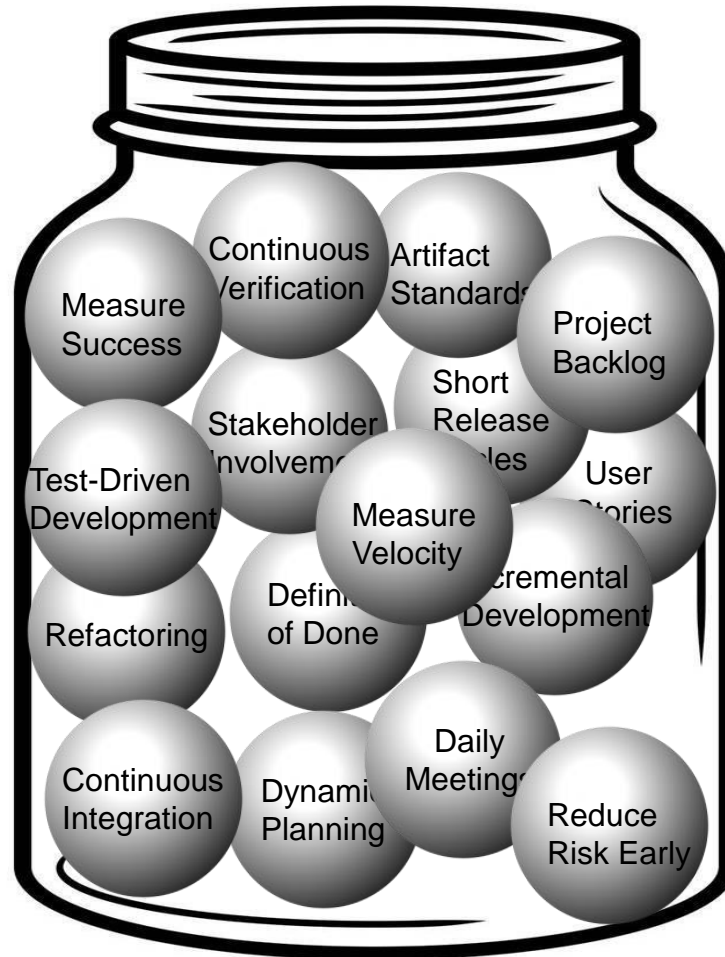
IBM: [www-01.ibm.com/software/rational/leadership/thought/BruceDouglass.html](http://www-01.ibm.com/software/rational/leadership/thought/BruceDouglass.html)



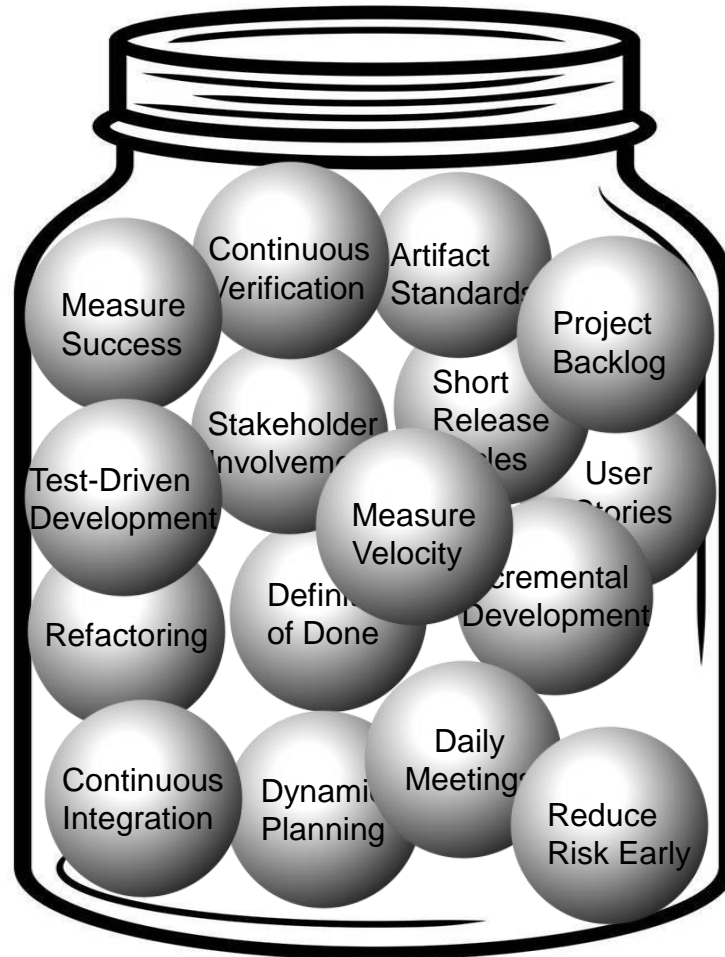
*“Dance like nobody is watching,  
Sing like you’re alone in the shower,  
Engineer like you’re a passenger  
hurtling through space in a speeding  
tube of death that you designed.”*

*Law of Douglass # 135*

# Agile Practices

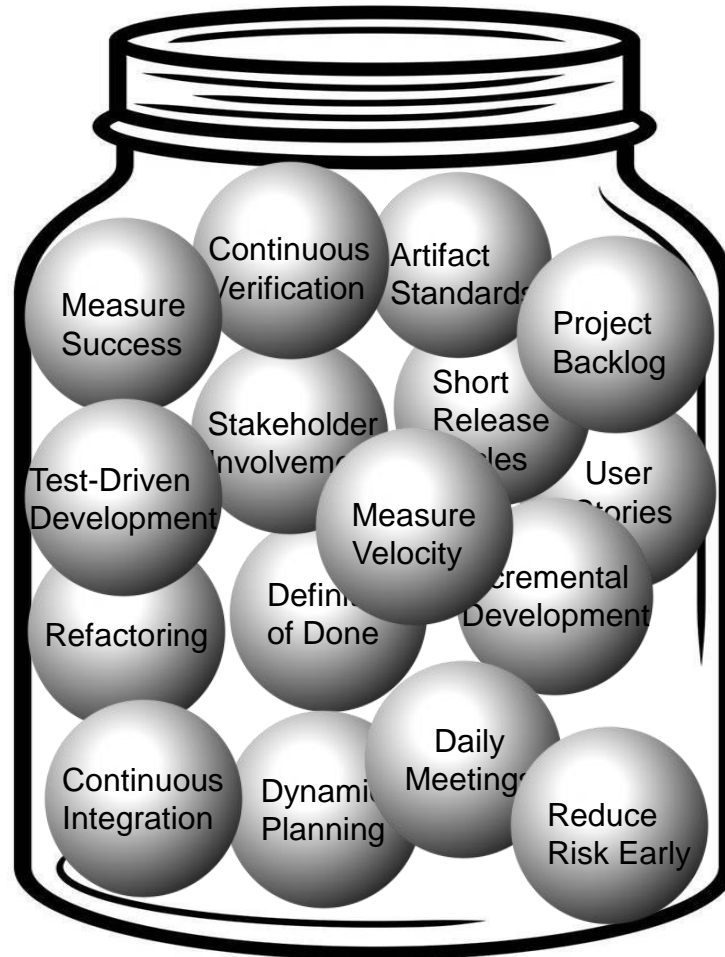


# Agile Practices



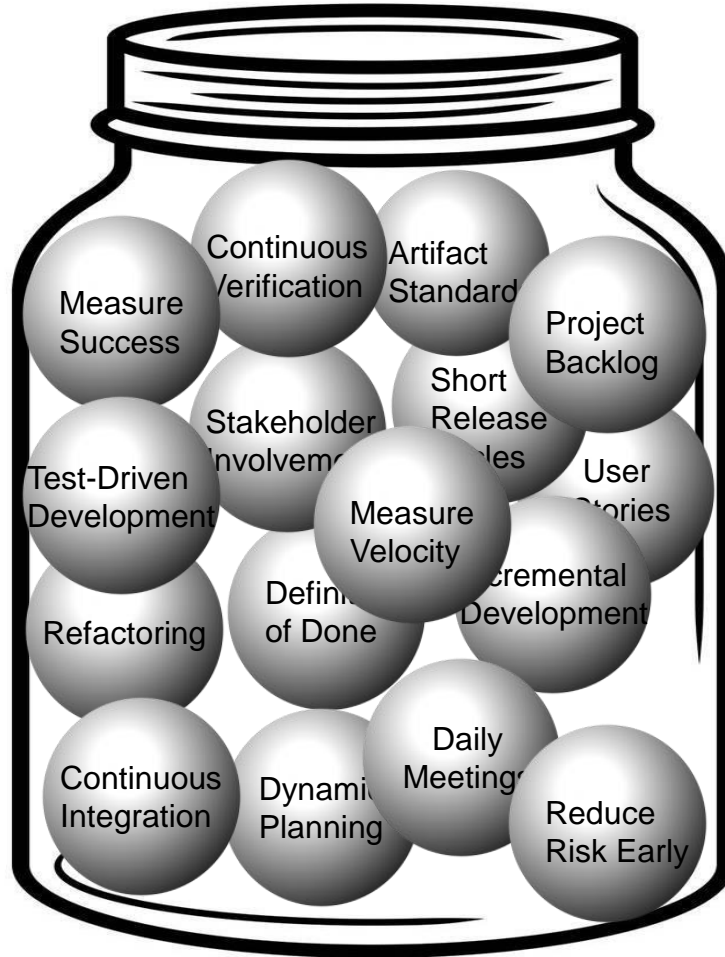
Create and apply test cases as you develop the product, not after the fact

# Agile Practices



Continuously verify  
the correctness of  
your engineering  
data

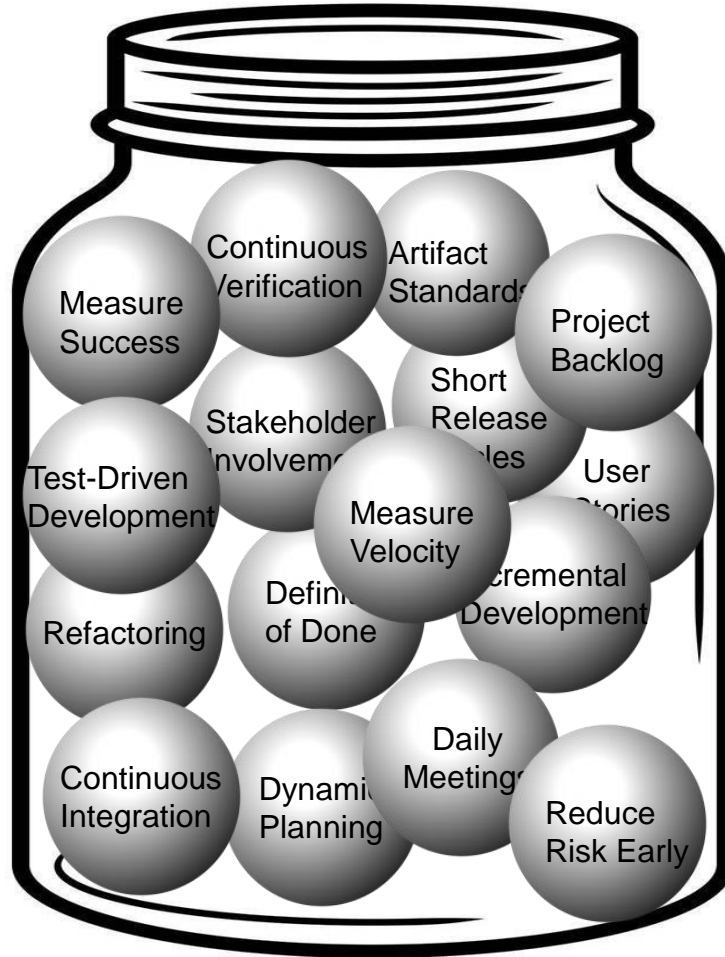
# Agile Practices



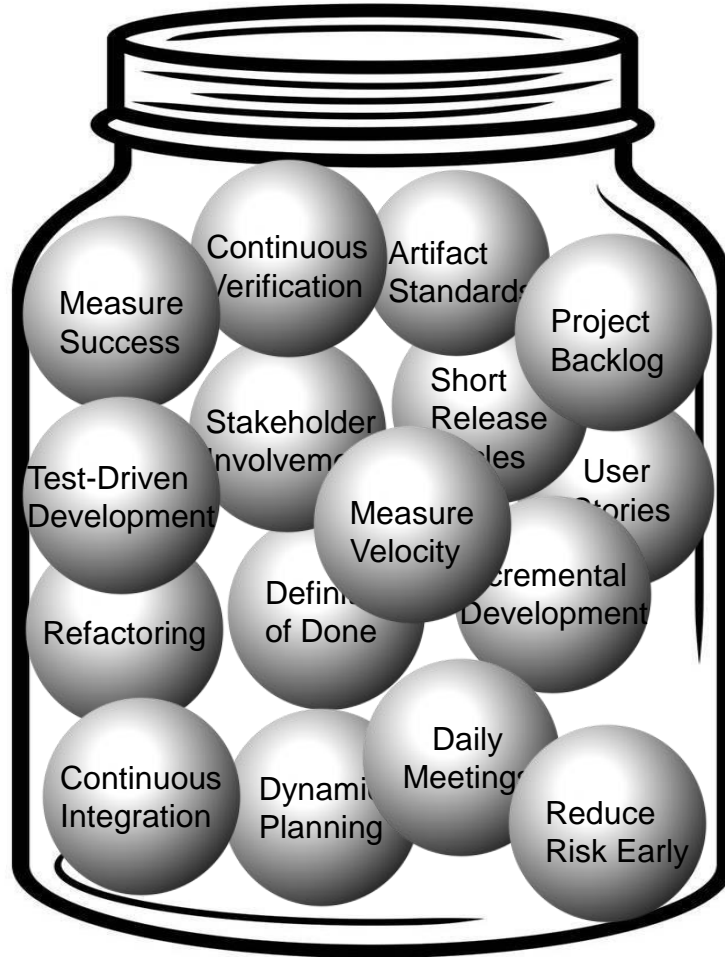
Ensure work products have the right form and content

# Agile Practices

Continuously integrate work product components to ensure on-going consistency



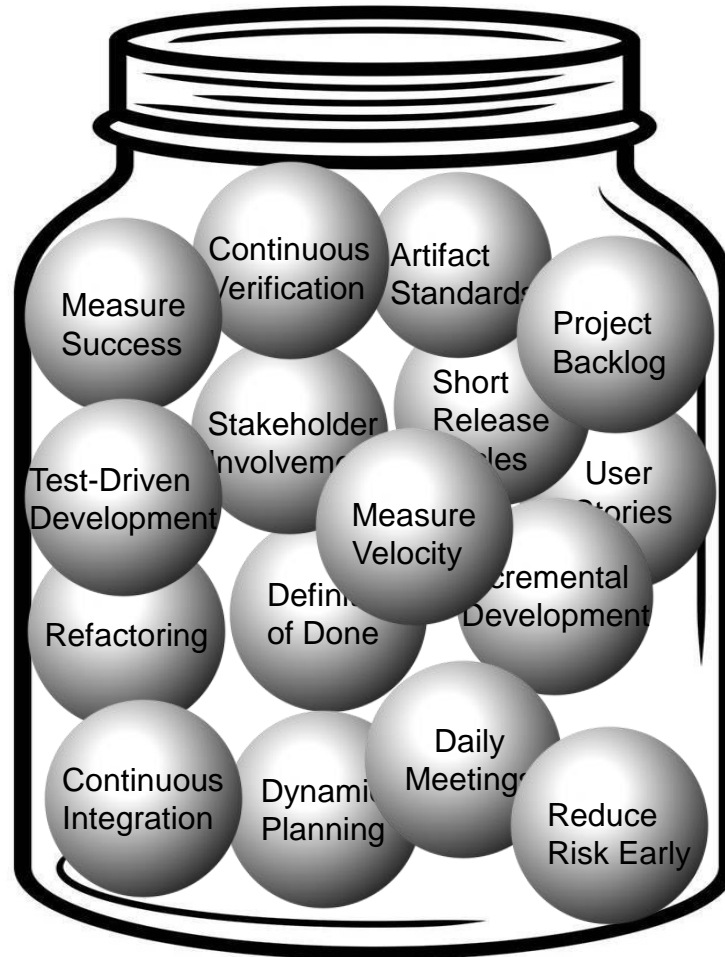
# Agile Practices



Measure progress against plan



# Agile Practices

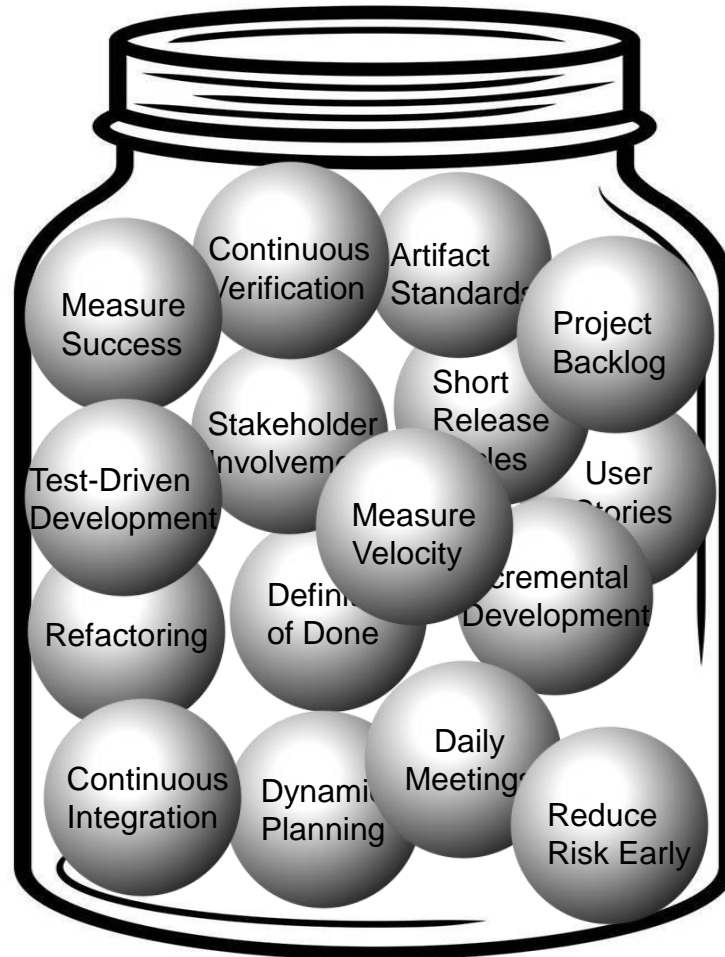


Constantly measure your progress against goals and objectives with metrics, such as

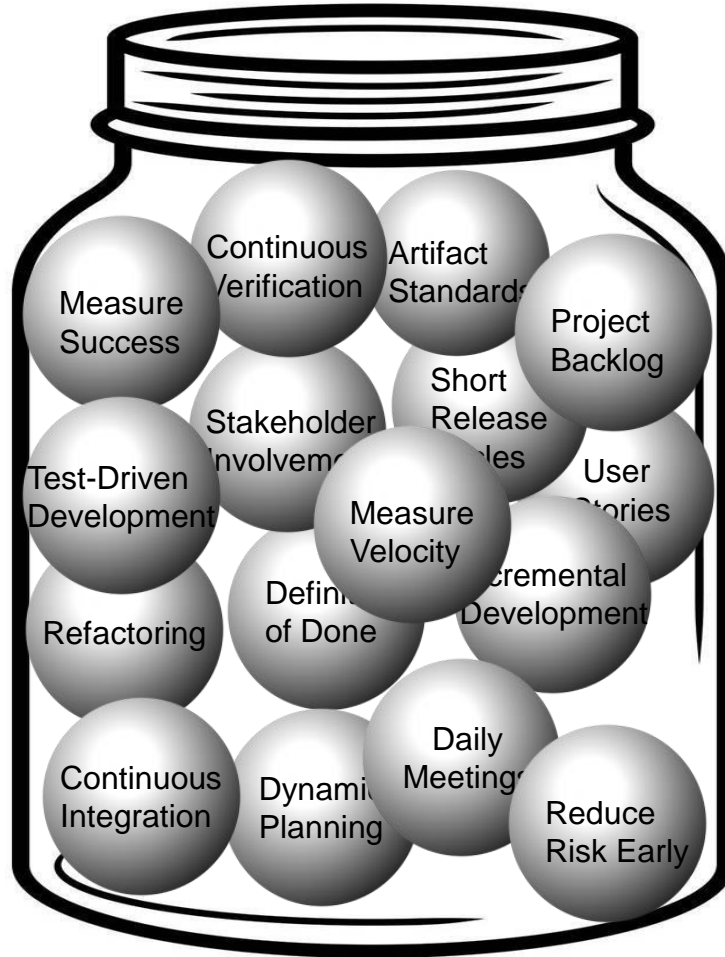
- Velocity
- Deviation from plan
- Burn down rate
- Remaining risk
- Defect rate
- Defects remaining
- Requirements churn
- Test coverage

# Agile Practices

Plan to the best of your information, but plan to replan as you learn more about the product and project

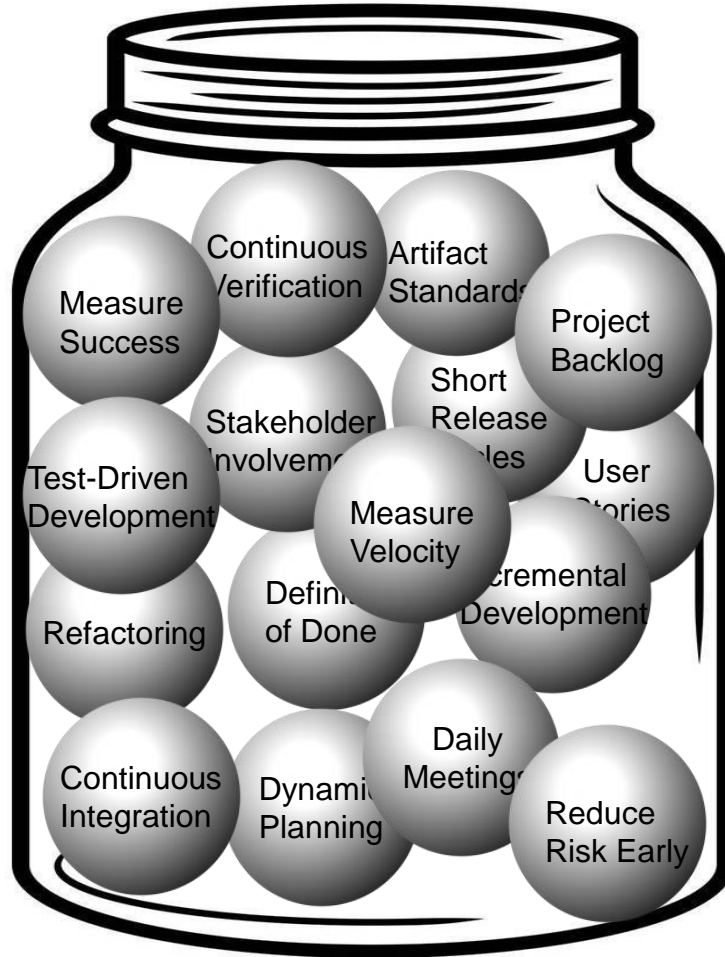


# Agile Practices



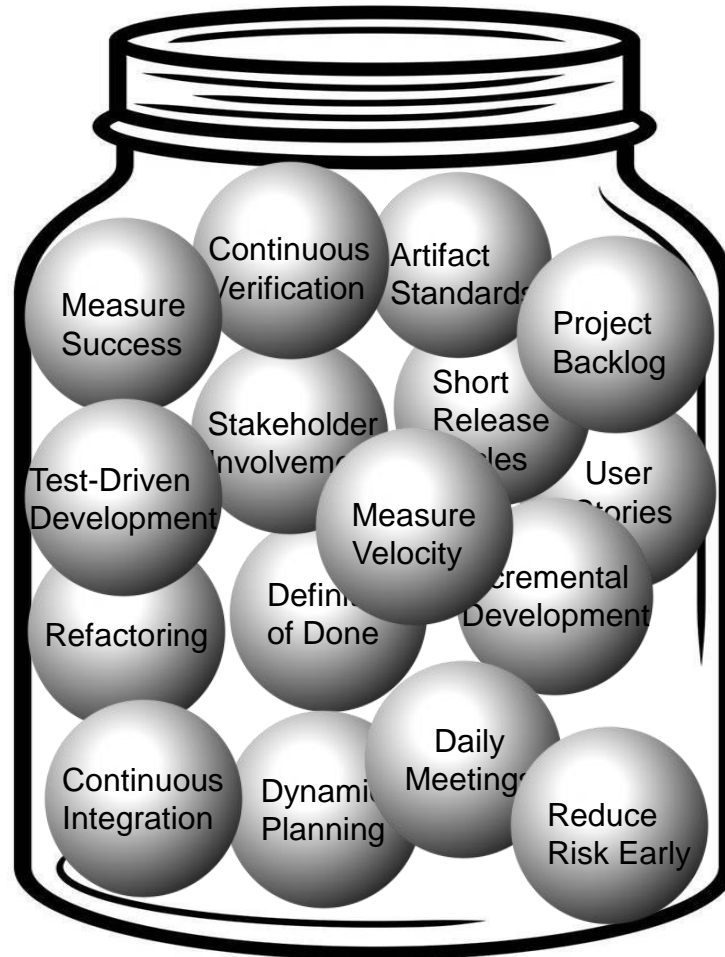
Develop the work products in small increments verifying their correctness as you go

# Agile Practices



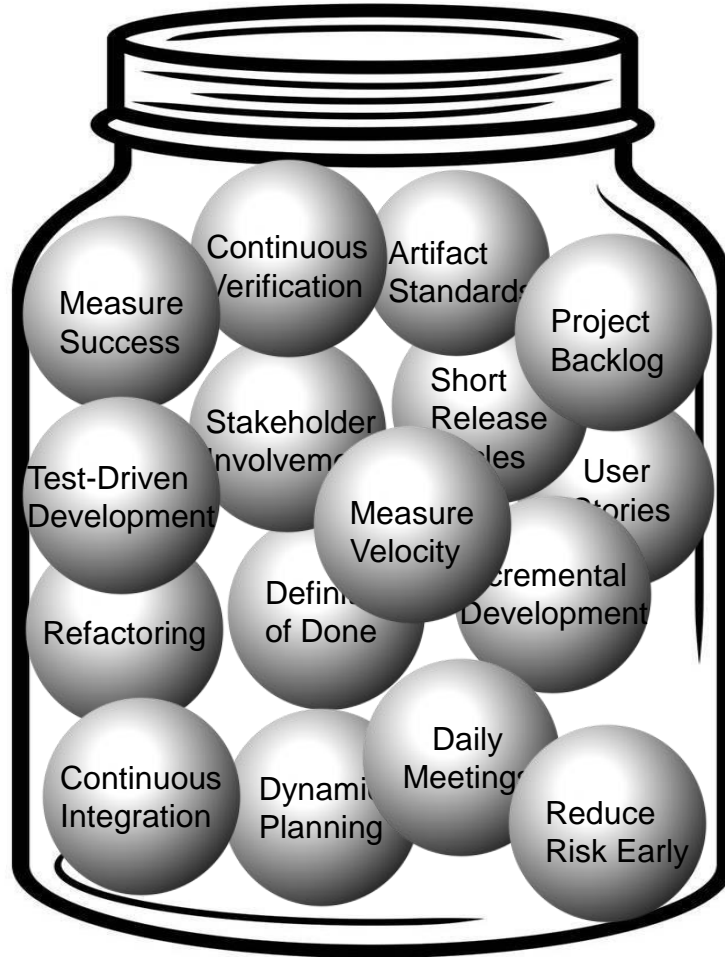
Increments should be small in degree of change and short in duration

# Agile Practices



Be clear on what it means to have successfully and fully reached the objectives of the task or increment and verify that you have done so

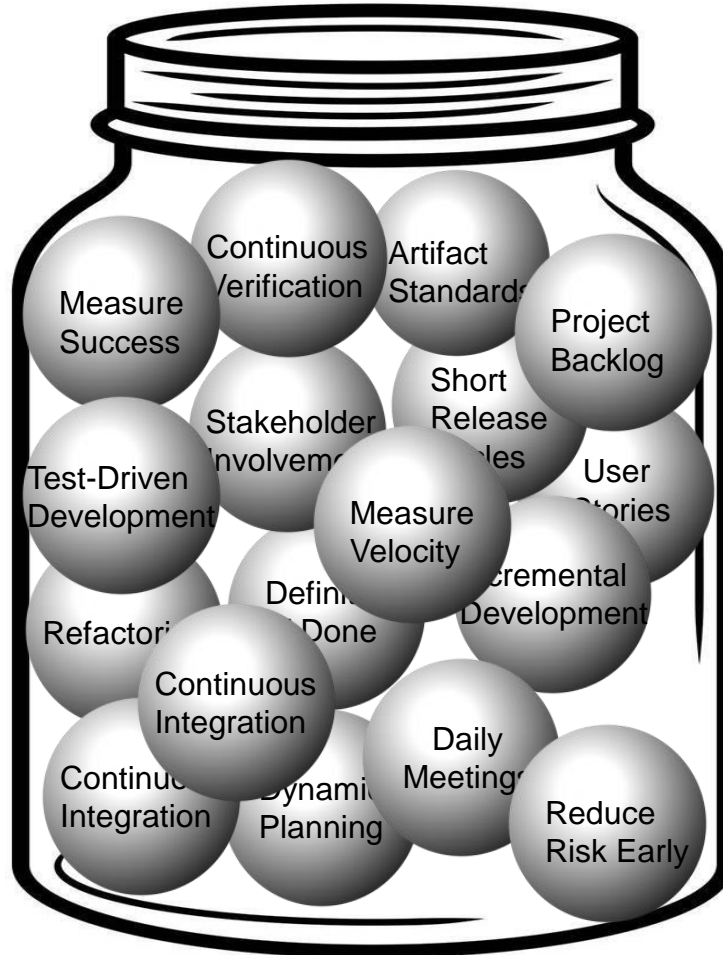
# Agile Practices



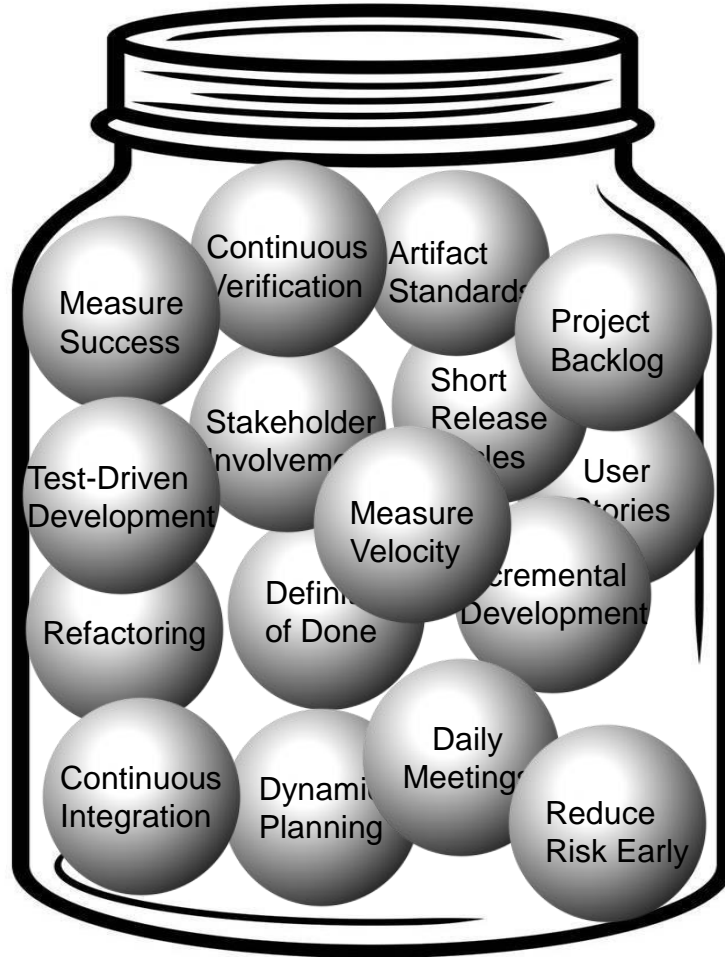
Identify risk to success, plan *spikes* to address them, and execute them within the increments

## Agile Practices

Incremental development is predicated on the idea that change is growth and refactoring is reorganization as more information becomes known



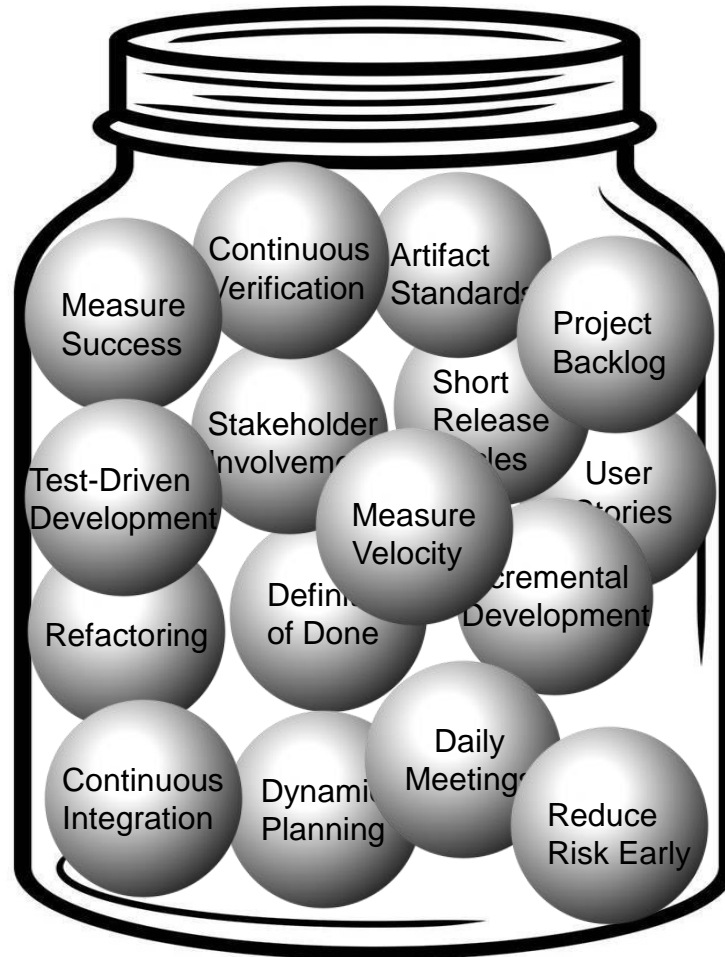
# Agile Practices



Incrementally validate the product with the stakeholder to ensure it meets their needs

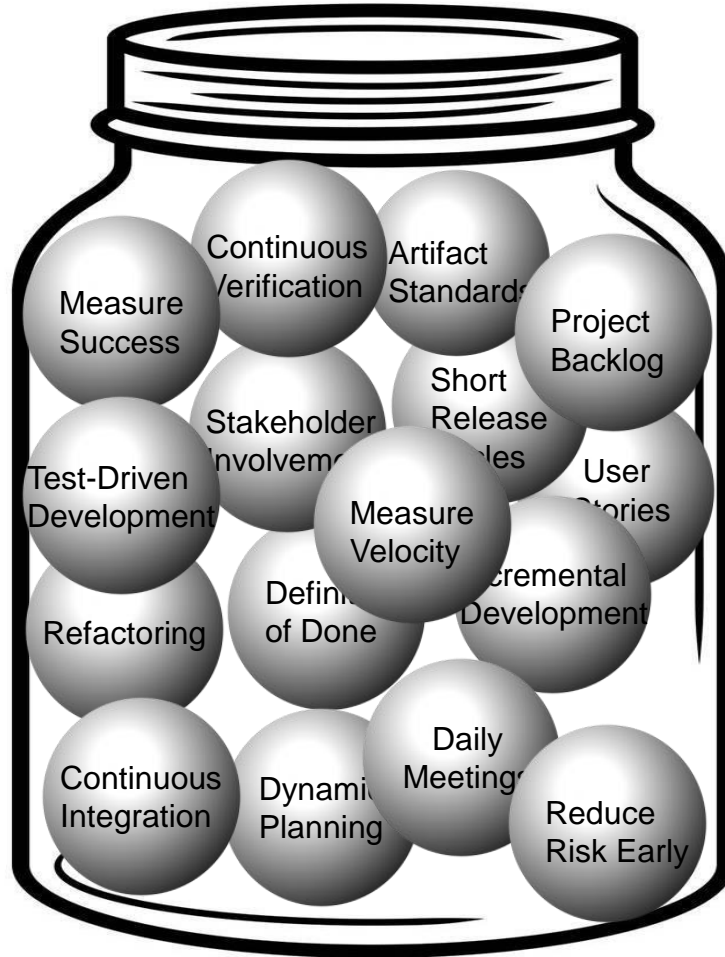


# Agile Practices



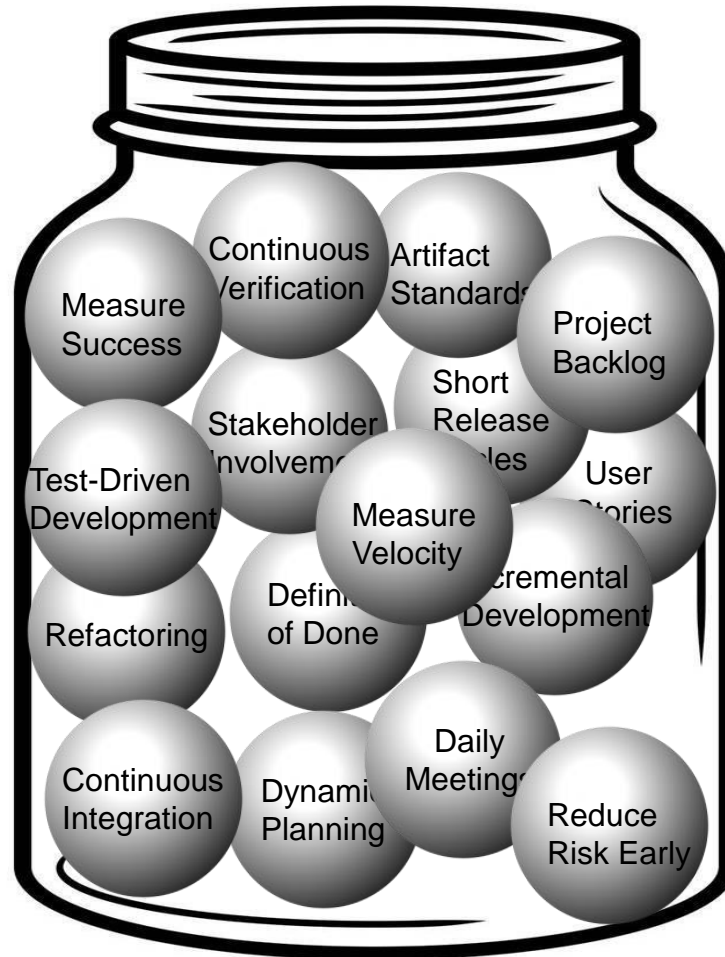
Maintain and burn down a prioritized list of things to do, including features to incorporate, design to include, and risks to reduce

# Agile Practices



Use Cases or User Stories aid in the capture and analysis of requirements

# Agile Practices



Each day, have a short meeting in which team members identify where they are and their “blockers”

## Common Systems Work Products

- Requirements
  - Stakeholder
  - System
  - Subsystem
  - Engineering Specific: Software, Electronics, Mechanical, Pneumatics, Hydraulic, ...
- Architecture
  - Functional
  - Logical
  - Physical
  - Trade studies
- Interfaces
  - System – Actor
  - Subsystem – Subsystem
  - Interdisciplinary (e.g. software – electronics)
- Dependability analysis & specifications
  - Safety
  - Reliability
  - Security
- Trace matrices

# What do we mean by “verification & validation” of work products?

## Semantic Verification

- “correct” (*compliance in meaning*)  
Performed by engineering personnel
- Three basic techniques
- **Semantic review** (subject matter expert & peer) – most common, weakest means
- **Testing** – requires executability of work products, impossible to fully verify
- **Formal methods** – strongest but hard to do and subject to invariant violation

## Syntactic Verification

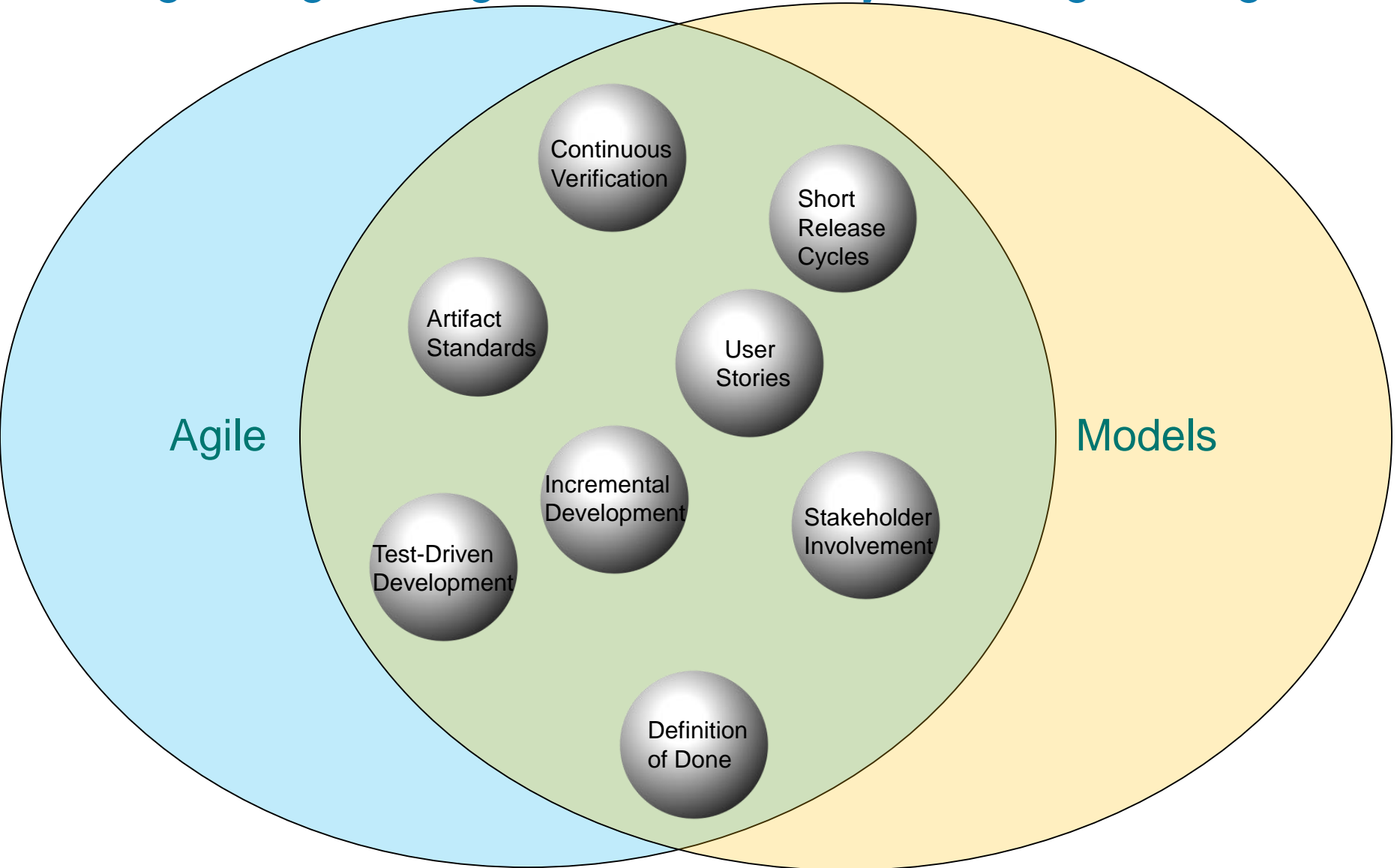
- “well-formed” (*compliance in form*)  
Performed by quality assurance personnel
- **Audits** – work tasks are performed as per plan and guidelines
- **Syntactic review** – work products conform to standard for organization, structure and format



## Validation

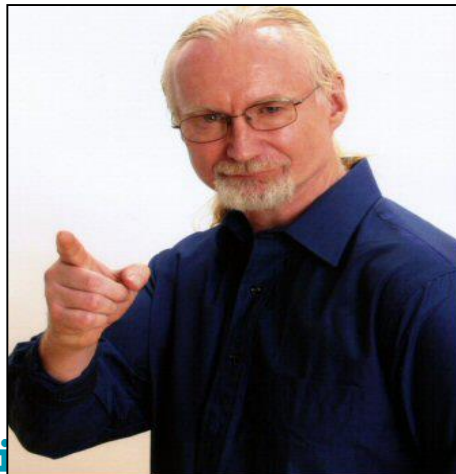
- “meets the stakeholder need”  
Performed by customer + engineering
- Some common techniques
- **Review** – (subject matter expert & customer) – most common, weakest
- **Simulation** – show simulated input → outputs
- **Sandbox** – exploratory usage in constrained environment
- **Flight test** – demonstration of system capabilities
- **Deployment** – early usage of system of partial capability

# Putting the Agile in Agile Model-Based Systems Engineering



## Modeling is Essential for Agile MBSE

- Models:
  - Answer questions
  - Faithfully, precisely, and completely address the purpose and scope of the model
  - Trace to both source and subsequent work products
  - Support autogeneration of subsequent work products, when applicable:
    - Architecture Notebook
    - Interface Specifications (e.g. ICD)
    - Trace matrices
    - Test plans and test cases
    - Project process work and objectives
  - Provide the ability to verify the correctness, accuracy, precision, and completeness of engineering data



All useful models are  
falsifiable

*Bruce Powell Douglass*

# Common SysML Views for Systems Engineering

## Diagrams

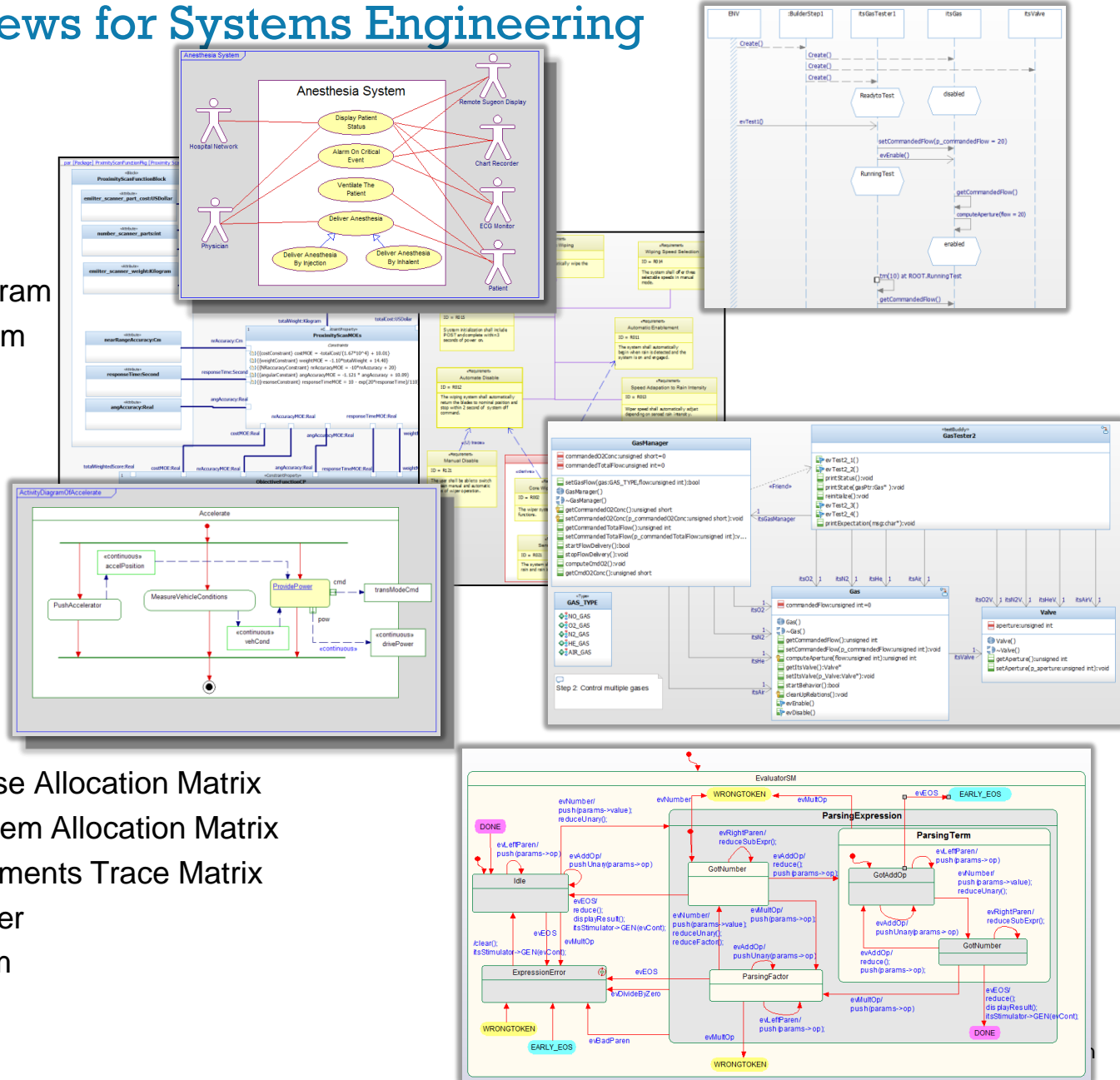
- Use case diagram
- Requirements Diagram
- Block Diagram
  - Block Definition Diagram
  - Internal Block Diagram
- Activity Diagram
- Sequence Diagram
- State Diagram
- Parametric Diagram

## Tables

- Requirements Table
- Allocation Table
- Trace Table

## Matrices (traceability)

- Requirements – Use Case Allocation Matrix
- Requirements – Subsystem Allocation Matrix
- Requirements – Requirements Trace Matrix
  - System → stakeholder
  - Subsystem → system

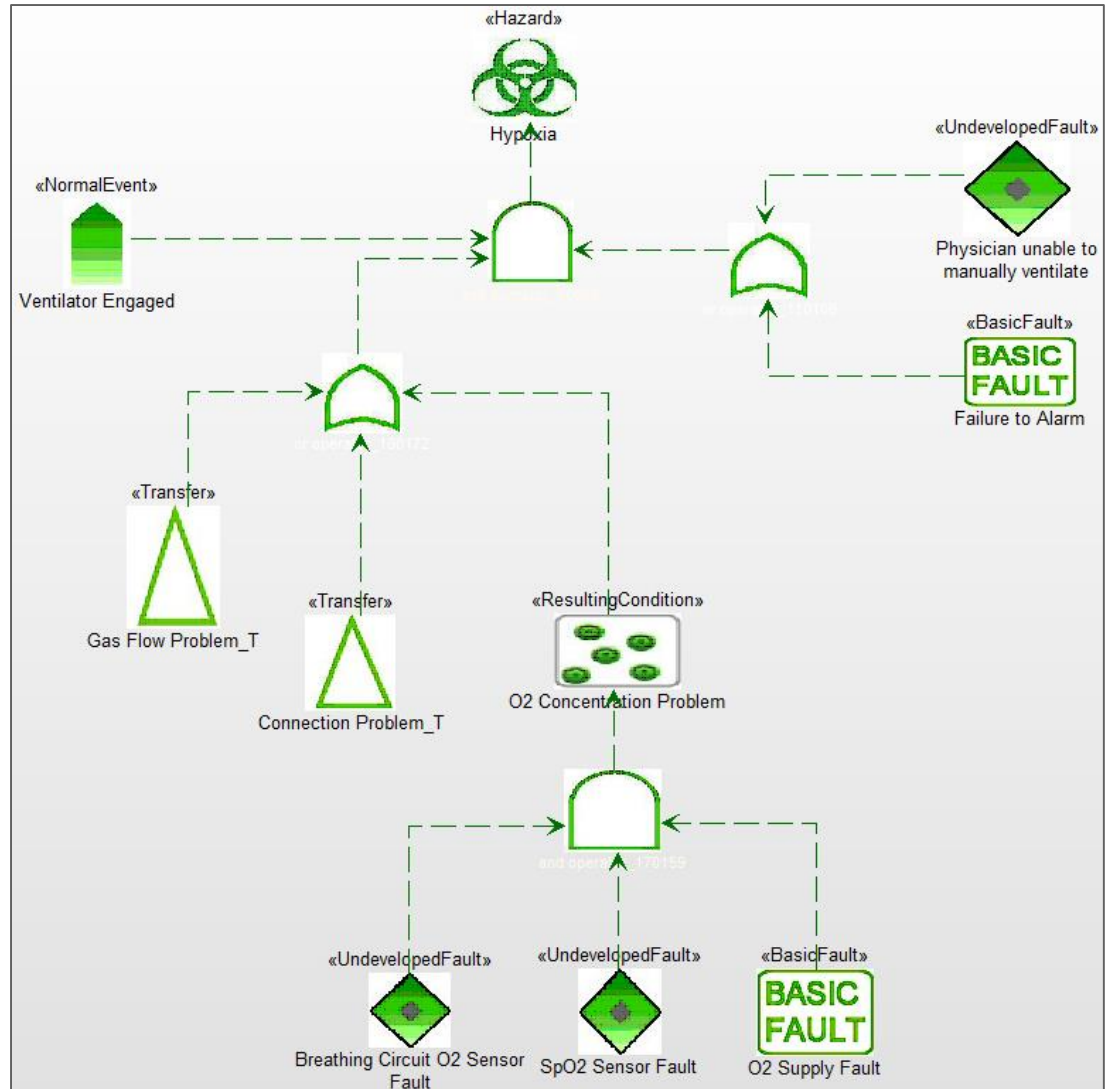




# Integrated Safety and Reliability Analysis

UML Dependability Profile

- 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
  - ▶ Impacts of architectural, technological, and design choices on safety



[http://merlinscave.info/Merlins\\_Cave/Models/Entries/2017/3/3\\_Dependability\\_Analysis\\_Profile.html](http://merlinscave.info/Merlins_Cave/Models/Entries/2017/3/3_Dependability_Analysis_Profile.html)

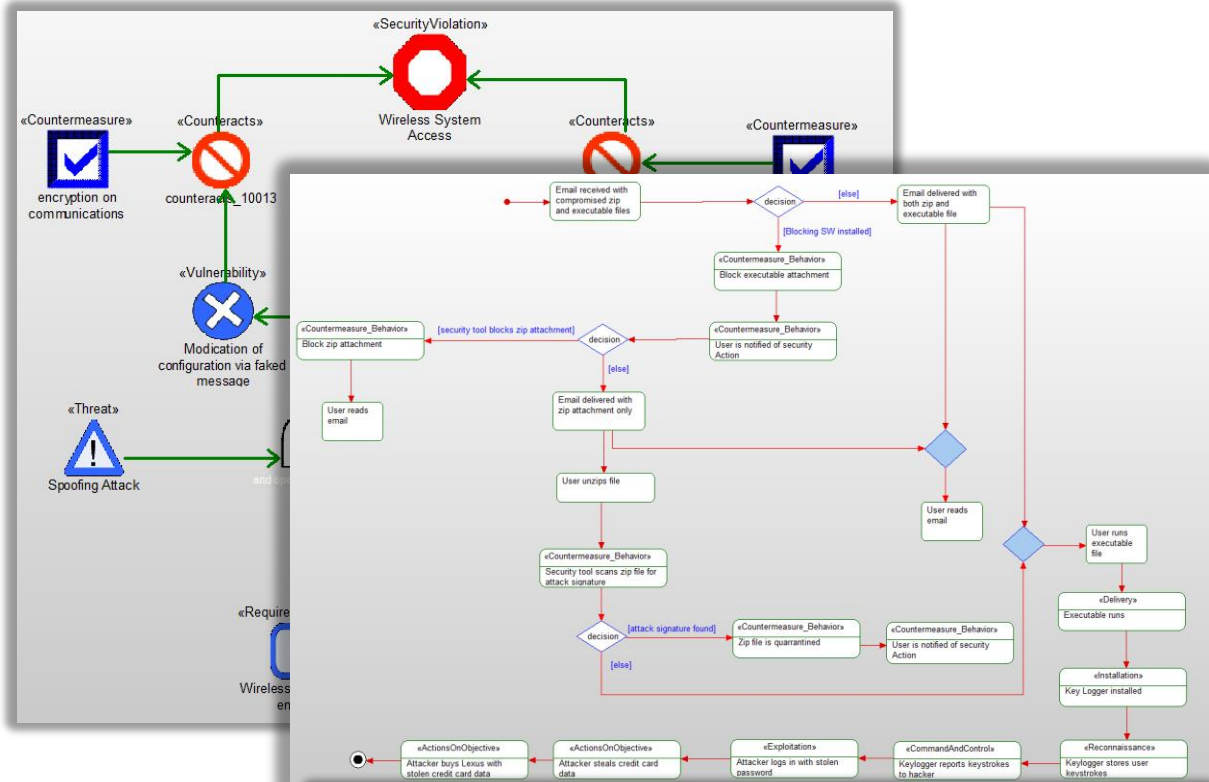
# Model-Based Threat Analysis

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
- Cost of security penetration
- Adequacy of countermeasures
- Who has access to assets



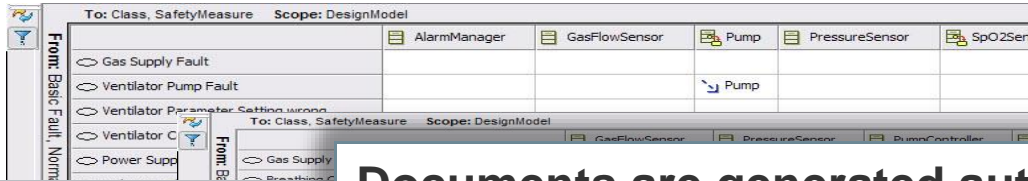
### Threat Analysis Table

Asset value is the value of the asset to be protected (1=very low, 10=very high).  
 Likelihood is the probability of the attack (1=very low, 10=certain).  
 Reproducibility refers to how easy it is to reproduce the attack (for example, does it depend on timing or other circumstances?) (1=hard, 10 = very easy).  
 Exploitability refers to how easy it is to launch the attack (1=very easy, 10=very hard).  
 Breadth is a measure of the extent of the attack. How widespread is it or how many systems are affected? (1=few, 10=very many).  
 Discoverability is how easy is it for outsiders to find out about and exploit the vulnerability (1=very easy, 10=very hard).  
 Threat Priority is the product of the above values and is used to prioritize the threats for countermeasures.

These are in the range of 1 -10

Asset	Vulnerability	Threat Vector	Asset Value	Likelihood of attack	Reproducibility	Exploitability	Breadth	Discoverability	Threat Priority	Countermeasure
Patient Demo-graphic Data	Access via Ethernet	input validation weak	4	7	9	4	1	9	9072	Internal encryption
	Access via USB	Auto-execution of USB SW	4	7	9	3	1	9	0004	Internal encryption
	Access via packet snooping	Messages sent in plain text	4	7	9	5	1	5	10080	Message encryption

# Auto-generation of summary documentation from models



## Failure Mode and Effects Analysis

		Pre-action			Post-action		
Existing Control Measures	Recommendations	Responsible	Actions	Urgency	Severity	RPN	
none	Make pedal assembly self-lubricating	Joe	Added scaled piston with lubrication	2	9	180	
start up (cann check with sensor)	Use three pedal position sensors	Susan	Added two more sensors with voting	2	9	144	
Continuous monitoring of CAN bus	None	n/s					
Continuous monitoring of	Update monitoring to send lifeclicks to error node		Updated lifeclick			102	

Documents are generated automatically from engineering work in models

Typical auto-generated documentation includes

- Traceability matrix
- Hazard Analysis
- FMEA / FMECA
- Cyberphysical threat analysis table
- Interface Control Document
- Design Description
- Architecture Notebook

### INTERFACE CONTROL DOCUMENT

Source: Model A76-BrakingSubsystemModel v2.1

Subsystem: BrakingManagementSubsystem

Interface: iBrakingForce

Service: GetBrakingForce

Data: brakingForce

Type: Scaled 32-bit integer  
 Media: CAN Bus message  
 Range: 0 .. 100N  
 Accuracy: ± 0.05 N  
 Return value: none  
 Rate: 5 ms  
 Worst Case Response time: 1 ms

Interface: iBrakingCommands

Service: EnableBrakingAugmentation

Data: enable

Type: 1-bit  
 Media: CAN Bus message  
 Range: 0 (FALSE) .. TRUE (1)  
 Accuracy: N/A  
 Return value: ACK\_Type  
 Rate: no faster than 2/s  
 Worst Case Response Time: 2 ms

Service: PowerOnSelfTest

Data: None

Return value: POST\_Return\_Type  
 Rate: No faster than 1/10 minutes  
 Worst Case Response Time: 1000 ms

Service: CalibrateBrake

### Threat Analysis Table

Asset value is the value of the asset to be protected (1=very low, 10=very high)

Likelihood is the probability of the attack (1=very low, 10=central)

Reproducibility refers to how easy it is to reproduce the attack (for example, does it depend on timing or other conditions)

Exploitability refers to how easy it is to launch the attack (1=very easy, 10=very hard)

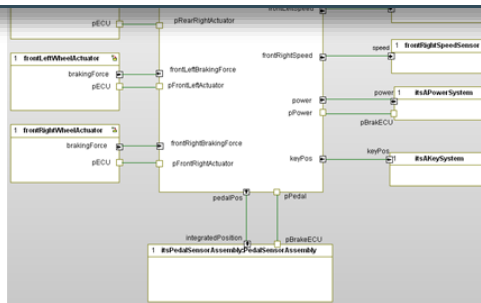
Breadth is the measure of the extent of the attack. How widespread is it or how many systems are affected? (1=very narrow, 10=very broad)

Discoverability is how easy it is for outsiders to find out about and exploit the vulnerability (1=very easy, 10=very hard)

Threat Priority is the product of the above values and is used to prioritize the threats for countermeasures.

These are in the

Asset	Vulnerability	Threat Vector	Asset Value	Likelihood of attack	Reproducibility	Exploitability
Patient Demographic Data	Access via Ethernet	Input validation leak	4	7	9	4
	Access via USB	Auto-execution of USB SW	4	7	9	3
	Access via packet snooping	Messages sent in plain text	4	7	9	5



Failure Mode	Time	Probability	Severity	Risk	Safety integrity level
Overpressure can damage the lungs. This is an especially severe trauma, possibly fatal, to neonates.	200 milliseconds	1.00E+04	4	3.00E+04	3
Hyperoxia problems are usually limited to neonates, where it can cause blindness.	10 minutes	1.00E+05	4	4.00E+05	4
Inadequate anesthesia leads to patient discomfort and memory retention of the surgical procedures. This is normally not life threatening but can be severely disorienting.	5 minutes	1.00E+04	2	2.00E+04	2
Over anesthesia can lead to death.	3 minutes	1.00E+03	4	4.00E+03	4
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

## So What IS a Model then?

**Modeling** is the development of a semantically correct set of engineering data of relevant systems and their properties

**Models** have views (e.g. diagrams)

**Diagrams** show subsets of eng. data

**Diagrams** have singular purpose

**Diagrams** answer questions

**Diagrams** support specific reasoning

**Models** have scope

**Models** have purpose

**Models** have accuracy

**Models** have fidelity

**Models** are falsifiable

**Models** are verifiable

**Models** *are*  
*interconnected data!*

# Harmony Agile MBSE Delivery Process



Team (IBM)

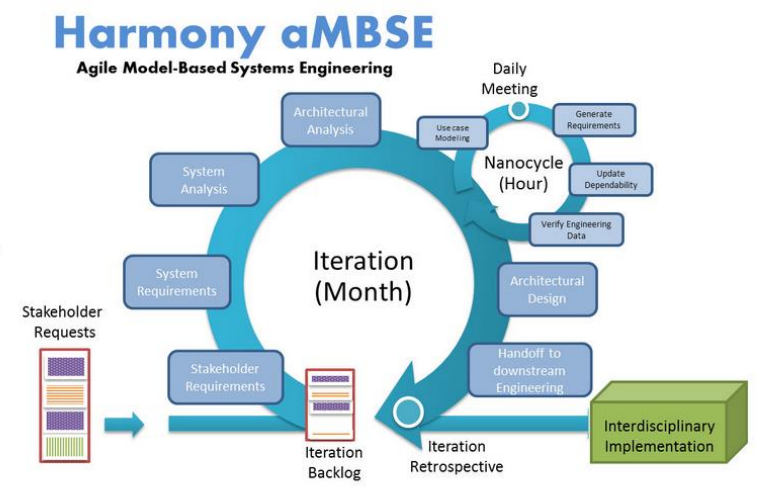
- Welcome to the Rational Harmony Agile Model-Based Systems Engineering
- Getting Started
- Delivery Processes
- Practices
- Roles Sets
- Tasks
- Work Products
- Guidance
- Tools
- Release Info

Welcome to the Rational Harmony Agile Model-Based Systems Engineering

Welcome to the Rational Harmony Agile Model-Based Systems Engineering

The Rational Harmony Agile Model-Based Systems Engineering (aMBSE) process is a delivery process for the development of systems engineering data and work product using both model-based systems techniques with UML and SysML but is at the same time agile and incorporates agile practices for improved quality and engineering efficiency.

**Main Description**

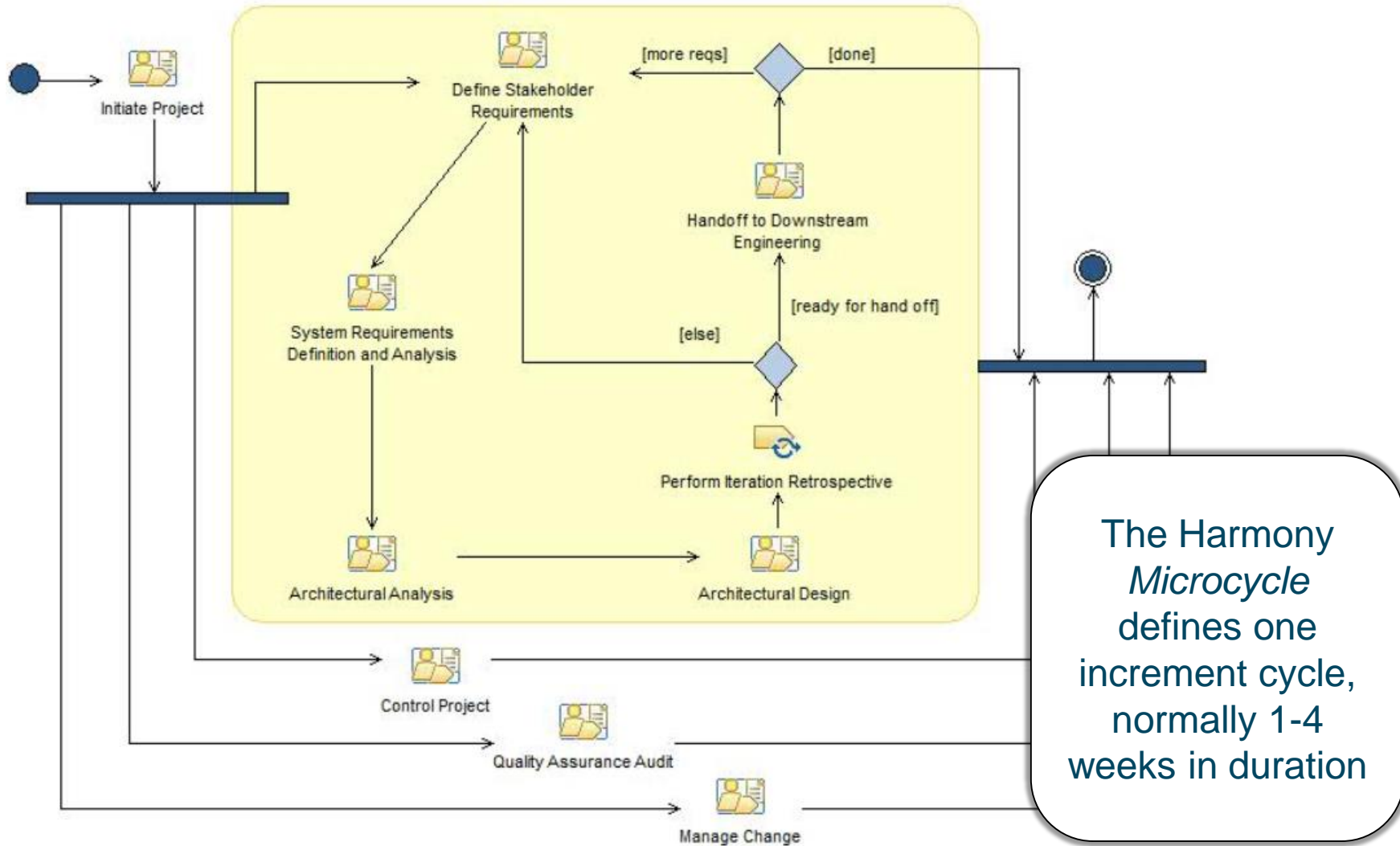


With the initial release of the UML in 1995, systems engineers had a standard language in which they could express requirements, architectures, designs, and other kinds of engineering data. However, there was widespread belief that the Unified Modeling Language (UML) itself was too “software oriented” for general use in systems engineering which led to the development and release of the Systems Modeling Language (SysML). UML and SysML provide a number of key advantages for the development of system engineering data:

- Precision of engineering data
- Data consistency across work products and engineering activities
- A common source for engineering truth
- Improved visualization and comprehension of engineering data
- Ease of integration of disparate engineering data
- Improved management and maintenance of engineering data

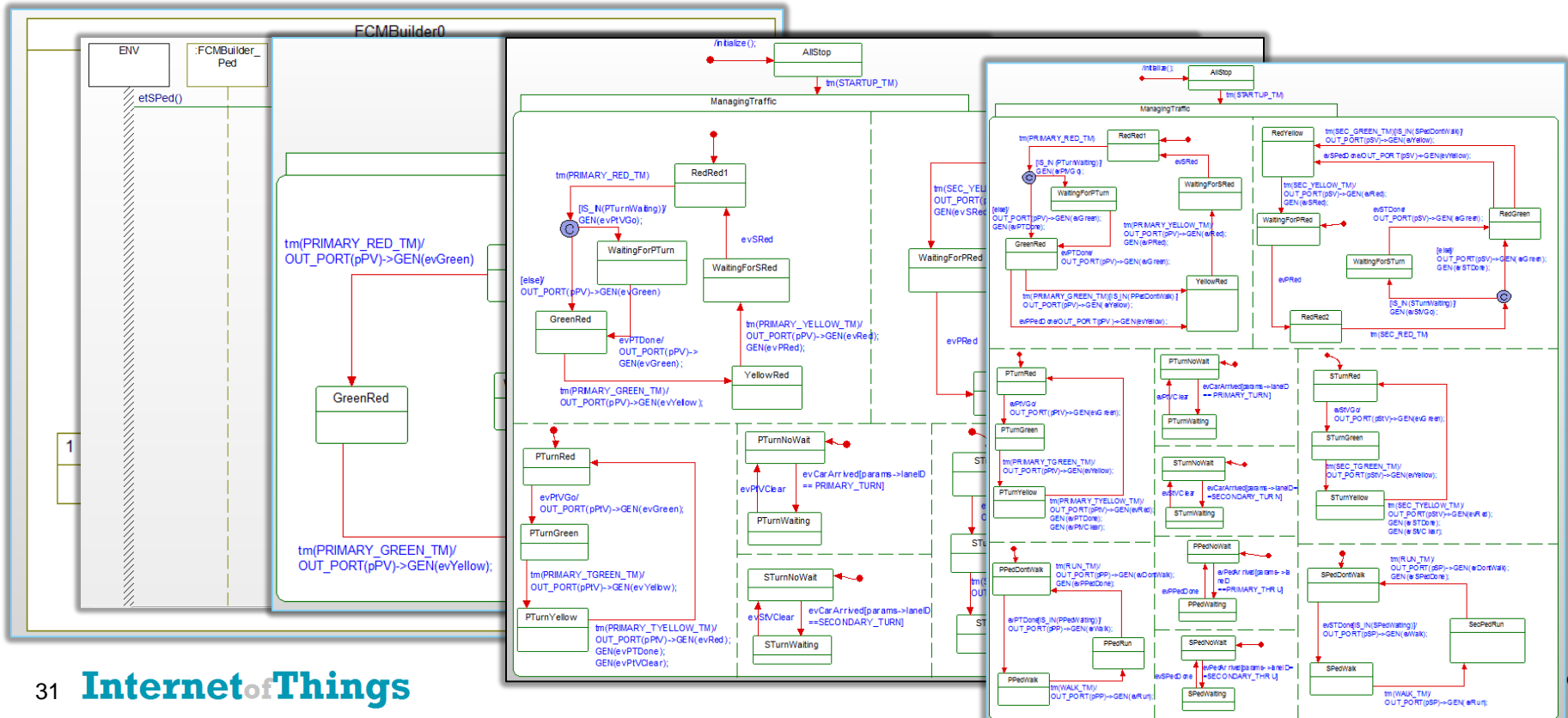
# Harmony aMBSE Practices: Incremental Development

Harmony aMBSE Delivery Process

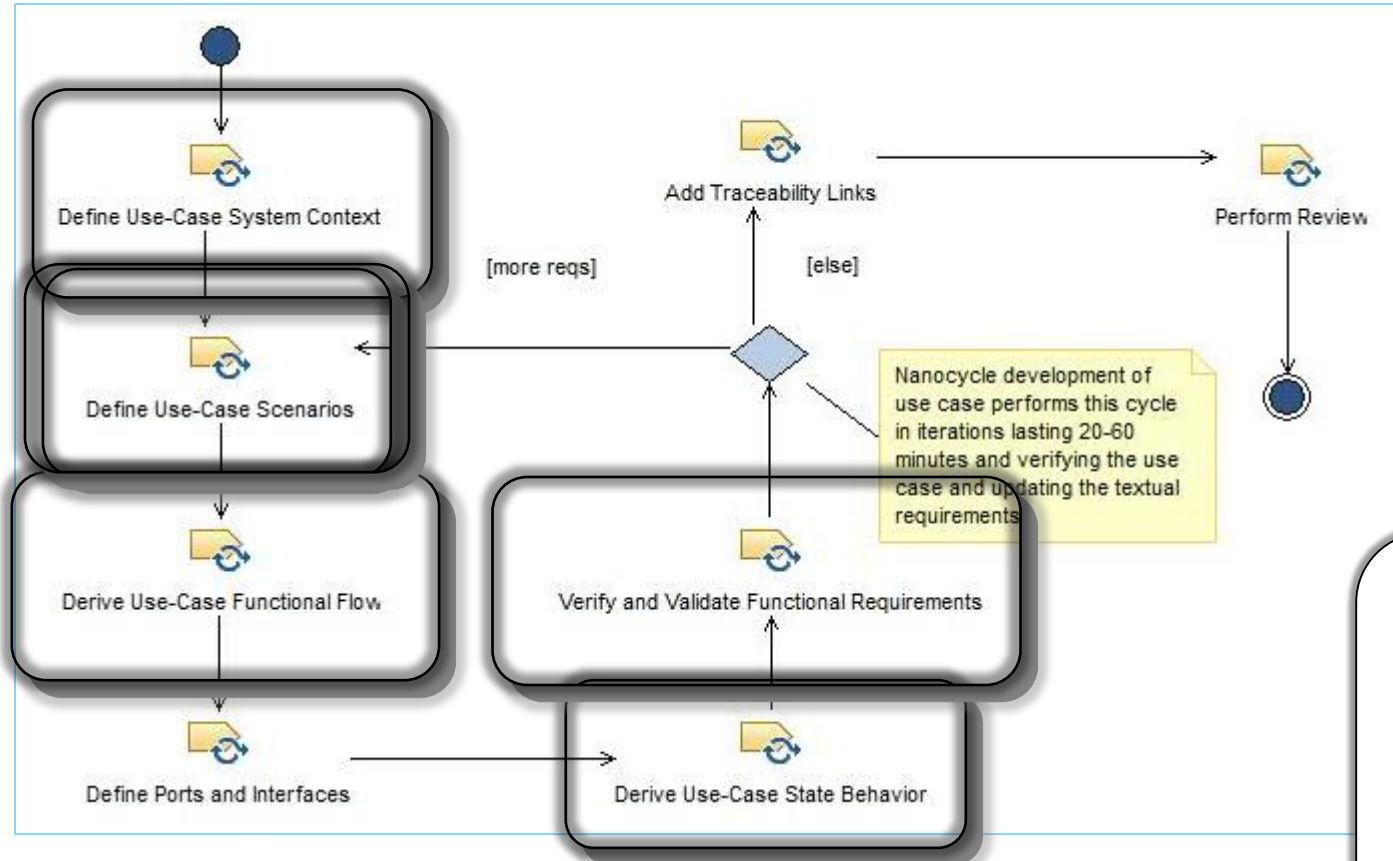


# Test-Driven Development for MBSE Work Products

- 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, architecture and design models

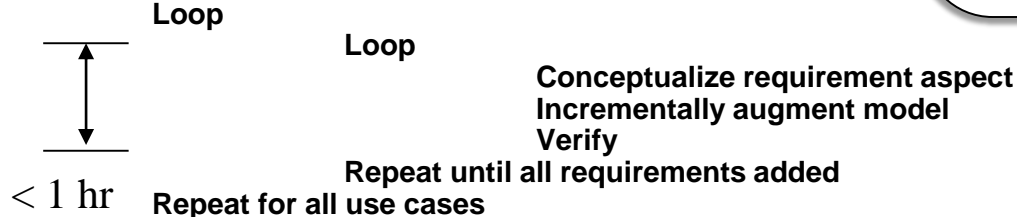


# Scenario Driven Use Case Construction / Validation



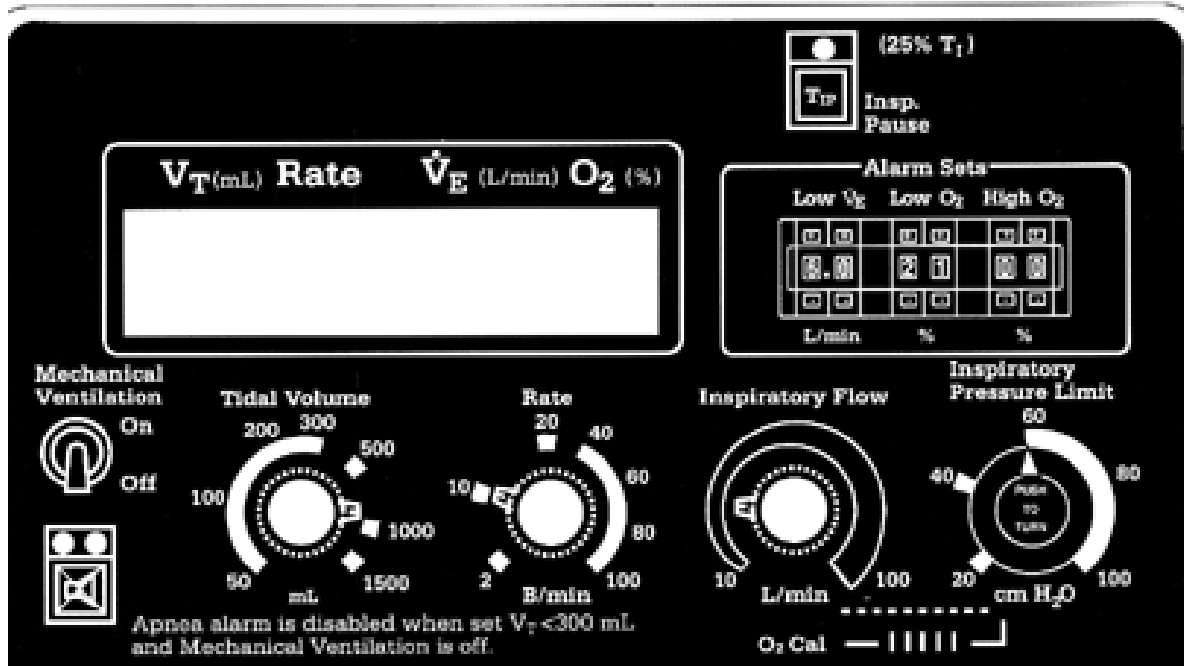
The Harmony Nanocycle defines a short work product development cycle, 20-60 minutes in duration

### Making it Agile





## Exploring Requirements – Then vs Now



### Questions

- ▶ What happens if the user turns the  $V_t$  knob *and then turns the Rate knob before pushing in to confirm?*
- ▶ How to I abort a  $V_t$  change once started?
- ▶ What happens if the user tries to set the  $V_t$  to 1500 and the system is configured for neonates?

- The system shall set  $V_t$  in the range of 50 to 1500 ml
- The user shall push in the knob to confirm the  $V_t$  before the value becomes active
- While monitoring, the system will display measured  $V_t$  output
- Respiration Rate shall be set in the range of 2 – 100 b/m
- The user shall push in the Rate knob to confirm the Rate value before it becomes active
- Neonate mode shall support  $V_t$  from 50 to 500 ml
- ...

## The Traditional Option

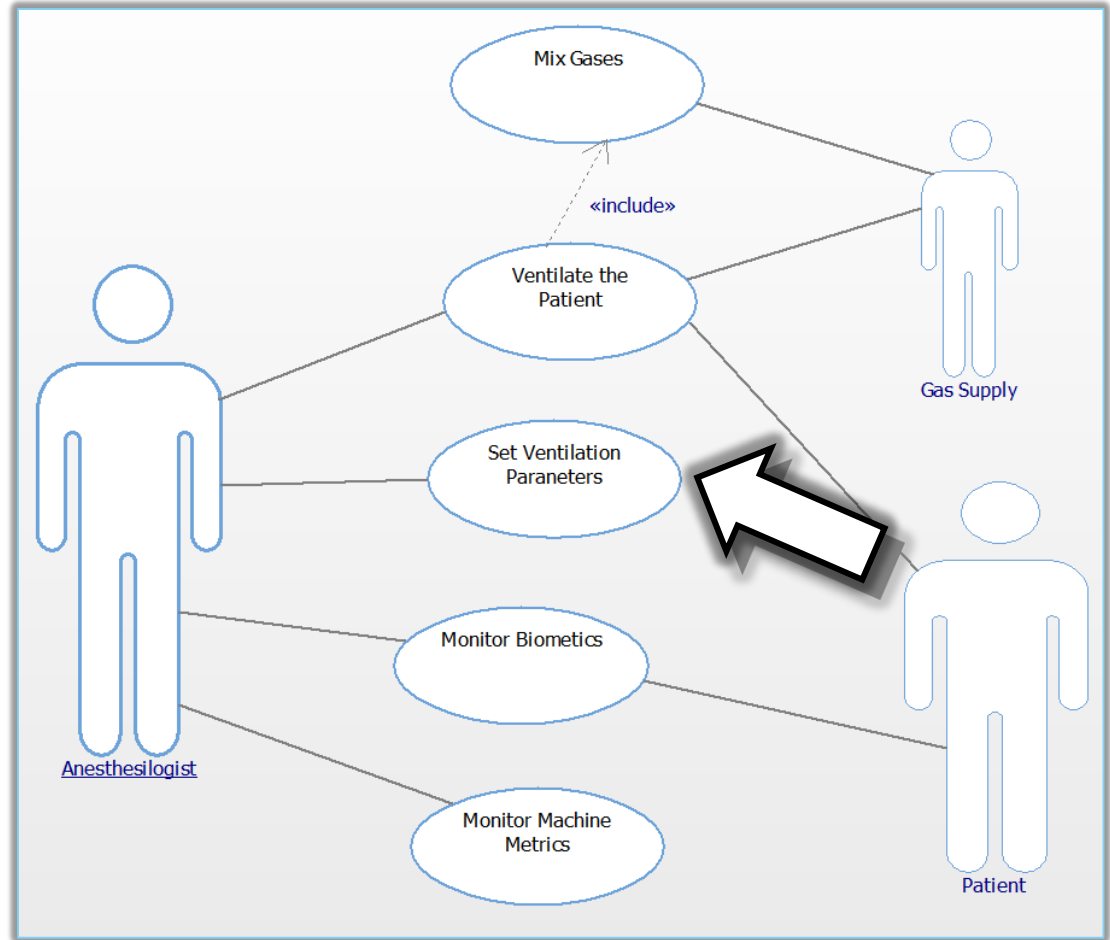
- Search through the (hundreds to thousands of) requirements to find the one that answers the question
- Once you've determined that it isn't in the spec, go back to the stakeholder(s) and ask them what you should do
- Or make up something that seems reasonable



# Executable Requirements Models

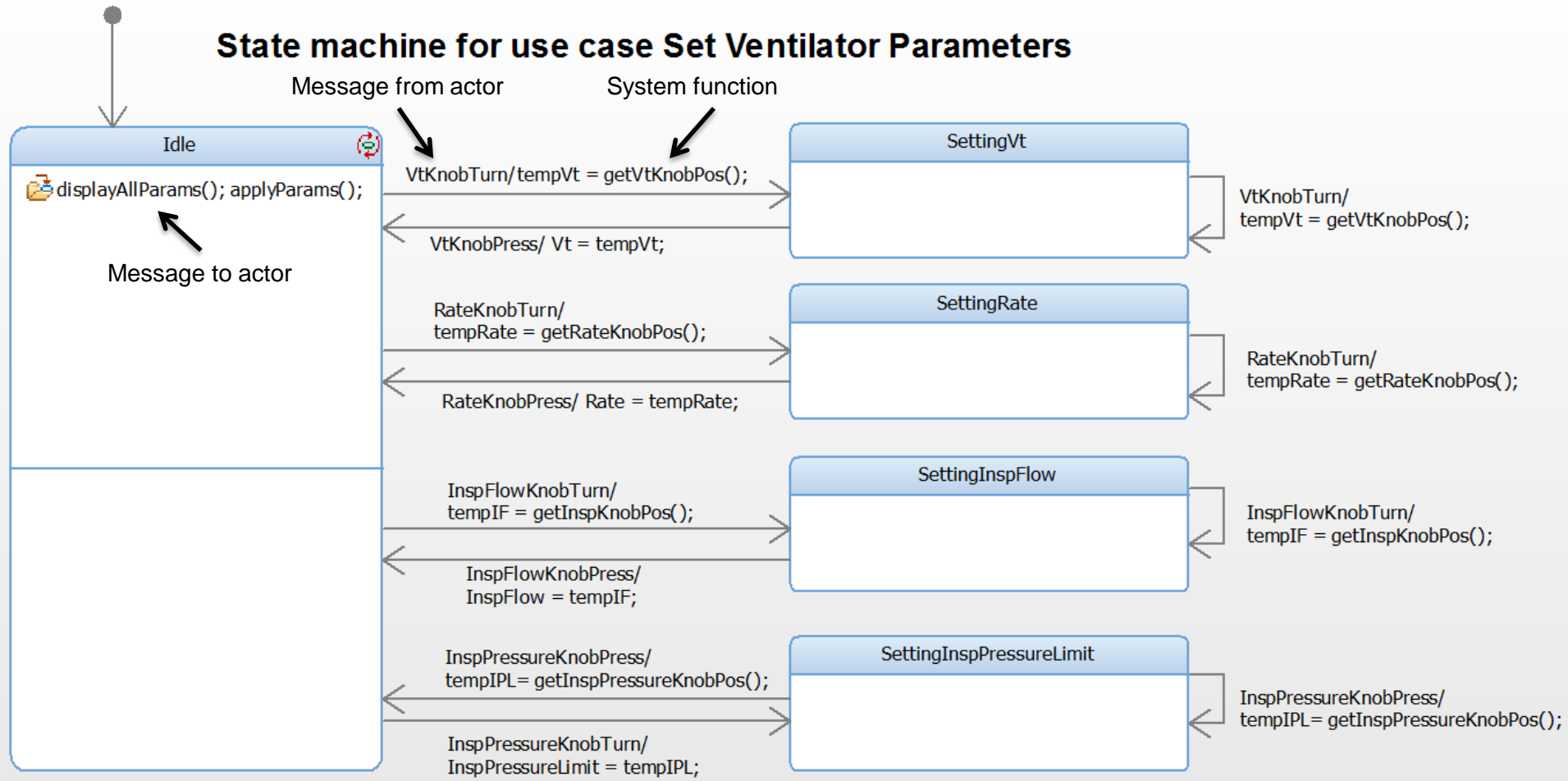
## Benefits

- Ability to explore and evaluate requirements
- Improve ability to identify requirement defects:
  - Missing requirements
  - Incomplete requirements
  - Conflicting requirements
- Provides facilities to do “what about this ...?” analysis
- Reliably results in *better requirements*



# The Modeling Option

**State machine for use case Set Ventilator Parameters**



Note that this state machine is a precise specification of **requirements**, and not design

# Running the Requirements Model

The screenshot displays two views from the IBM Rational Rhapsody Developer. On the left is a 'Sequence Diagram: Animated sequencediagram\_3\*' showing an actor named ':Anesthesiologist' and a participant ':ucSetParameters'. The actor sends a 'SettingValues' message to the participant, which then enters an 'Idle' state. The participant performs two self-messages: 'displayAllParams()' and 'applyParams()'. On the right is a 'Statechart of : ucSetParameters - ucSetParameters' titled 'State machine for use case Set Ventilator Parameters'. It features an 'Idle' state (highlighted in pink) with a green arrow indicating the current state. Transitions from 'Idle' lead to four sub-states: 'SettingVt', 'SettingRate', 'SettingInspFlow', and 'SettingInspPressureLimit'. Each transition is triggered by a specific event (e.g., 'VtKnobTurn/tempVt = getVtKnobPos();') and results in an action (e.g., 'tempVt = getVtKnobPos();').

Actor

Current state

Use case

Behaviors executed

Animated Sequence diagram automatically generated from the model execution

Animated state diagram shows the state machine execution

# Running the Requirements Model

The screenshot displays the IBM Rational Rhapsody Developer interface. The left pane shows a sequence diagram titled 'Animated sequencediagram\_3\*' with participants ':Anesthesiologist' and ':ucSetParameters'. The right pane shows a statechart titled 'State machine for use case Set Ventilator Parameters'.

**Sequence Diagram:** Shows the interaction between the Anesthesiologist and the ucSetParameters use case. The Anesthesiologist sends a 'VtKnobTurn(n = 89)' message to the ucSetParameters, which then updates the 'SettingVt' state.

**Statechart:** A state machine for the use case 'Set Ventilator Parameters'. It starts in an 'Idle' state. Transitions to other states are triggered by specific events and actions:

- Idle to SettingVt:** Triggered by 'VtKnobTurn/tempVt = getVtKnobPos();'. Action: 'displayAllParams(); applyParams();'.
- SettingVt to Idle:** Triggered by 'VtKnobPress/ Vt = tempVt;'.
- Idle to SettingRate:** Triggered by 'RateKnobTurn/tempRate = getRateKnobPos();'.
- SettingRate to Idle:** Triggered by 'RateKnobPress/ Rate = tempRate;'.
- Idle to SettingInspFlow:** Triggered by 'InspFlowKnobTurn/ tempIF = getInspKnobPos();'.
- SettingInspFlow to Idle:** Triggered by 'InspFlowKnobPress/ InspFlow = tempIF;'.
- Idle to SettingInspPressureLimit:** Triggered by 'InspPressureKnobPress/ tempIPL = getInspPressureKnobPos();'.
- SettingInspPressureLimit to Idle:** Triggered by 'InspPressureKnobTurn/ InspPressureLimit = tempIPL;'.

Anesthesiologist turns the  $V_t$  knob

# Running the Requirements Model

The screenshot displays the IBM Rational Rhapsody Developer interface for a C++ project named 'Ventilator.rpy'. The main workspace is split into two views:

- Sequence Diagram: Animated sequencediagram\_3**: Shows the interaction between an `Anesthesiologist` actor and a `ucSetParameters` use case. The `Anesthesiologist` sends `SettingValues` messages to `ucSetParameters`. `ucSetParameters` has an `Idle` state and a `SettingVt` state. The diagram shows the `SettingVt` state being entered and then exited.
- Statechart of: ucSetParameters - ucSetParameters**: A state machine diagram for the `ucSetParameters` use case. It starts in an `Idle` state. Transitions to other states are triggered by events and guarded by conditions:
  - `SettingVt`: Triggered by `VtKnobTurn/tempVt = getVtKnobPos();` and `VtKnobPress/ Vt = tempVt;`. It has a self-loop with guard `VtKnobTurn/ tempVt = getVtKnobPos();`.
  - `SettingRate`: Triggered by `RateKnobTurn/ tempRate = getRateKnobPos();` and `RateKnobPress/ Rate = tempRate;`.
  - `SettingInspFlow`: Triggered by `InspFlowKnobTurn/ tempIF = getInspKnobPos();` and `InspFlowKnobPress/ InspFlow = tempIF;`.
  - `SettingInspPressureLimit`: Triggered by `InspPressureKnobPress/ tempIPL = getInspPressureKnobPos();` and `InspPressureKnobTurn/ InspPressureLimit = tempIPL;`.

A callout box with a black border and white background contains the text: **Anesthesiologist turns the  $V_t$  knob more ...**. A black arrow points from this text to the `SettingVt` state in the statechart, and another black arrow points from the `SettingVt` state in the statechart to the `SettingVt` state in the sequence diagram.

# Running the Requirements Model

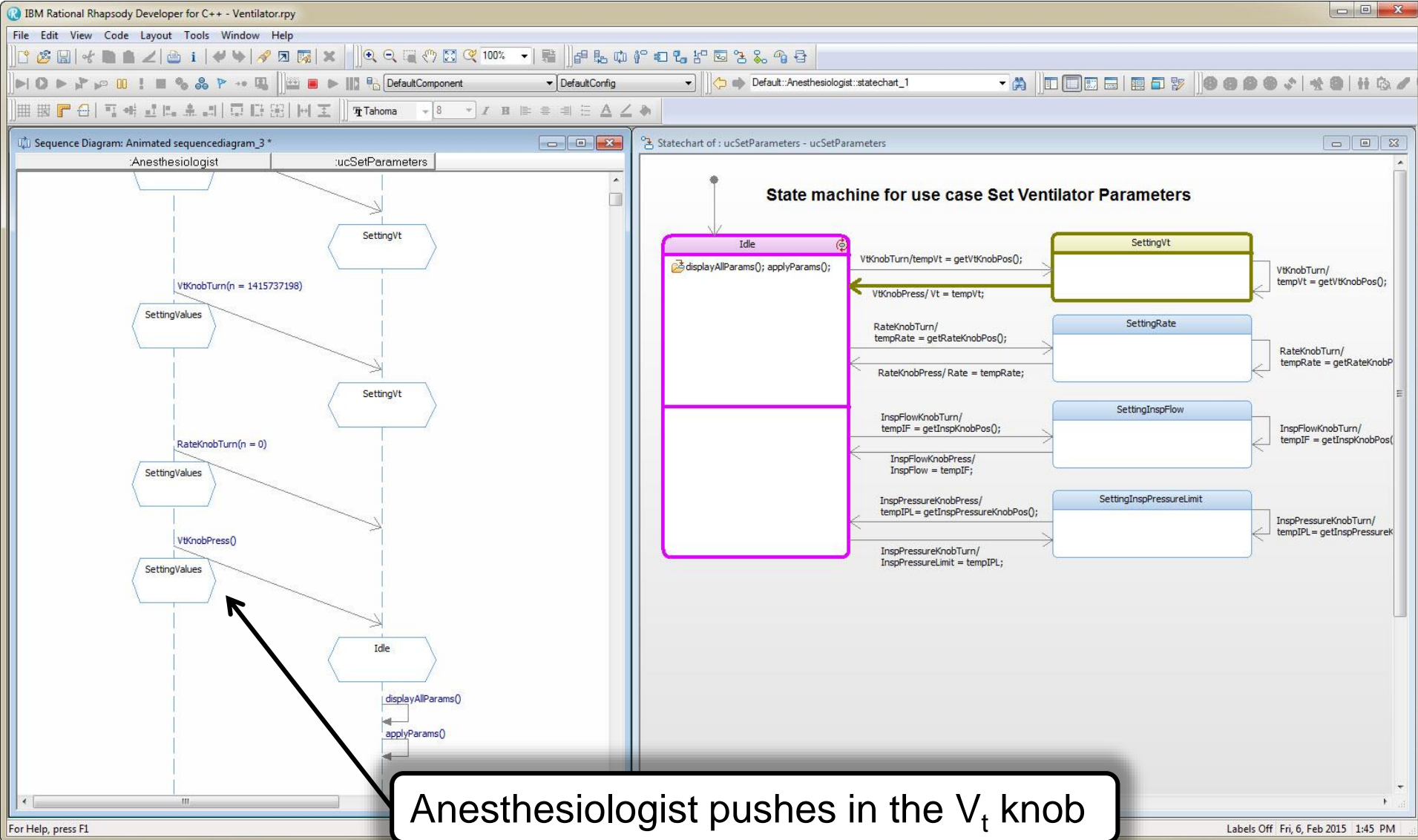
The screenshot displays the IBM Rational Rhapsody Developer interface. On the left, a sequence diagram titled "Animated sequencediagram\_3\*" shows interactions between an "Anesthesiologist" and "ucSetParameters". The diagram includes messages like "VtKnobTurn(n = 1415737198)", "SettingValues", and "RateKnobTurn(n = 0)". On the right, a statechart titled "State machine for use case Set Ventilator Parameters" shows states: "Idle", "SettingVt", "SettingRate", "SettingInspFlow", and "SettingInspPressureLimit". Transitions are labeled with events and actions, such as "VtKnobTurn/tempVt = getVtKnobPos();" leading to "SettingVt".

An arrow points from a text box to the "RateKnobTurn(n = 0)" message in the sequence diagram.

Anesthesiologist turns the Rate knob without confirming – *the event is ignored*

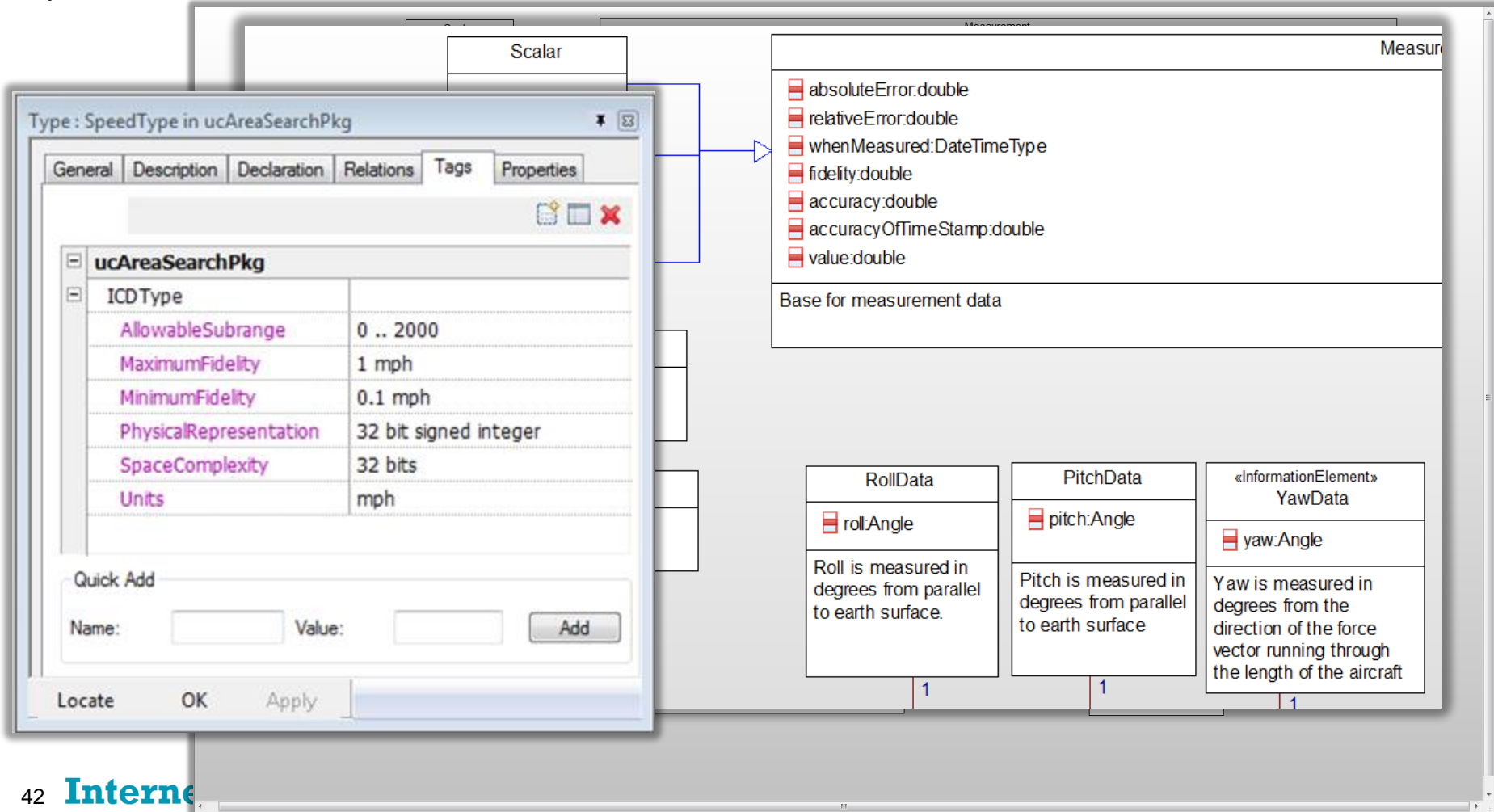


# Running the Requirements Model

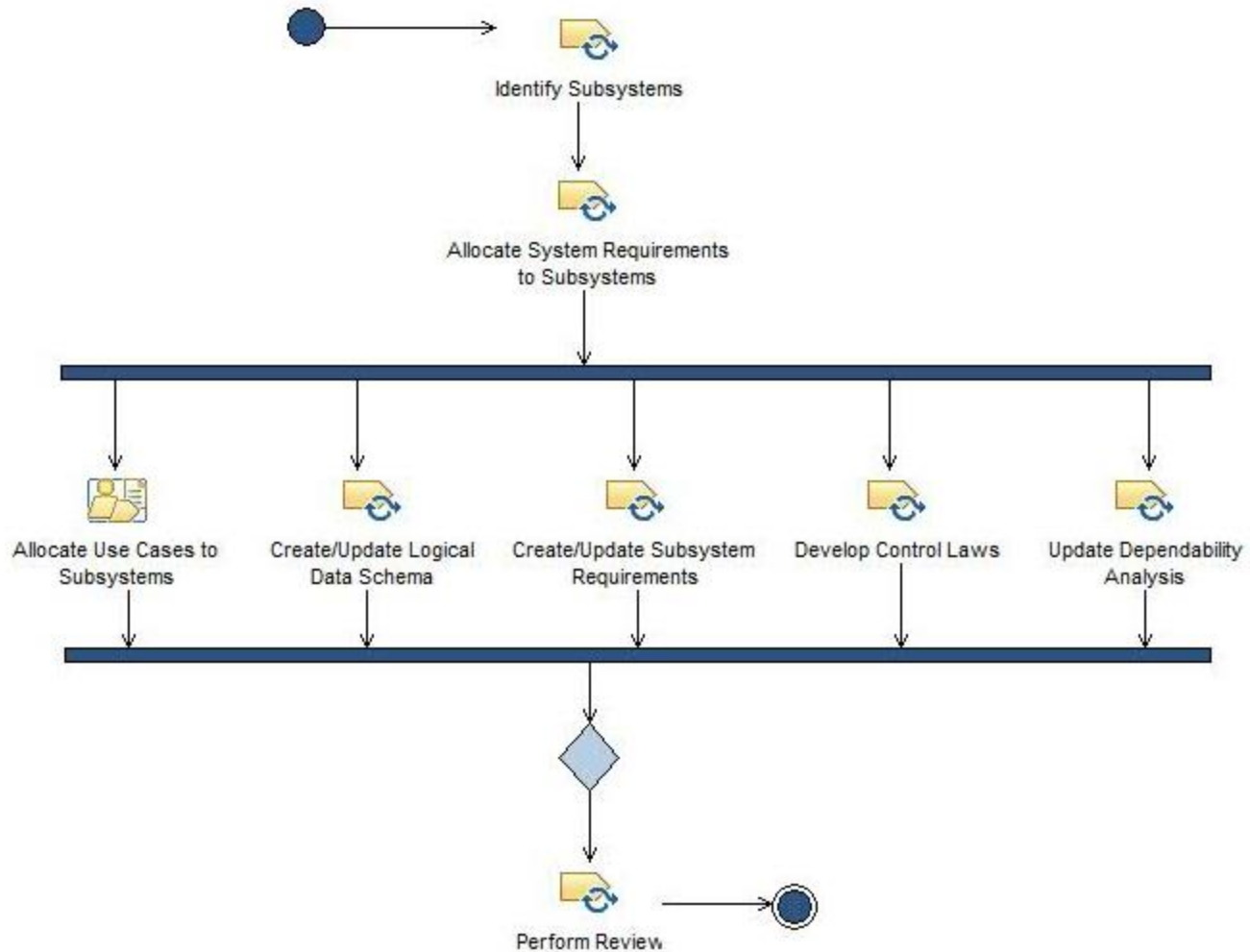


# Logical Data and Flow Schema Modeling

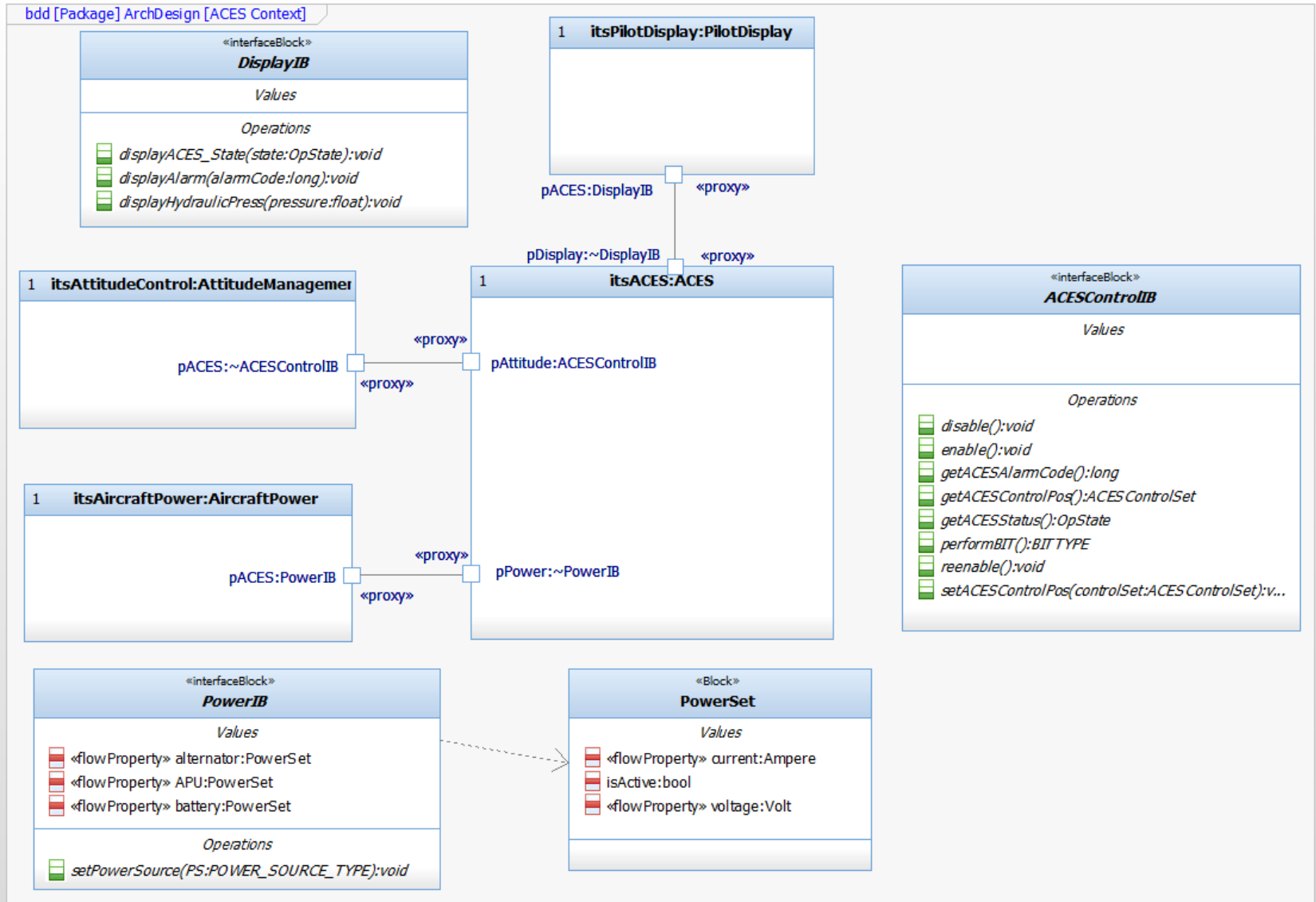
- A logical data schema identifies the logical properties of important data elements and types and the relations among such data elements and their metadata
- Although the name is “data schema” it includes physical, materiel, and energy flows specification as well



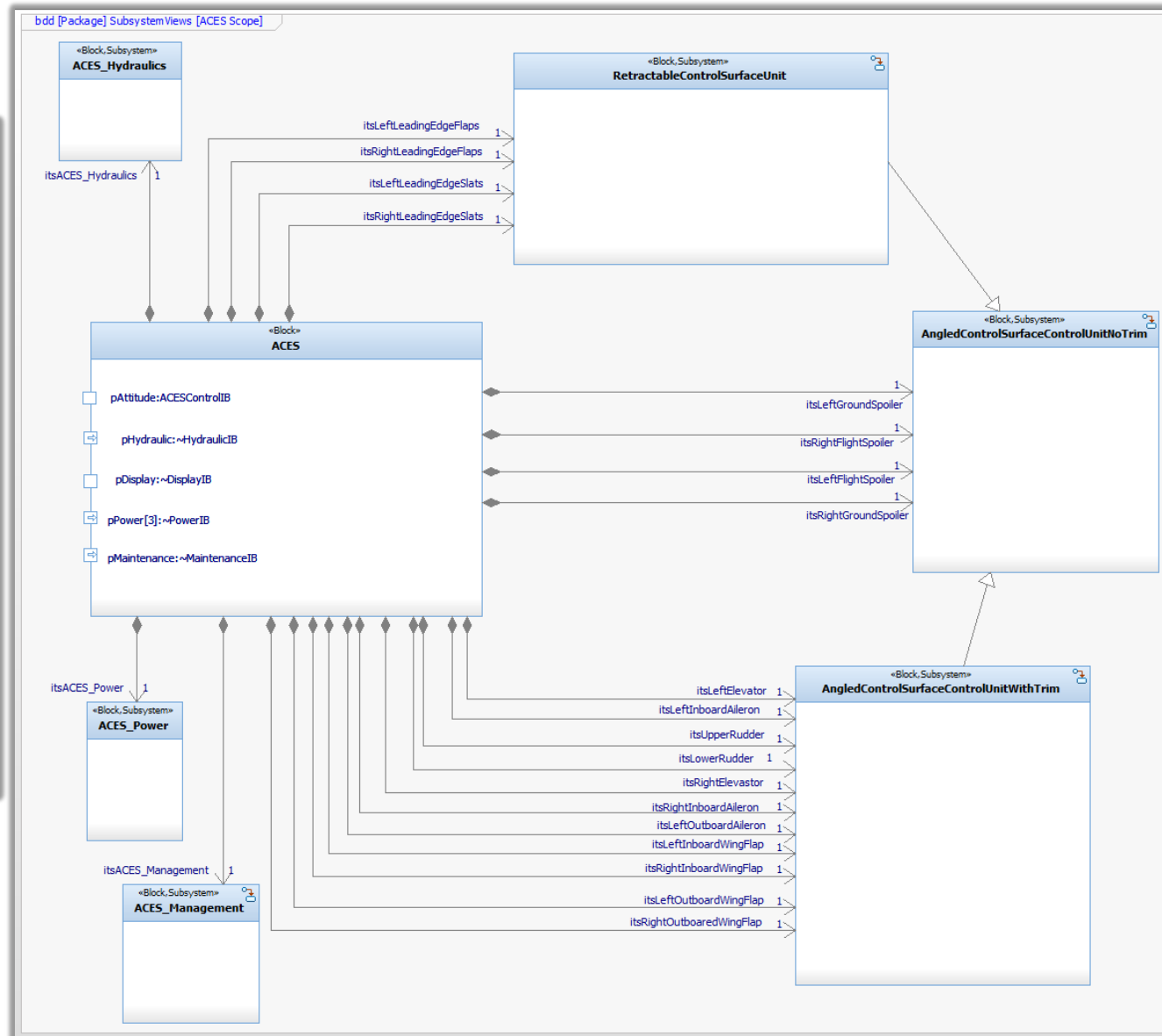
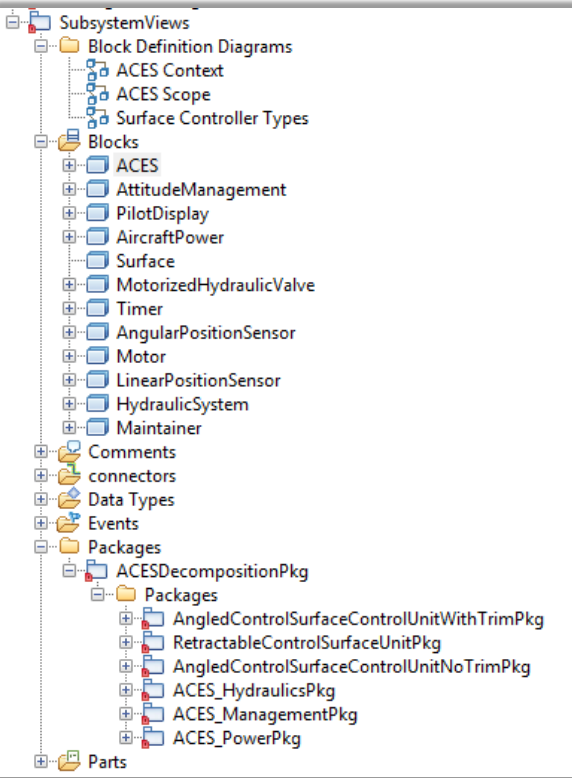
# Specifying System Architecture



# Architecture: System Context



# Architecture Structure 2



# Showing the physical messaging details for an ICD\*

- ICD tables can be constructed automatically from model data.

Here we see columns:

- Message name
- Message content field
- Content field type
- Content field metadata value, such as
  - Range
  - Format
  - Accuracy
  - Fidelity
  - Timing
  - ...

Name in cls	Name in Attr	Classifier in Attr	Name in tags	Value in tags
CBP_HydraulicStatus	status	HydraulicStatus		
CBP_Move	position	double	Numer_Of_Bytes	4
CBP_Move	surfaceID	SurfaceIDType		
CBP_Move	position	double	Format	4-byte IEEE floating point format
CBP_Move	position	double	Usage	Commanded position
CBP_MoveDone	surfaceID	SurfaceIDType	Numer_Of_Bytes	1
CBP_MoveDone	timeUsed	Interval_In_MS	Usage	Duration of movement time in ms
CBP_MoveDone	timeUsed	Interval_In_MS	Starting_Byte_Number	5
CBP_MoveDone	posAchieved	double	Format	4-byte IEEE floating point format
CBP_MoveDone	posAchieved	double	Numer_Of_Bytes	4
CBP_MoveDone	posAchieved	double	Usage	The measured position achieved in movement
CBP_MoveDone	posAchieved	double	Starting_Byte_Number	1
CBP_MoveDone	posAchieved	double	Endianism	Big
CBP_MoveDone	timeUsed	Interval_In_MS	Numer_Of_Bytes	4
CBP_MoveDone	surfaceID	SurfaceIDType	Endianism	Big
CBP_MoveDone	surfaceID	SurfaceIDType	Starting_Byte_Number	0
CBP_MoveDone	surfaceID	SurfaceIDType	Usage	ID of the referenced control surface
CBP_MoveDone	timeUsed	Interval_In_MS	Endianism	Big
CBP_PowerSource	powerSource	POWERSOURCE_TYPE		
CBP_PowerStatus	status	PowerStatus		
CBP_ReportError	when	TimeDate_Type		
CBP_ReportError	errorType	ERROR_TYPE		
CBP_ReportError	surfaceID	SurfaceIDType		
CBP_RequestConfiguration	surfaceID	SurfaceIDType		
CBP_RequestSWStatus	surfaceID	SurfaceIDType		
CBP_State	stateID	SystemOperationalState	Endianism	Big
CBP_SurfaceConfiguration	lowPos	double	Starting_Byte_Number	0
CBP_SurfaceConfiguration	lowPos	double	Usage	spec for low movement range end point. Starting_Byte is relative to start of contents.
CBP_SurfaceConfiguration	lowPos	double	Endianism	Big
CBP_SurfaceConfiguration	lowTrimPos	double	Starting_Byte_Number	8
CBP_SurfaceConfiguration	lowTrimPos	double	Usage	Spec for low end of Trim range. Number of B'Ytes is relative to start of contents.
CBP_SurfaceConfiguration	lowTrimPos	double	Format	4-byte IEEE floating point format
CBP_SurfaceConfiguration	lowTrimPos	double	Endianism	Big
CBP_SurfaceConfiguration	lowTrimPos	double	Numer_Of_Bytes	4
CBP_SurfaceConfiguration	highPos	double	Numer_Of_Bytes	4
CBP_SurfaceConfiguration	surfaceID	SurfaceIDType	Endianism	Big
CBP_SurfaceConfiguration	surfaceID	SurfaceIDType	Numer_Of_Bytes	1
CBP_SurfaceConfiguration	surfaceID	SurfaceIDType	Starting_Byte_Number	22
CBP_SurfaceConfiguration	surfaceID	SurfaceIDType	Usage	Id of the surface this configuration refers to. Number of B'Ytes is relative to start of contents.
CBP_SurfaceConfiguration	highExtPos	double	Starting_Byte_Number	20
CBP_SurfaceConfiguration	highExtPos	double	Numer_Of_Bytes	4
CBP_SurfaceConfiguration	lowPos	double	Format	4-byte IEEE floating point format
CBP_SurfaceConfiguration	highExtPos	double	Usage	Spec for high end of extension range. Number of B'Ytes is relative to start of contents.
CBP_SurfaceConfiguration	highExtPos	double	Endianism	Big
CBP_SurfaceConfiguration	highExtPos	double	Format	4-byte IEEE floating point format
CBP_SurfaceConfiguration	lowPos	double	Numer_Of_Bytes	4
CBP_SurfaceConfiguration	lowExtPos	double	Starting_Byte_Number	16
CBP_SurfaceConfiguration	lowExtPos	double	Format	4-byte IEEE floating point format
CBP_SurfaceConfiguration	lowExtPos	double	Usage	Spec for low end of extension range. Number of B'Ytes is relative to start of contents.
CBP_SurfaceConfiguration	lowExtPos	double	Numer_Of_Bytes	4
CBP_SurfaceConfiguration	lowExtPos	double	Endianism	Big
CBP_SurfaceConfiguration	highTrimPos	double	Endianism	Big
CBP_SurfaceConfiguration	highTrimPos	double	Usage	Spec for high end of trim range. Number of B'Ytes is relative to start of contents.
CBP_SurfaceConfiguration	highTrimPos	double	Format	4-byte IEEE floating point format
CBP_SurfaceConfiguration	highTrimPos	double	Numer_Of_Bytes	4

\* Interface Control Document


# Download the New Harmony aMBSE Deskbook for Free

**Harmony aMBSE Deskbook Version 1.00**  
**Agile Model-Based Systems Engineering Best Practices with IBM Rhapsody**

Bruce Powel Douglass, Ph.D.  
 Chief Evangelist  
 Global Technology Ambassador  
 IBM Internet of Things

[bruce.douglass@us.ibm.com](mailto:bruce.douglass@us.ibm.com)


**Black Edition:  
 Rhapsody Only**



© Copyright IBM Corporation 2017. All Rights Reserved Harmony aMBSE Deskbook 1

WELCOME WIZARDS TUTORIALS MODELS TIPS 'N TRICKS LINKS NEWS ABOUT

## HARMONY AMBSE DESKBOOK



*In this section you will find useful tutorials and demonstrations.*

HARMONY AMBSE DESKBOOK

**Overview**  
 From my good friend Dr. Bruce Powel Douglass - the latest and greatest Harmony aMBSE Deskbook and sample models:

[Harmony aMBSE Deskbook Version 1.pdf](#)  
[Harmony aMBSE Deskbook Models-v1.0.zip](#)

Also refer to the following extra resources:

[Harmony aMBSE Modeling Guidelines](#)  
[Systems Engineering Toolkit Handbook](#)

PREVIOUS NEXT

[merlinscave.info/Merlins\\_Cave/Tutorials/Entries/2017/9/13\\_Harmony\\_aMBSE\\_Deskbook.html](http://merlinscave.info/Merlins_Cave/Tutorials/Entries/2017/9/13_Harmony_aMBSE_Deskbook.html)

# Other Free Stuff

merlinscave.info/Merlins\_Cave/Models/Entries/2018/1/8\_Handoff\_Profile.html

merlinscave.info/Merlins\_Cave/Models/Entries/2017/3/3\_Dependability\_Analysis\_Profile.html



# References

