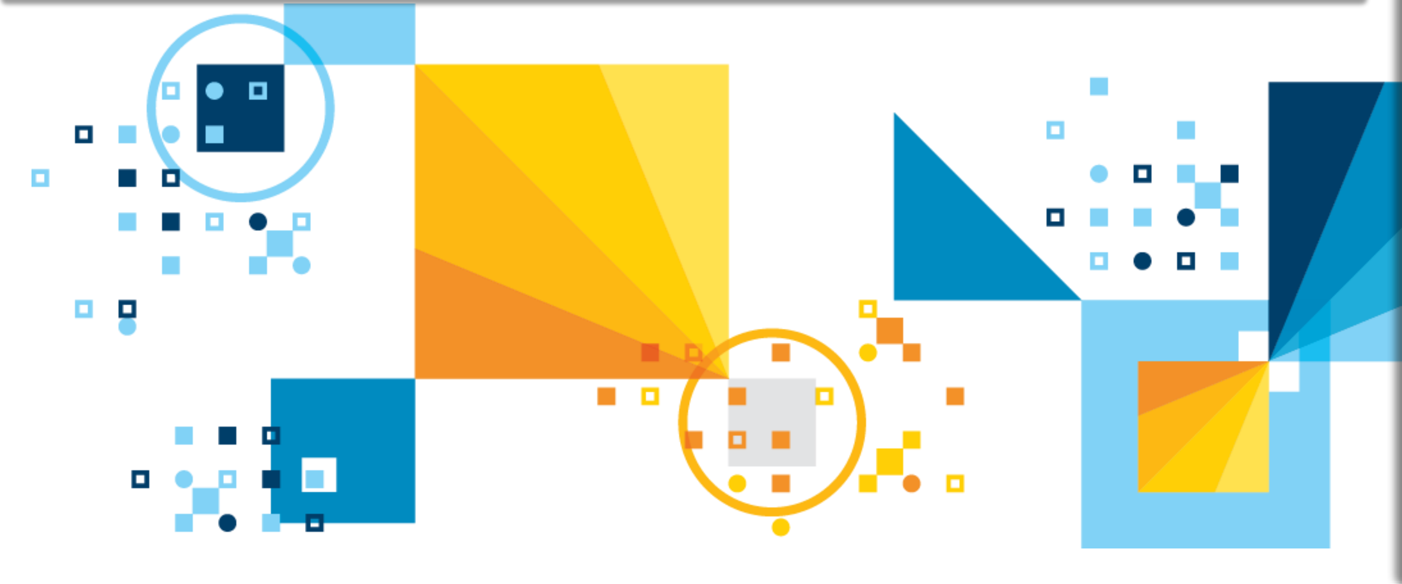
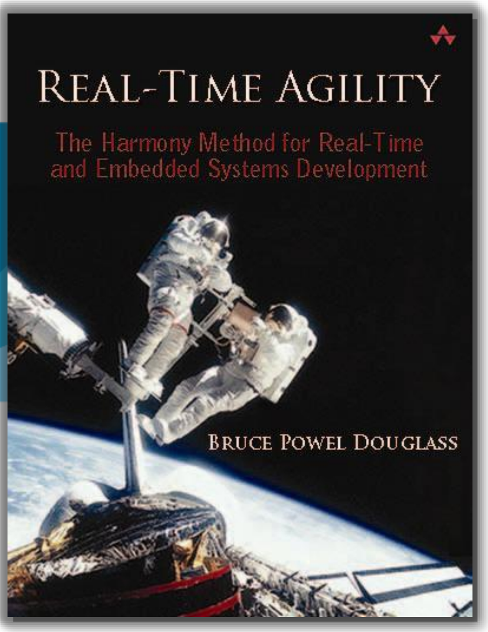
Epics, Use cases, and User Stories, Oh My!

A Guide to Requirements in an Agile World

Bruce Powel Douglass, Ph.D.
Chief Evangelist, IBM IoT
Bruce.Douglass@us.ibm.com
Twitter: @IronmanBruce
www.bruce-douglass.com





REAL-TIME AGILITY

The Harmony Method for Real-Time and Embedded Systems Development

BRUCE POWEL DOUGLASS

Epics, use cases, and user stories, oh my

Months

(multiple iterations)

EPIC

An **epic** is a coherent set of features, use cases, and user stories at a strategic level.

Some epics are **functional** in nature, decomposing to use cases and user stories while others are **technical** in nature, decomposing to technical work items.

Weeks

(single iteration)

Feature

A **feature** is a chunk of functionality that delivers business value. Features can include additions or changes to existing functionality.

Use Case

A **use case** is made up of a set of possible sequences of interactions between systems and actors (including users) in a particular environment and related to a particular goal.

Days

(single iteration)

Work Item

A small unit of work in the product backlog, such as a user story or spike

User Story

A kind of requirement, a user story depicts a simple interaction with the product to achieve a goal.

Spike

A work item that is meant to reduce some risk, such as a technical, project, or business risk.

Hours

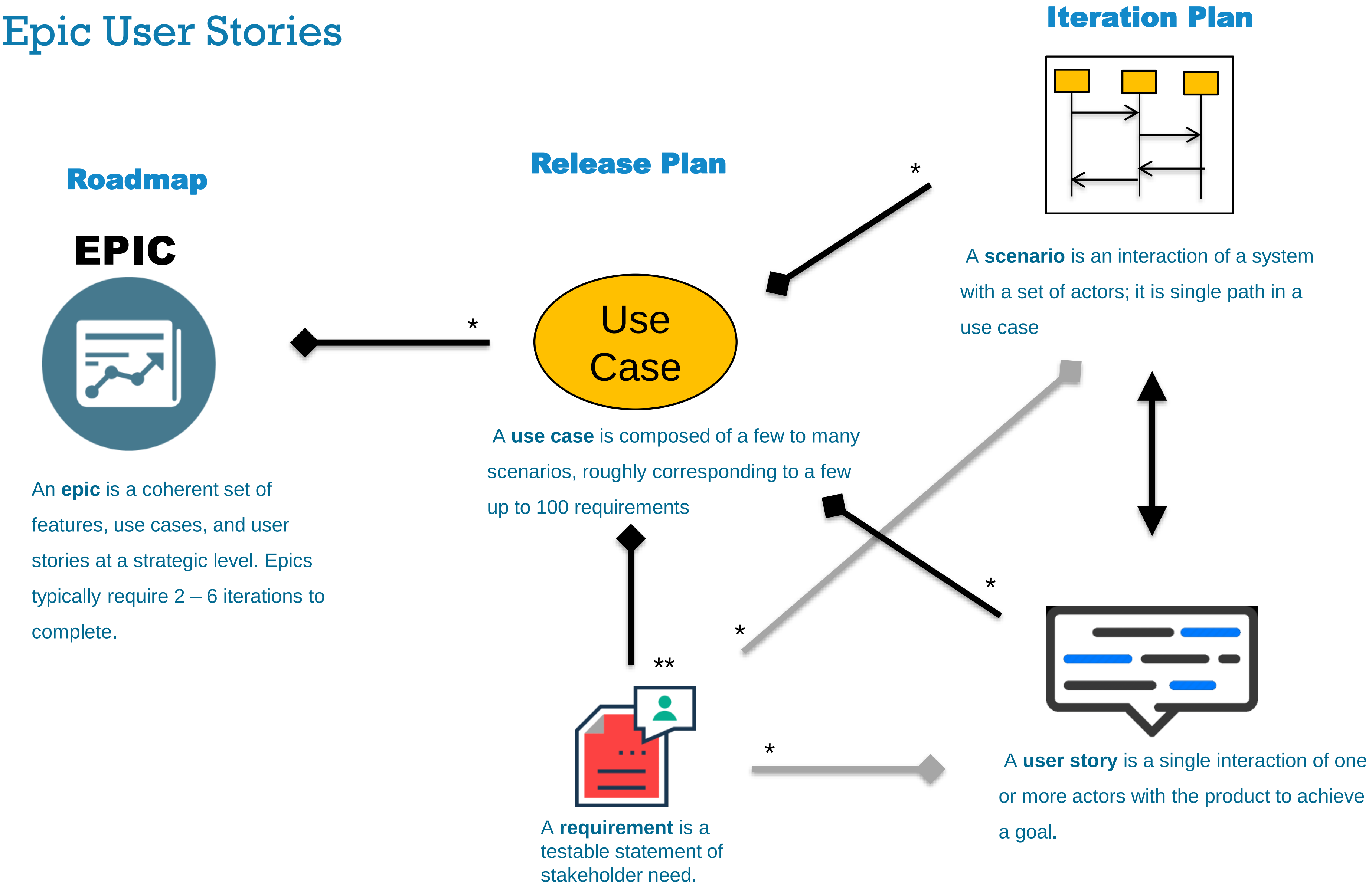
(single nanocycle)

A **requirement** is a statement of what the system must do or a constraint.

Task

The technical work that a development team performs in order to complete a product backlog item.

Epic User Stories



Roadmap

Release Plan

Iteration Plan

An **epic** is a coherent set of features, use cases, and user stories at a strategic level. Epics typically require 2 – 6 iterations to complete.

A **use case** is composed of a few to many scenarios, roughly corresponding to a few up to 100 requirements

A **scenario** is an interaction of a system with a set of actors; it is single path in a use case

A **requirement** is a testable statement of stakeholder need.

A **user story** is a single interaction of one or more actors with the product to achieve a goal.

Epic Use Cases Example

Epic name: Surgical ventilation

Goal: Establish company in the high-end surgical medical device market. Also reduce the number of different ventilation architecture platforms supported by the company by creating a customizable device.

Purpose: Provide ventilation which is highly reliable, easy to configure, easy to maintain, and interacts with the

Primary needs addresses:

- Simplify set up time
- Provide highly reliable ventilation even during patient episodes and loss of power
- Tie in reporting to hospital information network (HIN)

Target Group: Surgical anesthesiologist

Products: Mixologist series of ventilators, Merlin ventil

Use Case

Use case name: Mix Gases

Purpose: Allow accurate mixing of gases for delivery

Description: Provides the well-controlled mixing of up to 6 different gases from wall supplies

Actors: Gas supply, breathing circuit, physician

Pre-conditions: Gas is available, system is connected to breathing circuit

Post-conditions: mixed gas is delivered at the percentages and rates commanded

Constraints: total output flow is limits to 100 L/min

Risks: None

Use Case Points: 10

Use Case

Use case name: Monitor device health

Purpose: Identify system failures that could lead to patient episode

Description: The system monitors actuators and sensors to ensure that they are operating properly.

Actors: physician

Pre-conditions: system is on and has initial POST

Post-conditions: Errors are logged and reported to attending physician

Constraints: none

Risks: It may not be possible to identify gas leaks

Use Case Points: 6

Use Case

Use case name: CO2 Scavenging

Purpose: Remove CO2 from the expiratory gas

Description: Removes almost all expired CO2 from expired gas but alarms if CO2 exceeds threshold.

Actors: Breathing circuit

Pre-conditions: Connected to the breathing circuit

Post-conditions: Removes CO2 or alerts attending physician

Constraints: expiratory flow is limited to 100 L/min max

Risks: Unsure if we can meet the target CO2 concentration at high flow rate

Use Case Points: 4

Use Case

Use case name: Monitor patient parameters

Purpose: Provide the physician with timely information about patient health

Description: Monitors and reports SpO2, O2 input flow, O2 input percentage, heart rate, and NIBP

Actors: Physician

Pre-conditions: system is on and has initial POST

Post-conditions: patient data displayed in a timely fashion

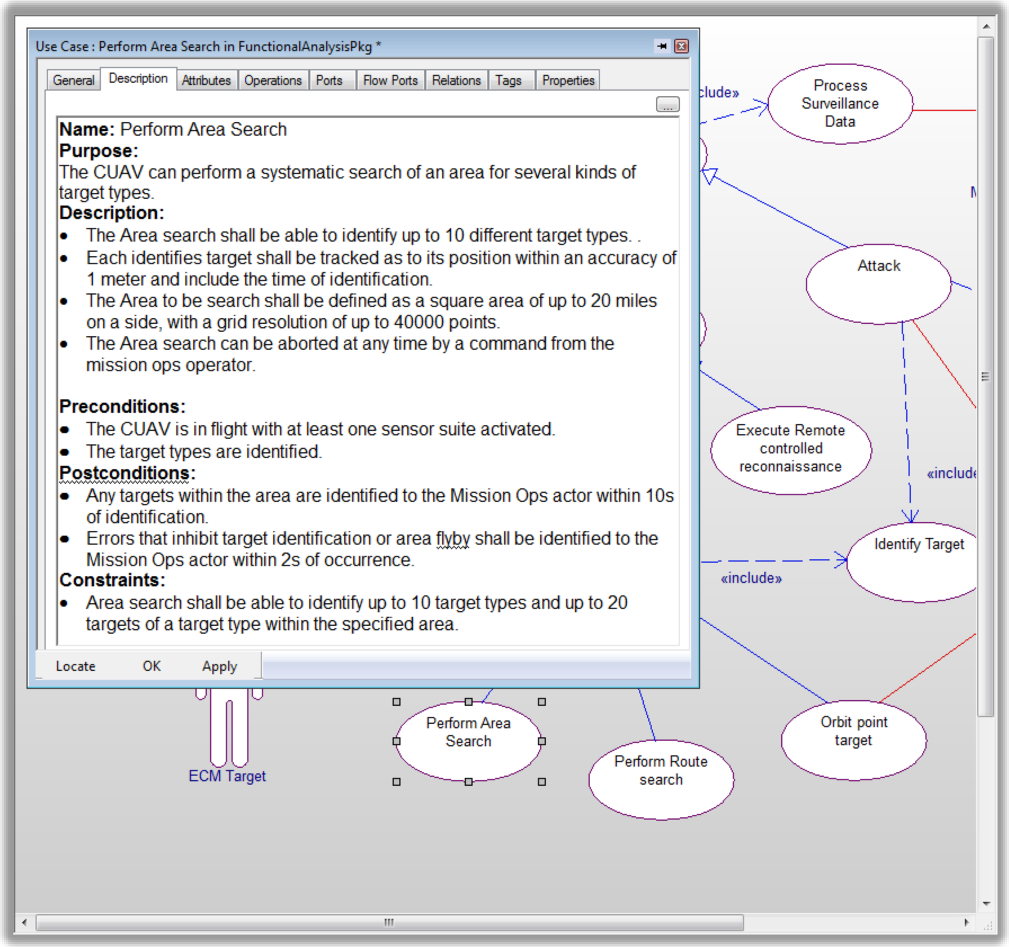
Constraints: none

Risks: Our current SpO2 OEM vendor is going out of business and it isn't clear there is a **viable** replacement

Use Case Points: 7

For Every Use Case, a Description (Minispec) …

- Use Case Description Structure
  - Name
  - Purpose
    - Identifies the goals of the capability and its value to the stakeholders
  - Description
    - Summarizes the control and data transformations that the use case specifies
  - Preconditions
    - What is true prior to the execution of the capability?
  - Postconditions
    - What does the system guarantee to be true after the execution of the use case?
  - Invariants
    - What relevant conditions are assumed to be always true?
  - Constraints
    - Additional QoS requirements or other rules or limitations for the use case
  - Use case points



Name: Perform Area Search
Purpose:
The CUAV can perform a systematic search of an area for several kinds of target types.
Description:
- The Area search shall be able to identify up to 10 different target types. .
- Each identifies target shall be tracked as to its position within an accuracy of 1 meter and include the time of identification.
- The Area to be search shall be defined as a square area of up to 20 miles on a side, with a grid resolution of up to 40000 points.
- The Area search can be aborted at any time by a command from the mission ops operator.

Preconditions:
- The CUAV is in flight with at least one sensor suite activated.
- The target types are identified.

Postconditions:
- Any targets within the area are identified to the Mission Ops actor within 10s of identification.
- Errors that inhibit target identification or area flyby shall be identified to the Mission Ops actor within 2s of occurrence.

Constraints:
- Area search shall be able to identify up to 10 target types and up to 20 targets of a target type within the specified area.

What’s a use case?

01 **It’s an operational capability of a system**

Stated from the user or actor’s perspective

02 **A use case organizes requirements**

- Normally 10-100 textual requirements
- Normally 3-20 user stories or scenarios
- Normally a few to a few dozen use cases per system

03 **May group stakeholder, system, subsystems or software requirements**

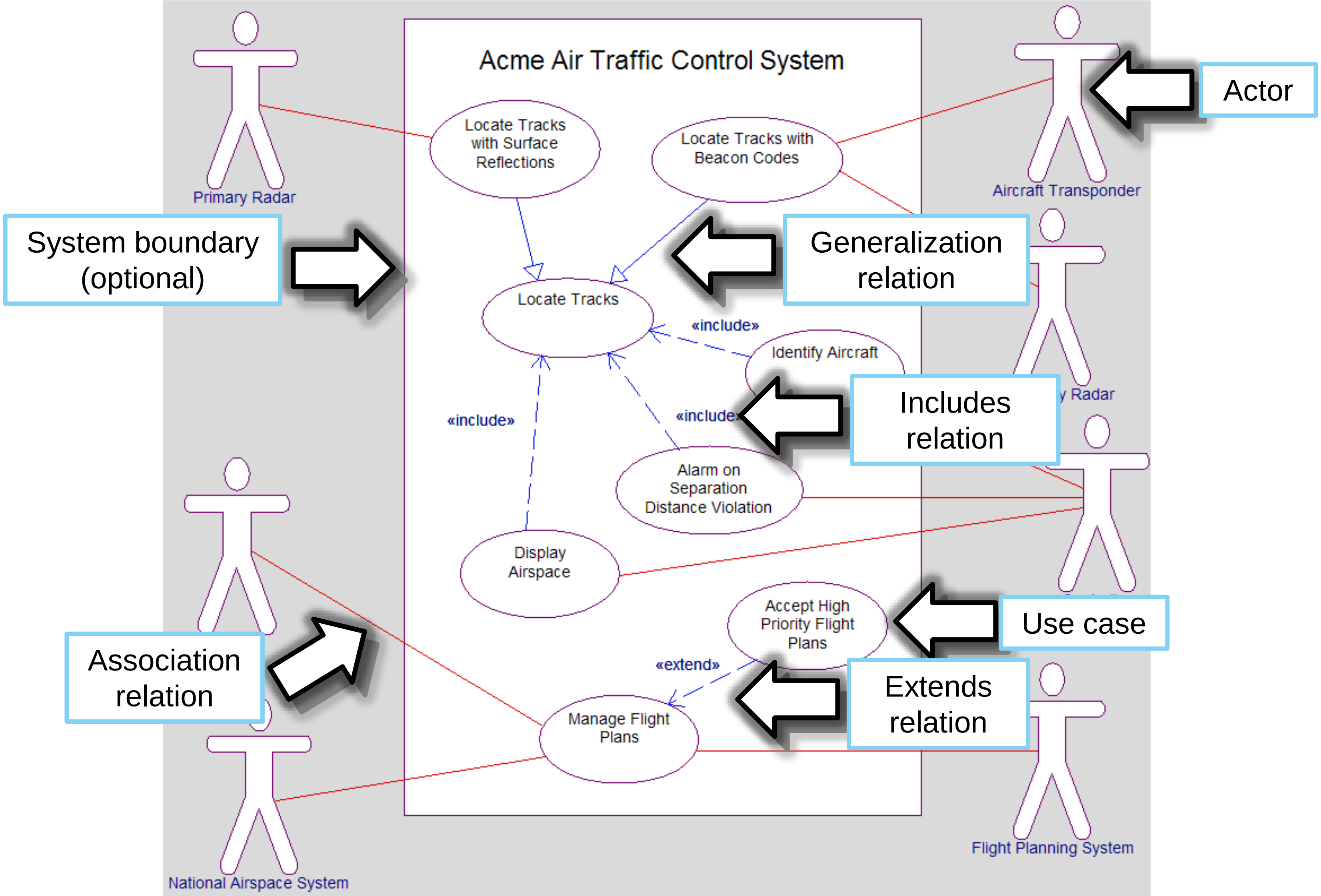
04 **Returns a result visible to one or more actors**

05 **Does not reveal or imply internal structure of the system**

06 **Is independent of other use cases and may be concurrent with them**

07 **May be constrained with various QoS parameters**

Use Case Syntax



Actor

System boundary (optional)

Generalization relation

Includes relation

Use case

Association relation

Extends relation

Use Case Recommendations

Use short verb or verb-phrase names

Not nouns! “Move Control Surface” not “Surface Controller”

Name from problem domain vocabulary

Not solution vocabulary! “Apply Braking” not “Apply Hydraulic Disk Pressure”

Give each use case a short specification

Describe the use case in meaningful, stakeholder terms

Aspects of use cases

- Identify services in/out
  - ex. Heat water(set temp), report water temp(measured temp)
- Identify data / flows in/out
  - ex. Set temperature, measured temperature, alarm limit temperature
- Identify control/data/flow transformation
  - ex. cold water in → hot water out
- Identify levels of **fidelity** (precision of the input) and **accuracy** (precision of the output) of the use case
  - ex. temperature set in units of 0.5C, accuracy managed to 0.1C
- Specify required performance, reliability, safety, security, etc
  - ex. Water must be heated to set temperature within 30s

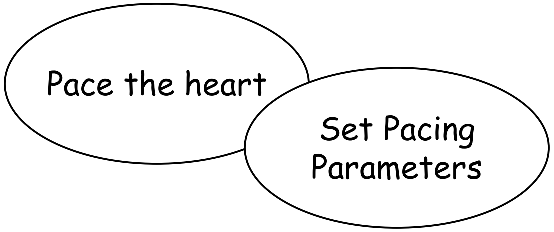
Actors

- Identify their goals and objectives for the use case
- Identify which services they need from or will provide to the system while executing the use case
- Include data and flows in/out
- What transformations are expected?

Use Case Size

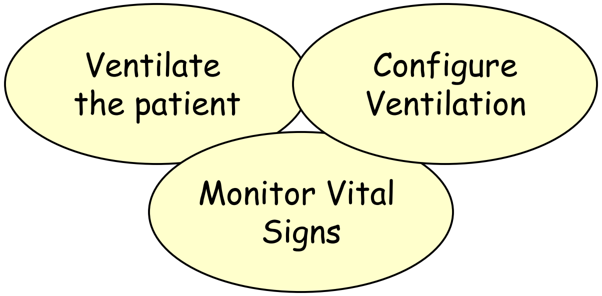
Small system

- A few hundred requirements
- Normally 6 – 24 use cases



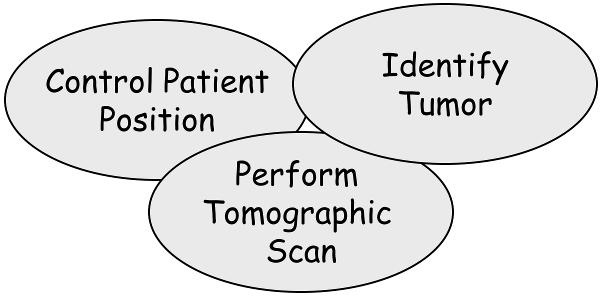
Medium system

- A few hundred to a few thousand requirements
- Normally 10-70 use cases



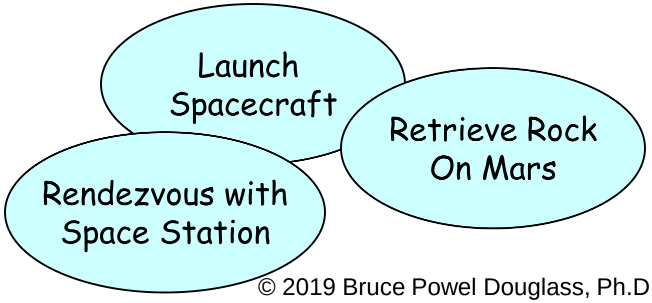
Large system

- Several thousands of requirements
- Normally 6-24 "high-level" use cases (may be thought of as **epics**)
  - Decomposed 1-2 levels
  - Between 70 – 500 use cases total

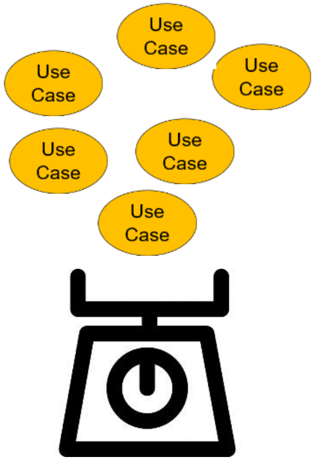


Huge system

- Tens of thousands of requirements
- Normally 6-24 "high-level" use cases (may be thought of as **epics**
  - Decomposed 3-5 levels
  - Up to 1000 use cases total



Estimating use case size



01

Purpose

Size/work effort estimation is important because it allows us to allocate work to iterations with some confidence of being able to achieve the work.

02

Description

Use case points are a common agile technique using approximate relative, rather than absolute sizing.

While estimating use case points, we assign a point value to each use case. Relative values are more important than the raw values. A 4-pt use case would take 4 times most effort to create than a 1-pt use case.

Alternatively, an absolute measure, such as work hours can use used as an estimate.

03

Hints

Use case points are not generally a continuous range. It is common to use doubled numbers such as 0, 1, 2, 3, 5, 8, 13, 20, 40 and 100 or Fibonacci sequence: 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144

04

Procedure

1. List all use cases, risk spikes, and technical work items
2. Arrange them in order of effort, from smallest to largest
3. Sort them by effort required until consensus is reached on the ordering
4. That done, then assign point values to the use cases, starting at the bottom (smallest)
5. As you move up the list, size each based on its relative size of the one below

Use Case Analysis Recommendations

Purpose: the purpose of use case analysis is, ultimately, to ensure the requirements are clear, complete, understood, and correct.

RECOMMENDED

✓ Each use case should have multiple user stories or scenarios

3 – 25 user stories or scenarios is common

✓ Initially focus on normal flows but don’t forget …

error, exception, safety, reliability, security, and invariant-violation scenarios (“rainy day scenarios”)

There are generally more rainy day cases than sunny day cases

✓ Avoid adding into design detail

Focus on flows between the actors and the system

✓ Each scenario message should represent one or more requirements

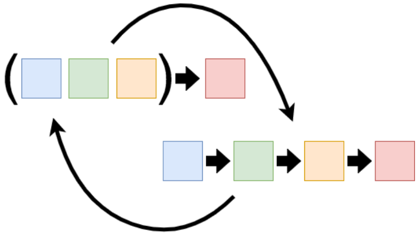
If you trace requirements to the user stories / scenarios, then you can easily do a ‘coverage analysis’ to be sure that no requirements are forgotten

✓ Computable models are a great way to validate requirements

Building a formal, computable requirements model assists in uncovering missing, incomplete, ambiguous and conflicting requirements. This is normally done with activity or state models.

Understanding Functional Flow

OPTIONAL



01

Purpose

Modeling the functional flow is a useful tool to understand complex use cases before creating user stories, especially to make user that features and options are not neglected

02

Description

Activity diagrams or state machines can be used to represent, in a graphical way, the functional flow of the system.

- Activity diagrams are preferable to represent
  - Continuous flows in cyberphysical systems
  - Algorithmic flow
- State machines are preferable to represent
  - Use cases where behavior is highly dependent on 'mode'
  - When different behavior occurs in different circumstances

03

Procedure

1. Create the flows in the selected representational format
2. Derive user stories and scenarios are paths within the flow specifications

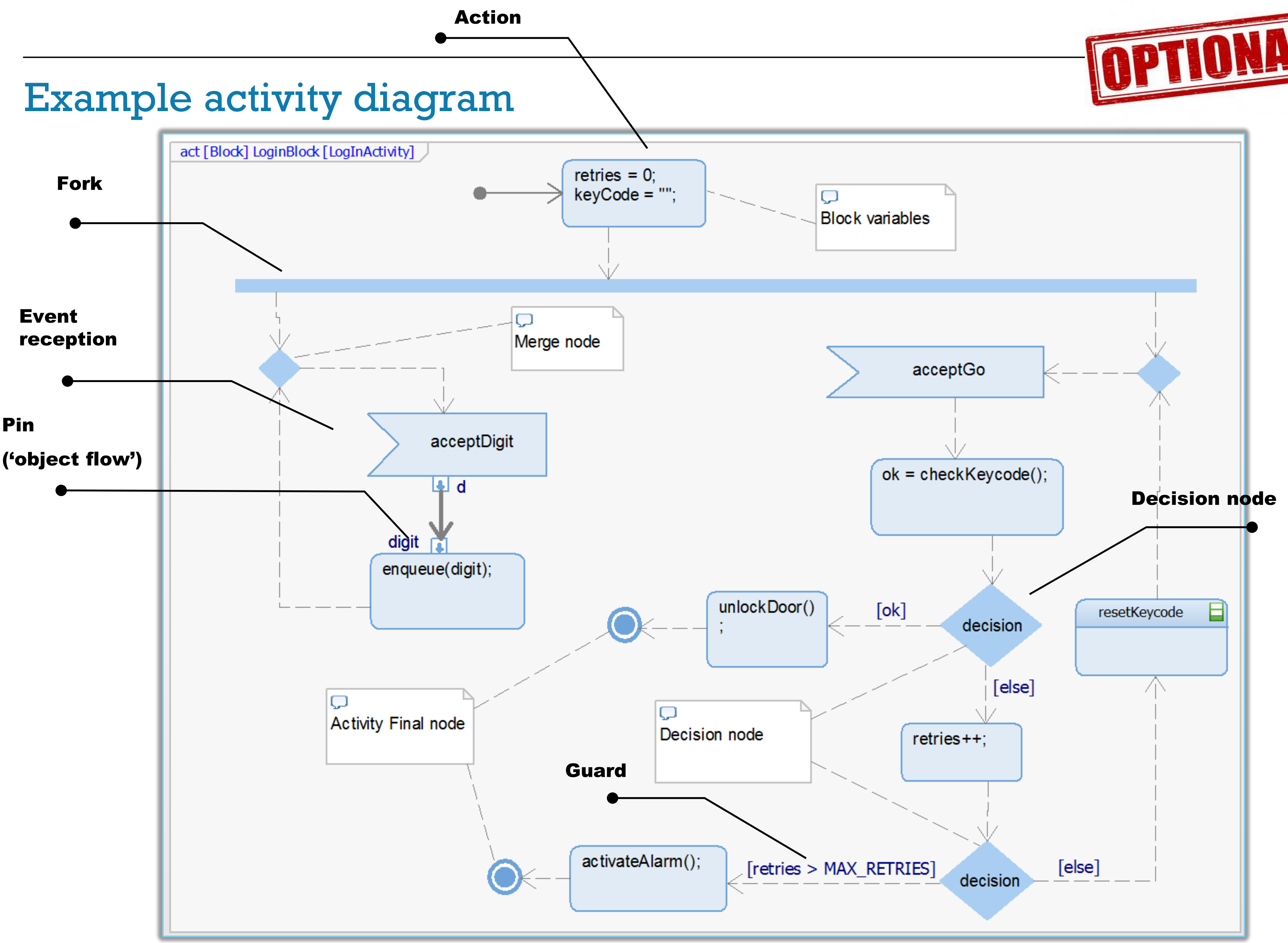
04

Hints

1. This is best done with UML/SysML or similar tools
2. Rigorously defined flows can be made executable to allow the exploration of use case implications and 'what if' scenarios

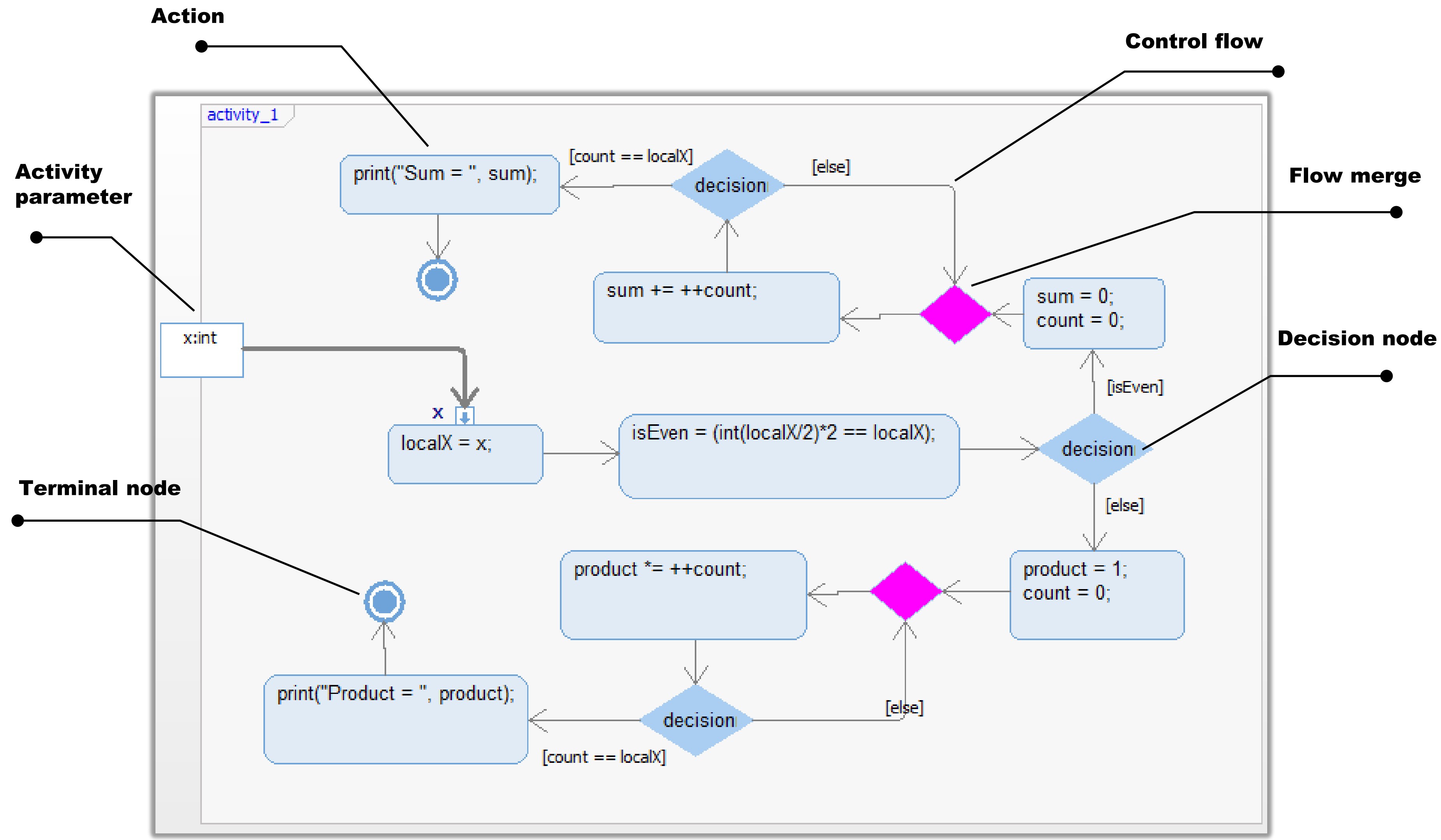
OPTIONAL

Example activity diagram



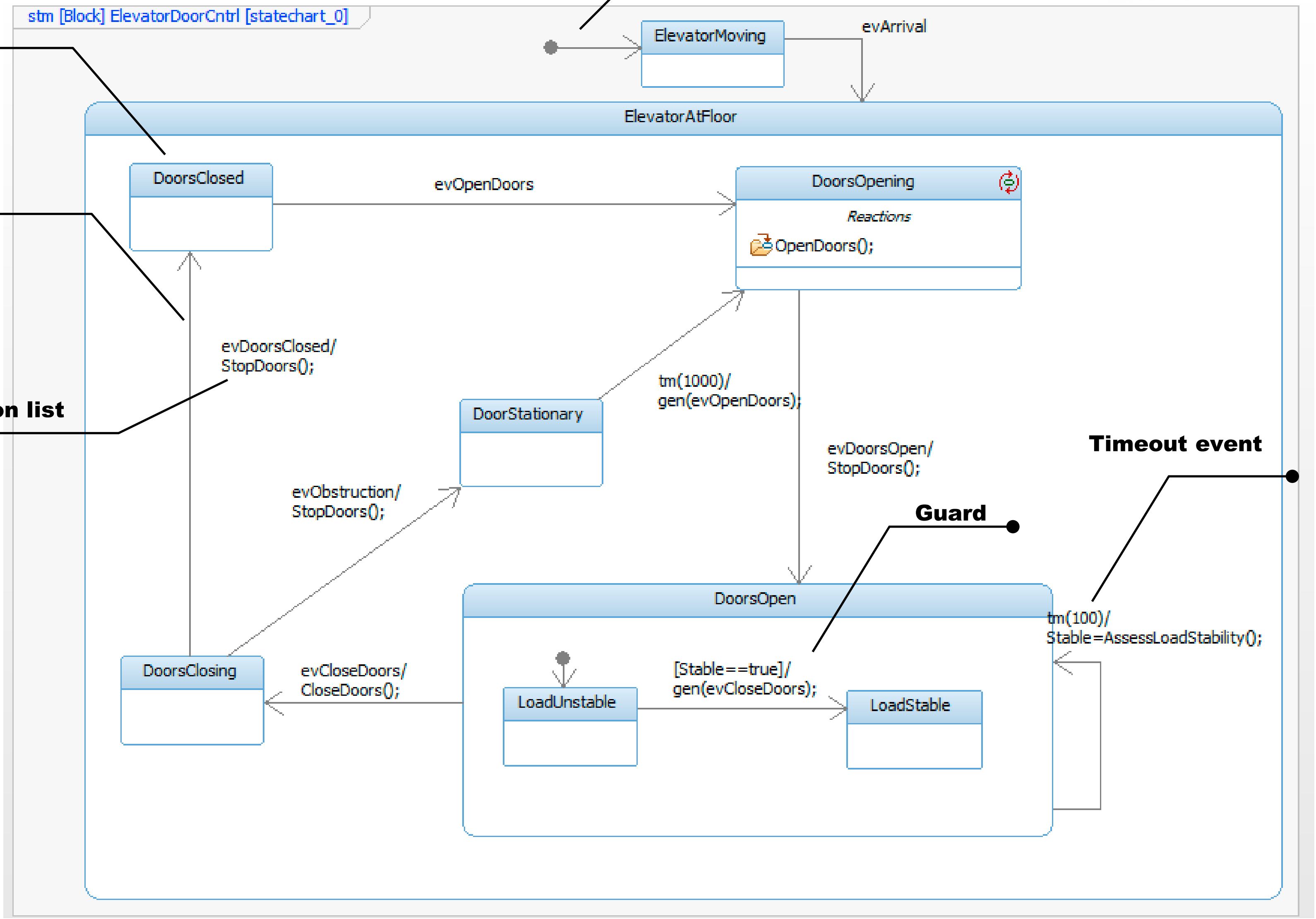
Activity diagram example

OPTIONAL



OPTIONAL

State machine example



Initial transition

State

Transition

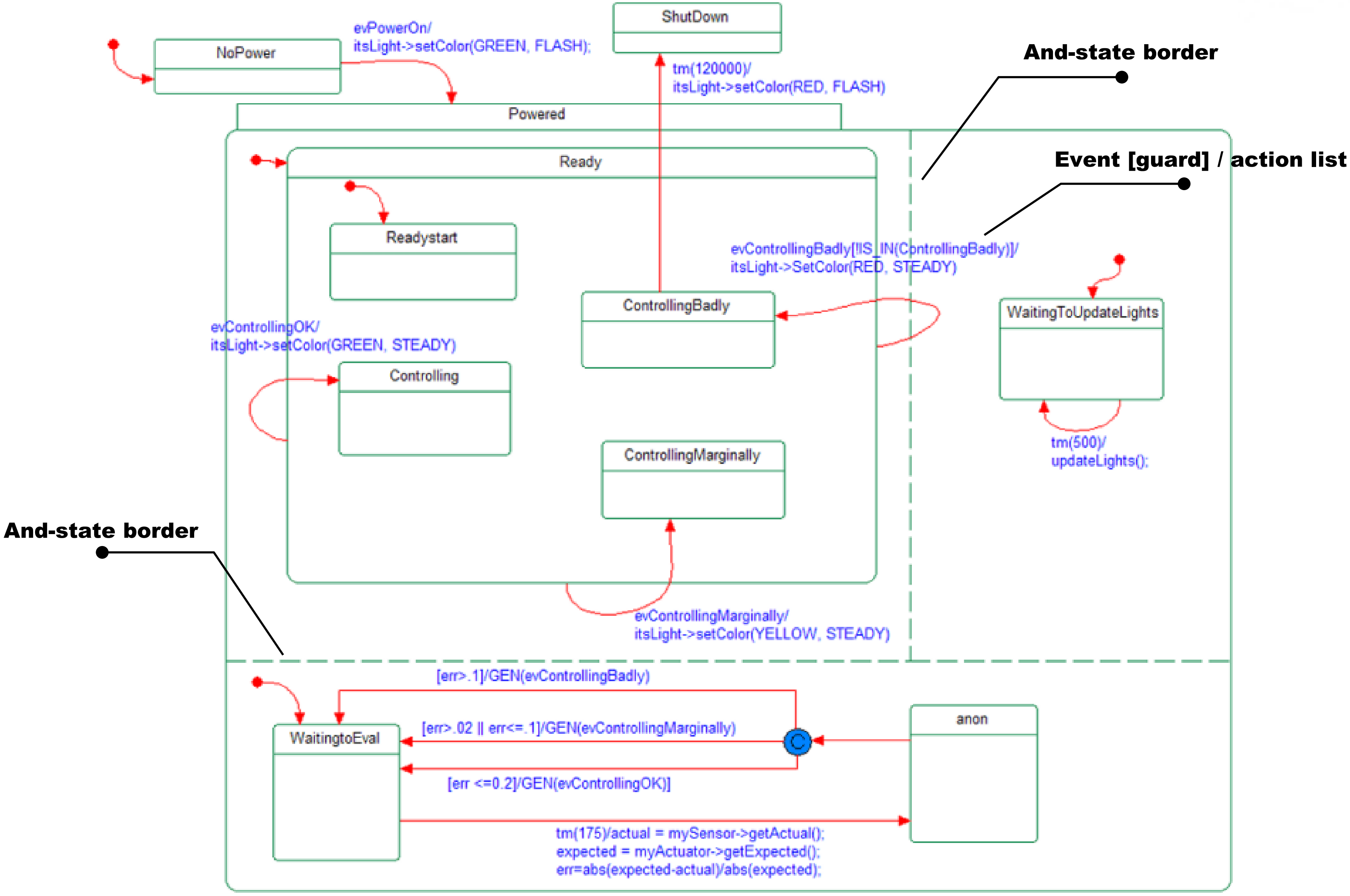
Event / action list

Timeout event

Guard

OPTIONAL

State machine example

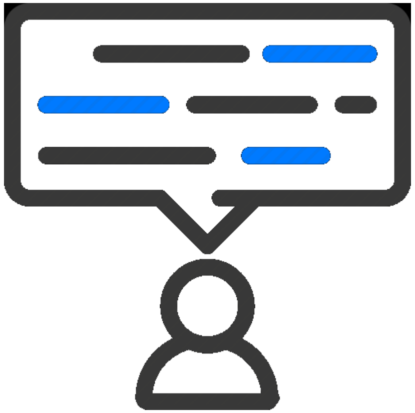

NoPower
evPowerOn/
itsLight->setColor(GREEN, FLASH);
ShutDown
tm(120000)/
itsLight->setColor(RED, FLASH)
Powered
Ready
Readystart
ControllingBadly
evControllingBadly[!IS_IN(ControllingBadly)]/
itsLight->SetColor(RED, STEADY)
evControllingOK/
itsLight->setColor(GREEN, STEADY)
Controlling
ControllingMarginally
evControllingMarginally/
itsLight->setColor(YELLOW, STEADY)
WaitingToUpdateLights
tm(500)/
updateLights();
[err>.1]/GEN(evControllingBadly)
WaitingtoEval
[err>.02 || err<=.1]/GEN(evControllingMarginally)
C
anon
[err <=0.2]/GEN(evControllingOK)]
tm(175)/actual = mySensor->getActual();
expected = myActuator->getExpected();
err=abs(expected-actual)/abs(expected);

And-state border

Event [guard] / action list

And-state border

User Story



Canonical form for a user story:

"As a " <user> "I want" <feature> "so that " <reason>

01

Purpose

Characterize the stakeholder need through understanding the necessary interactions of the system with its environment

02

Description

A **user story** is a tool used in Agile software development to capture a description of a software feature from an end-user perspective. The user story describes the type of user, what they want and why. A user story helps to create a simplified description of a requirement

03

Detailed procedure

A user story is often captured in a canonical form

As a <role>, I want <feature> so that <reason>

This leads to the discovery and development of system and software requirements.

User stories are equivalent to scenarios which may be detailed using UML sequence diagrams.

04

Examples of user stories

- As a **pilot**, I want to **control the rudder of the aircraft using foot pedals** so that I can **set the yaw of the aircraft**.
- As a **power supply**, I want to **provide constant power at +15V +/- 0.1V with a current of 5 amps +/- 0.05 amps** to **power the rotor**.
- As a **navigation system**, I want to **report the position of the aircraft in 3 dimensions with an accuracy of +/- 1 m every 0.5s** so that I can **fly to the destination.**

Use case or user story?

User Story

Use Case

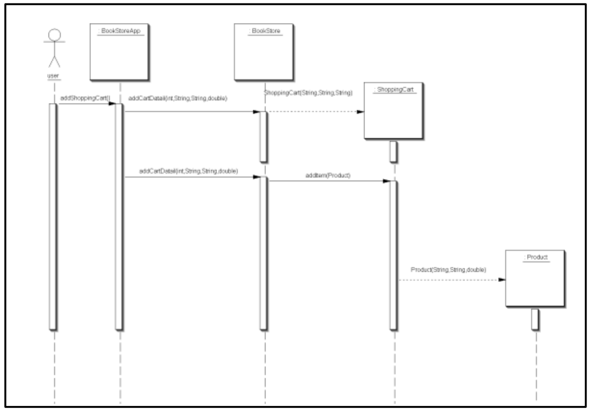
*

A use case is composed of a few to many scenarios, roughly corresponding to a few up to 100 requirements

A user story is a single interaction of one or more actors with the product to achieve a goal.

*

Scenarios are approximately equivalent to user stories

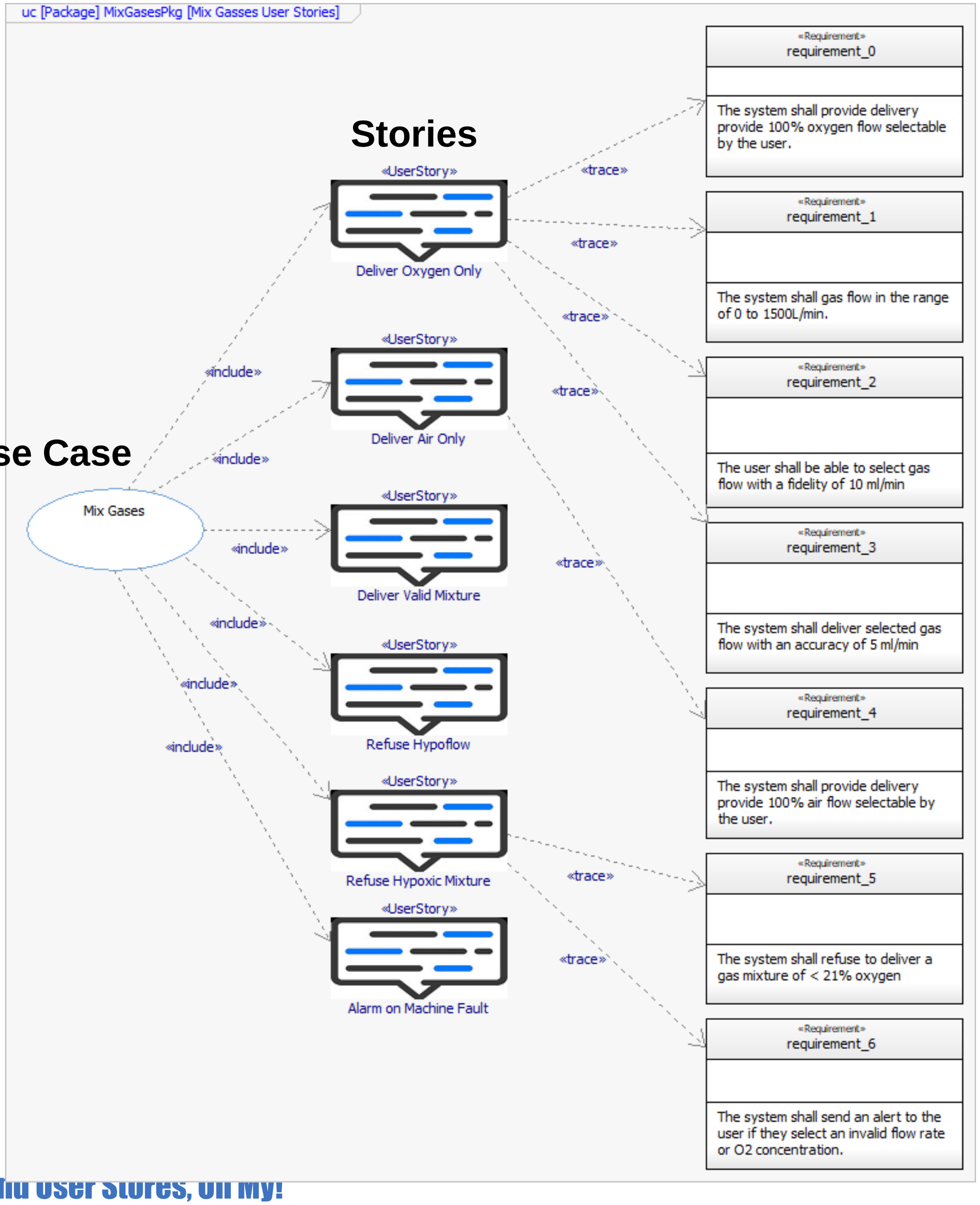

user
: BookStoreApp
: BookStore
: ShoppingCart
: Product
addShoppingCart()
addCartDetail(int,String,String,double)
ShoppingCart(String,String,String)
addCartDetail(int,String,String,double)
addItem(Product)
Product(String,String,double)

A scenario is an interaction of a system with a set of actors; it is single path in a use case

Use Cases to Stories to Requirements

Stories

Use Case



requirement_0
The system shall provide delivery provide 100% oxygen flow selectable by the user.

requirement_1
The system shall gas flow in the range of 0 to 1500L/min.

requirement_2
The user shall be able to select gas flow with a fidelity of 10 ml/min

requirement_3
The system shall deliver selected gas flow with an accuracy of 5 ml/min

requirement_4
The system shall provide delivery provide 100% air flow selectable by the user.

requirement_5
The system shall refuse to deliver a gas mixture of < 21% oxygen

requirement_6
The system shall send an alert to the user if they select an invalid flow rate or O2 concentration.

Requirements

User story or scenario?

User Story

They often are cast in a standardized form:

As a <role>, I want <feature> so that <reason>

For example,

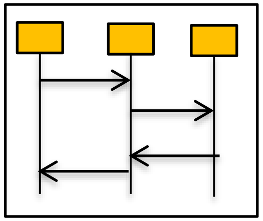
As a **pilot**, I **want the pedal to control the rudder in a range of -30 to +30 degrees** so **that I can steer left or right**.

- Simple
- No special tools needs
- Easy to review with stakeholders

- It may be difficult or impossible to write a user story for a complex interaction
- It is difficult to state qualities of service within a user story

Scenario

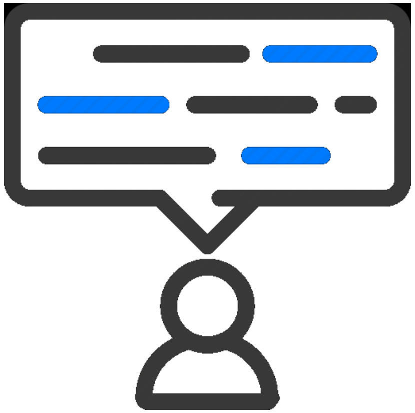
Supported in UML, the show the user story as a set of message interactions and services among a set of roles, once of which is the system



- Can represent far more complex interactions than textual user stores
- Supported by many UML/SysML tools
- Can support model-based trace to requirements and design elements with summary table generation
- QoS requirements can be added as annotations and constraints

- Requires a tool (although simple drawing tools can be used)
- A little bit more complex to read and understand
- Not in ‘natural language’

Developing user stories from use cases



01

Purpose

Decompose the use case into small pieces of functionality that meet a specific system-user need that can be delivered in a few days of work.

02

Description

A user story is often captured in a canonical form

As a <role>, I want <feature> so that <reason>

This leads to the discovery and development of system and software requirements.

User stories may also be detailed using UML sequence diagrams.

03

Procedure

1. Take the use case and identify different user-system interactions
2. Cast each as a user story sentence or paragraph
3. Alternatively, if you've done functional flow diagrams for the use case, capture each separate path as a user story or scenario
4. Review to make sure
    1. Each flow – and each requirement – is represented in at least one user story or scenario
    2. Exceptions, misuse, and error cases are handled

04

Hints

1. If you build an executable functional flow, then scenarios can be captured by executing the various branch cases. In the Rhapsody UML tool, these are called "animated sequence diagrams"

User Story Guidelines

1 **Focus on the users**
We want to avoid referencing or discussing design, but instead focus on achieving user goals

2 **Use personae to discover the stories**
Most systems have many stakeholders with needs to be met. Represent all the relevant ones

3 **Develop user stories collaboratively**
User stores are a lightweight analytic technique and can foster good discussions amongst the product owner and the developer.

4 **Keep stories simple and precise**
The story should be easy to understand yet fit the need. If a story is complex, break it up into multiple stories.

5 **Start with Epics or Use Cases**
User stories are small, finely grained things, while epics and use cases provide the larger context

6 **Refine stories**
As your understanding deepens and improves, this should be reflected in the stories

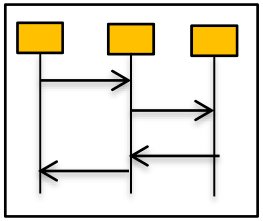
7 **Add acceptance criteria**
Acceptance criteria complete the narrative with clear statements about what it means to actually meet the need

8 **Paper cards are a great way to do lightweight user stories**
Post-it notes work too. Cards can be easily sorted and grouped. And they're cheap.

9 **Don't solely rely on user stories**
Not all needs are easily captured in user stories, such as workflows, visual interactions, safety, reliability, security and other qualities of services

10 **Keep your user stories visible and accessible**
The should foster communication and collaboration

Developing scenarios from use cases



01

Purpose

Decompose the use case into small pieces of functionality that meet a specific system-user need that can be delivered in a few days of work.

02

Description

A scenario is usually modeled in a sequence diagram. Vertical lines represent actors, the system or the use case; flows between the lifelines indicate messages or interactions

03

Procedure

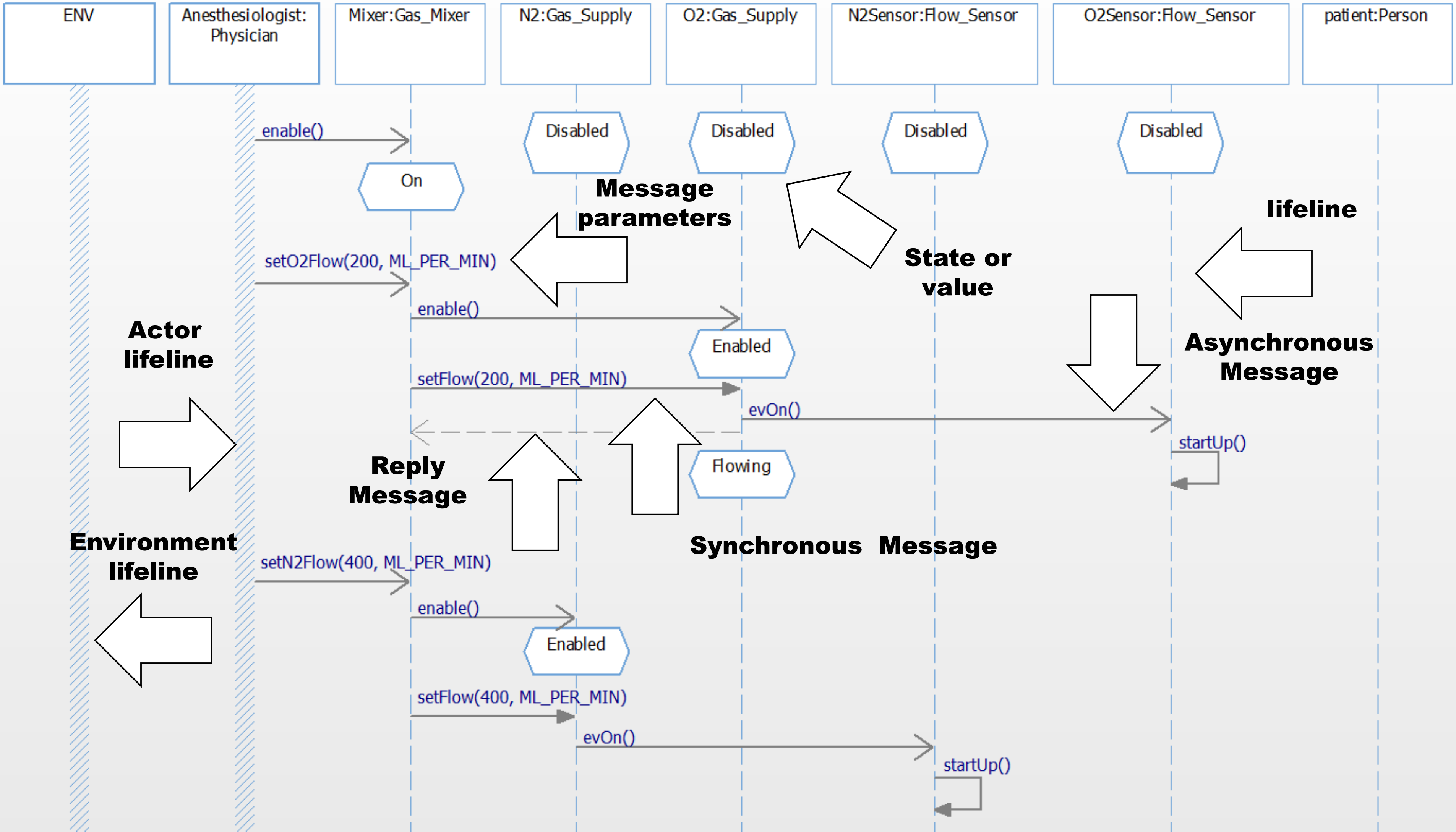
1. If you've done a functional flow, walk through the branch points and represent a complete path as a single scenario on one sequence diagram
2. Repeat until every path in the functional flow is represented in at least one scenario
3. Start with primary or 'sunny day' scenarios first
4. Later, add secondary scenarios and exception, error, and misuse paths
5. Add annotations or constraints for quality of service requirements

04

Hints

1. Every requirement traced to the use case should be represented in at least one scenario. If not, then add a lifeline, message, or scenario to get coverage

Sequence Diagrams



Use Case Scenario Recommendations

The key lifeline is the use case

- It could be ‘The System’ but more commonly it is the use case being analyized

Describe the scenario context

- Purpose
- Description
- Preconditions
- Postconditions
- Invariants

You need’t repeat the use case context

When commenting on specific scenarios only discuss the special conditions for this scenario. The use case context is “inherited” and needn’t be repeated

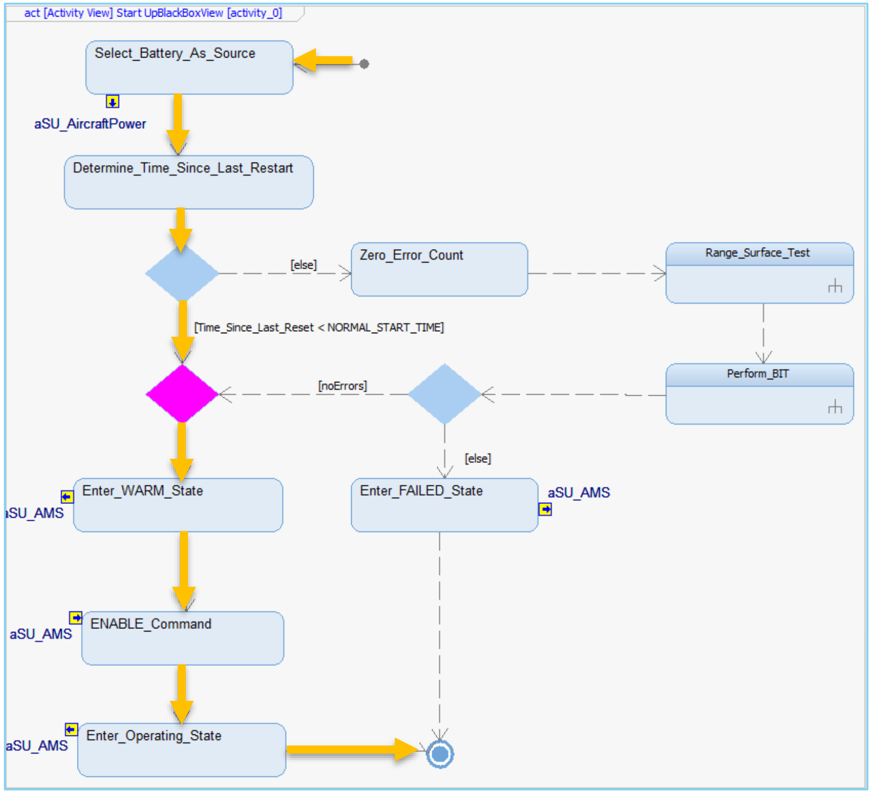
Time isn’t a lineline

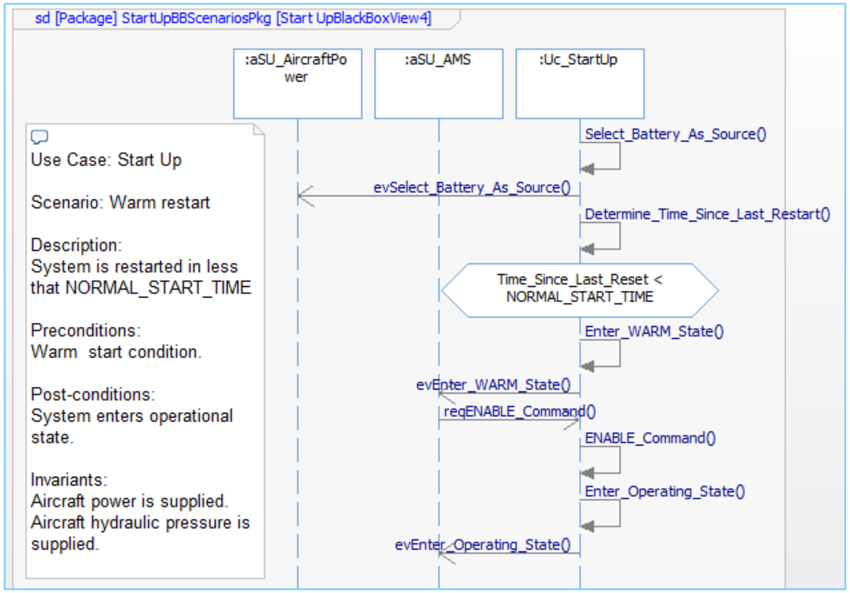
Represent timeouts as a time-triggered ‘message to self’

Add state marks when the functional flow is state-based

State marks show the change of state when appropriate

Producing Scenarios from Functional Flow




sd [Package] StartUpBBScenariosPkg [Start UpBlackBoxView4]
:aSU_AircraftPo wer
:aSU_AMS
:Uc_StartUp
Select_Battery_As_Source()
evSelect_Battery_As_Source()
Determine_Time_Since_Last_Restart()
Time_Since_Last_Reset < NORMAL_START_TIME
Enter_WARM_State()
evEnter_WARM_State()
reqENABLE_Command()
ENABLE_Command()
Enter_Operating_State()
evEnter_Operating_State()

Use Case: Start Up

Scenario: Warm restart

Description:
System is restarted in less that NORMAL_START_TIME

Preconditions:
Warm start condition.

Post-conditions:
System enters operational state.

Invariants:
Aircraft power is supplied.
Aircraft hydraulic pressure is supplied.

Let’s talk about requirements …

A requirement should be a simple, testable statement of *need* not of *design*

- It is common to use **shall** to indicate a requirement, **should** to indicate a recommendation, and **will** to indicate a requirement that will not be verified
  - E.g. “The system shall control the motor torque from 0.0 to 10,000.0 Newton-meters with an accuracy of 0.1NM. “

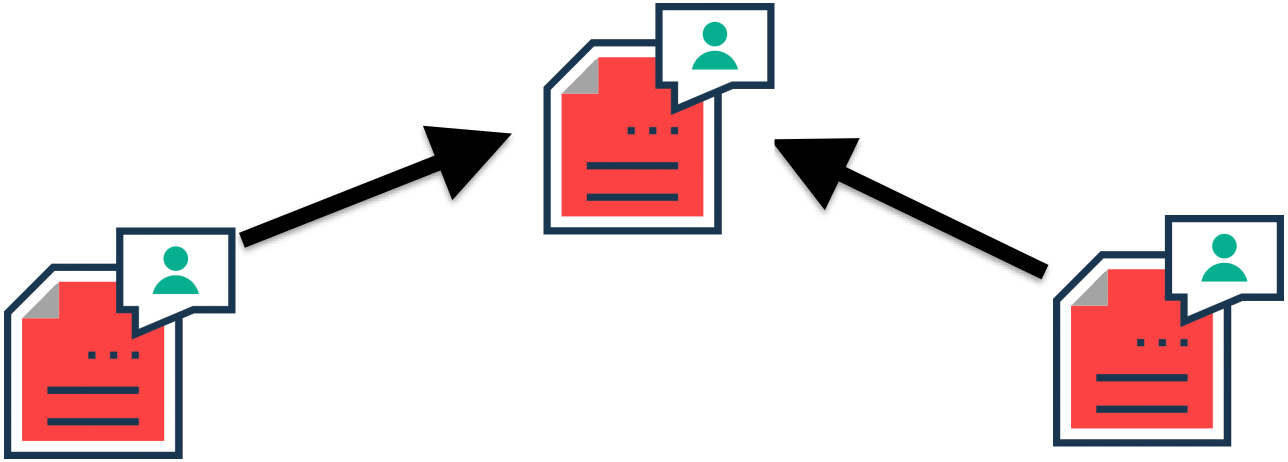
Requirements are organized by use case and allocated to user stories

- A use case is a coherent grouping of requirements around a usage of a system
  - Contains many requirements (often 10 – 100 requirements || 3 – 25 scenarios)
  - A use case may be thought of as a chapter within a requirements spec

Requirements are most commonly textual and captured in a word processing, spreadsheet or requirements management tool (e.g. Rational DOORS™)

- The big advantage of requirement management tools is that they provide traceability between requirements and between requirements and other work products.

Stakeholder vs System Requirements



Stakeholder requirement

A requirement is a statement of stakeholder need. It is a problem oriented statement.

Stakeholder requirements are generally less quantitative than system requirements

Requirements are developed in combination with the customer or subject matter expert and the requirements analyst

E.g. “The patient shall receive enough oxygen to sustain life for patients ranging from neonates to full adults in size and mass.”

System requirement

A requirement is a statement of what a system is required to do. It is a solution oriented statement with states qualities of service.

System requirements tend to be more quantitative than stakeholder requirements

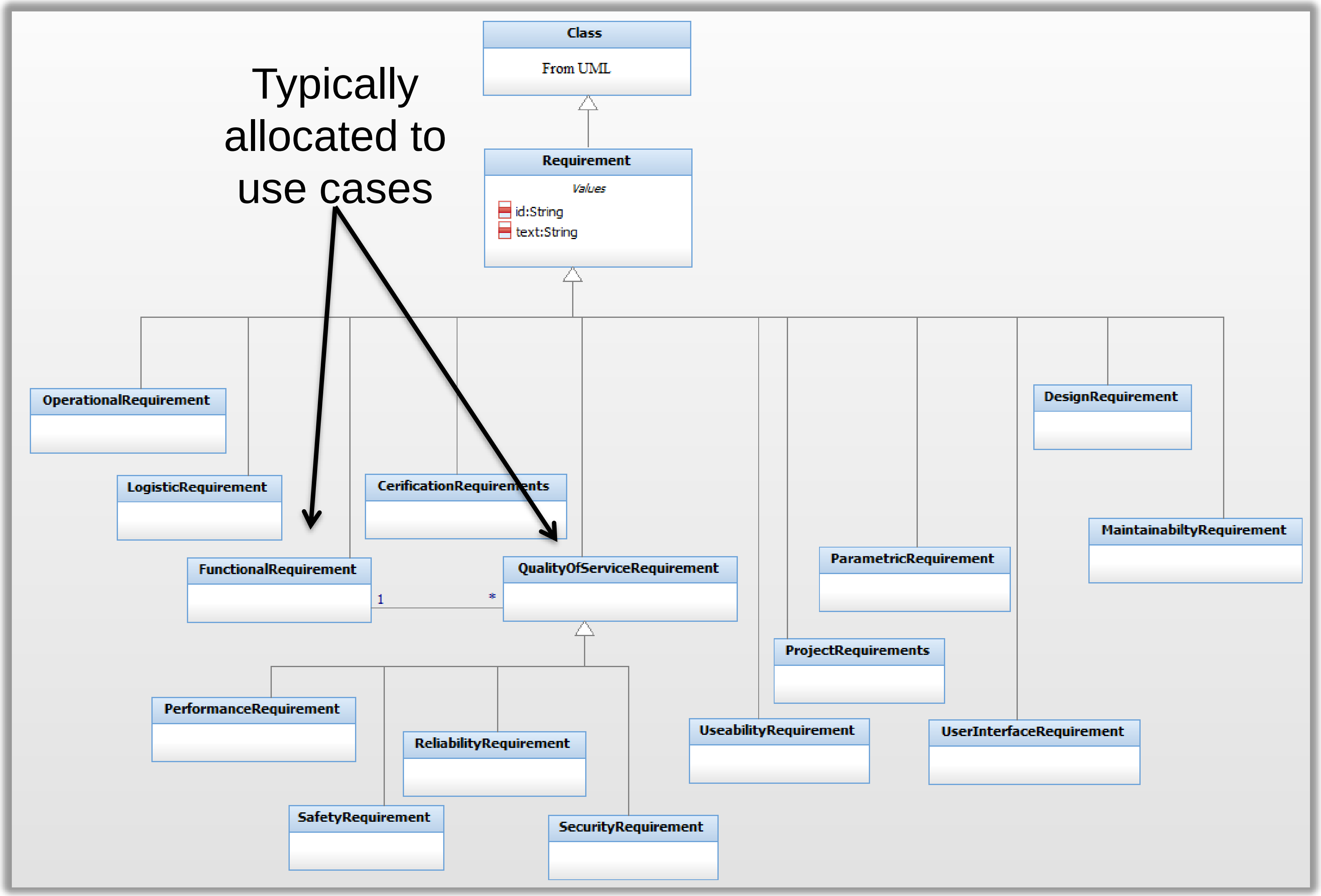
E.g “The system shall deliver oxygen from 50 ml/minute to 1,500 ml/minute settable

Requirements are developed in combination with the subject matter expert and the system engineer

System requirements should trace to stakeholder requirements

That is saying “What the system does should meet a stakeholder need”

Types of Requirements



Typically allocated to use cases

Types of requirements

- **Operational** (such as environmental conditions within which the system will function),
- **Logistics**, such as how the system will support business logistics flow or the logistics information provide by a system
- **Usability** – requirements about the ease of use and level of required skill and training of users
- **User interface** – requirements about how information or control features are presented to and controlled by the user; this includes languages and labeling
- **Parametric requirements** – requirements about static properties of the system such as weight or color
- **Maintainability Requirements** – requirements about the maintenance, repair, and upkeep of the system
- **Certification requirements** – requirements about what standards must be met and how the system will meet and be certified against them
- **Project requirements** – requirements about the development of the system itself, such as time, cost, and deliverables
- **Design requirement** – these are constraints levied against the design such as materials or the reuse of existing designs
- **Quality of Service Requirement** – qualifies "how well" a functional requirements must be performed

Typical Quality of Service (QoS) Requirements

- Flows (data, materiel, fluids, energy…)
  - Extent – what is the range of permitted values? Are all values within this range permitted or are there explicitly excluded values? What should happen if these ranges are violated?
  - Accuracy – what is the level of precision required for output?
  - Fidelity – what level of precisions is required for input?
- Performance
  - Worst case performance – what is the longest time this behavior should take to perform?
  - Average performance – on average, how long should this behavior take to perform?
  - Bandwidth – what is the rate of information or materiel transfer?
  - Throughput – what is the rate of successful information or materiel transfer?
  - Maximal delay – what is the longest time this behavior can be delayed?
  - Jitter – what variability in performance is allowed?
  - Signal to noise ratio – what is the proportional of the signal that is information?

Typical QoS Requirements

- Safety
  - Criticality – how critical is this system property?
  - Is there an appropriate safety objective (such as a Safety Integrity Level) required by a relevant safety standard?
- Reliability
  - Error rate – what is the rate at which unhandled errors may occur?
  - Mean time between failure (MTBF) – on average, how long should the system operate before failing?
  - Availability – what is the percentage of time the system aspect is available?
- Security
  - Value of assets – what is the value of features of the system that should be protected from interference, intrusion or theft?
  - Threats – what threats should the system handle?
  - Vulnerability – how easy should it be to attack aspects of the system?
  - Assurance – what is the level of guarantee that the system will not be successfully attacked?

Requirement Canonical Form

"The" <system> "shall" <key word> "a" <flow | attribute> | adjective | "adhering to the following constraints:"

- <constraint>
- <constraint>

[when subject to the following conditions: >

- <flow> | condition
- <flow> | condition

Compare this to the canonical form for a user story:

"As a " <user> "I want" <feature> "so that " <reason>

Types of Requirements - examples

"The" <system> "shall" <key word> "a" <flow | attribute> | adjective | "adhering to the following constraints:

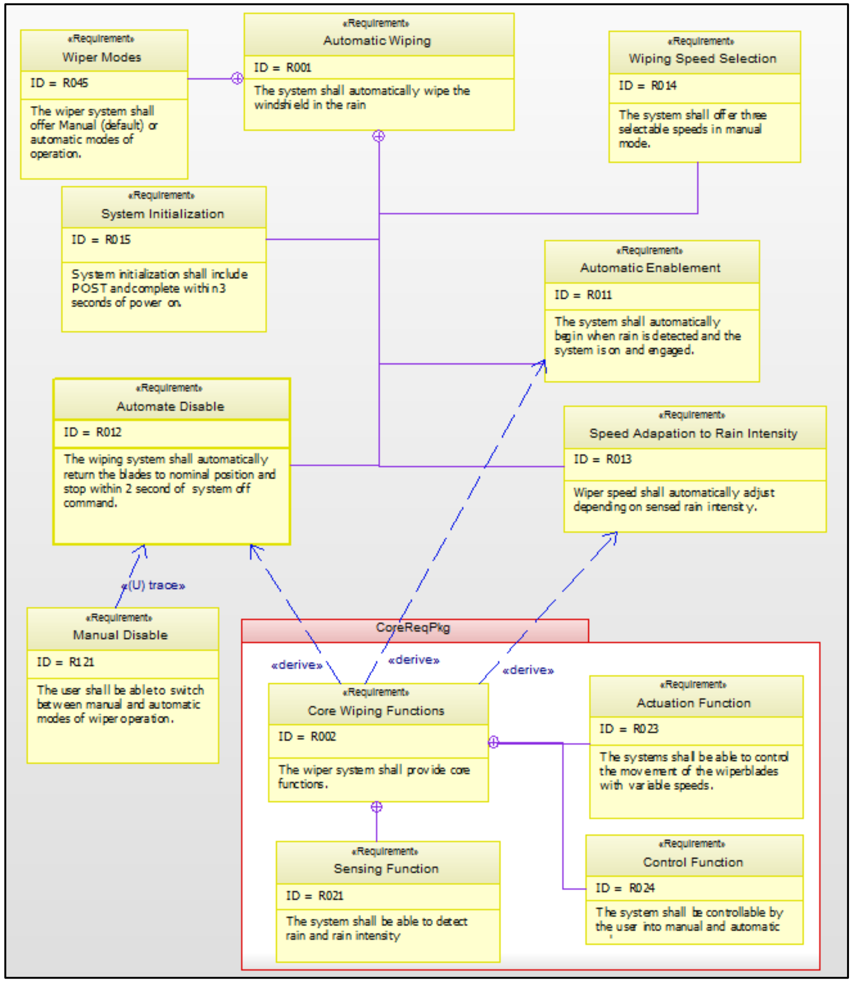
- <constraint>
- <constraint>

[when subject to the following conditions: >

- <flow> | condition
- <flow> | condition "

- Stakeholder requirements
  - The aircraft shall control roll, pitch and yaw smoothly
- Functional requirements
  - The system shall maintain all three rotational axes of airframe stability when subject to the following conditions: steady winds.
- Functional quality of service (QoS)
  - The system shall maintain airframe stability in all three rotational axes adhering to the following constraint: within 2 degrees of arc when subject to the following conditions: in the presence of gusts up to 40 kph.
- System parametrics
  - The system shall weigh no more than 30 kg when subject to the following conditions: fully loaded with hydraulic fluid.
  - The system shall have a maximum current draw adhering to the following constraint: 0.5 amps at 240V.
- Project QoS
  - The system design shall use existing hydraulic components adhering to the following constraint: from the FlightMagic system.

Representing Requirements



«Requirement»
Wiper Modes
ID = R045
The wiper system shall offer Manual (default) or automatic modes of operation.

«Requirement»
Automatic Wiping
ID = R001
The system shall automatically wipe the windshield in the rain

«Requirement»
Wiping Speed Selection
ID = R014
The system shall offer three selectable speeds in manual mode.

«Requirement»
System Initialization
ID = R015
System initialization shall include POST andcomplete within3 seconds of power on.

«Requirement»
Automatic Enablement
ID = R011
The system shall automatically begin when rain is detected and the system is on and engaged.

«Requirement»
Automate Disable
ID = R012
The wiping system shall automatically return the blades to nominal position and stop within 2 second of system off command.

«Requirement»
Speed Adapation to Rain Intensity
ID = R013
Wiper speed shall automatically adjust depending on sensed rain intensity.

«Requirement»
Manual Disable
ID = R121
The user shall be ableto switch between manual and automatic modes of wiper operation.

CoreReqPkg

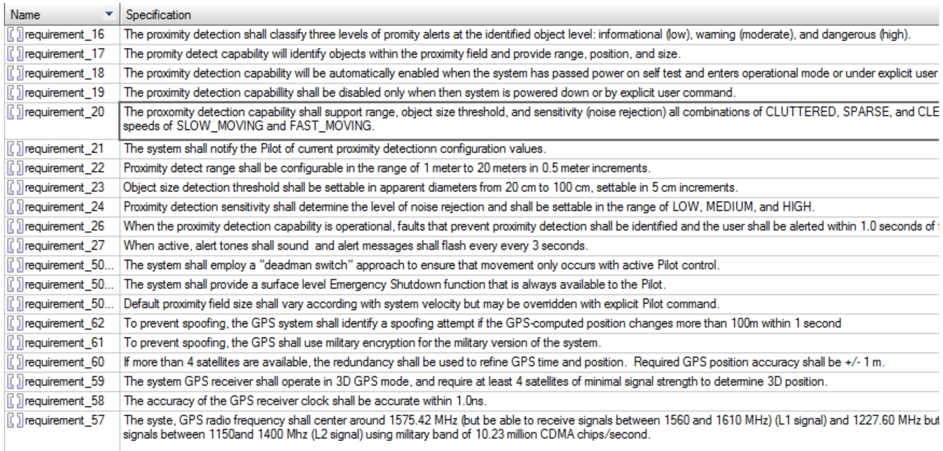
«Requirement»
Core Wiping Functions
ID = R002
The wiper system shall provide core functions.

«Requirement»
Actuation Function
ID = R023
The systems shall be able to control the movement of the wiperblades with variable speeds.

«Requirement»
Sensing Function
ID = R021
The system shall be able to detect rain and rain intensity

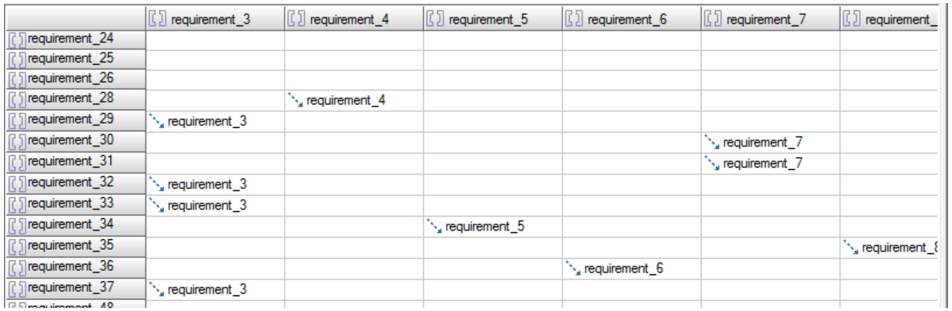
«Requirement»
Control Function
ID = R024
The system shall be controllable by the user into manual and automatic

Requirements Diagram



Name	Specification
requirement_16	The proximity detection shall classify three levels of promity alerts at the identified object level: informational (low), warning (moderate), and dangerous (high).
requirement_17	The promity detect capability will identify objects within the proximity field and provide range, position, and size.
requirement_18	The proximity detection capability will be automatically enabled when the system has passed power on self test and enters operational mode or under explicit user
requirement_19	The proximity detection capabillity shall be disabled only when then system is powered down or by explicit user command.
requirement_20	The proxomity detection capability shall support range, object size threshold, and sensitivity (noise rejection) all combinations of CLUTTERED, SPARSE, and CLE speeds of SLOW_MOVING and FAST_MOVING.
requirement_21	The system shall notify the Pilot of current proximity detectionn configuration values.
requirement_22	Proximity detect range shall be configurable in the range of 1 meter to 20 meters in 0.5 meter increments.
requirement_23	Object size detection threshold shall be settable in apparent diameters from 20 cm to 100 cm, settable in 5 cm increments.
requirement_24	Proximity detection sensitivity shall determine the level of noise rejection and shall be settable in the range of LOW, MEDIUM, and HIGH.
requirement_26	When the proximity detection capability is operational, faults that prevent proximity detection shall be identified and the user shall be alerted within 1.0 seconds of
requirement_27	When active, alert tones shall sound and alert messages shall flash every every 3 seconds.
requirement_50...	The system shall employ a "deadman switch" approach to ensure that movement only occurs with active Pilot control.
requirement_50...	The system shall provide a surface level Emergency Shutdown function that is always available to the Pilot.
requirement_50...	Default proximity field size shall vary according with system velocity but may be overridden with explicit Pilot command.
requirement_62	To prevent spoofing, the GPS system shall identify a spoofing attempt if the GPS-computed position changes more than 100m within 1 second
requirement_61	To prevent spoofing, the GPS shall use military encryption for the military version of the system.
requirement_60	If more than 4 satellites are available, the redundancy shall be used to refine GPS time and position. Required GPS position accuracy shall be +/- 1 m.
requirement_59	The system GPS receiver shall operate in 3D GPS mode, and require at least 4 satellites of minimal signal strength to determine 3D position.
requirement_58	The accuracy of the GPS receiver clock shall be accurate within 1.0ns.
requirement_57	The syste, GPS radio frequency shall center around 1575.42 MHz (but be able to receive signals between 1560 and 1610 MHz) (L1 signal) and 1227.60 MHz but signals between 1150and 1400 Mhz (L2 signal) using military band of 10.23 million CDMA chips/second.

Requirements Table



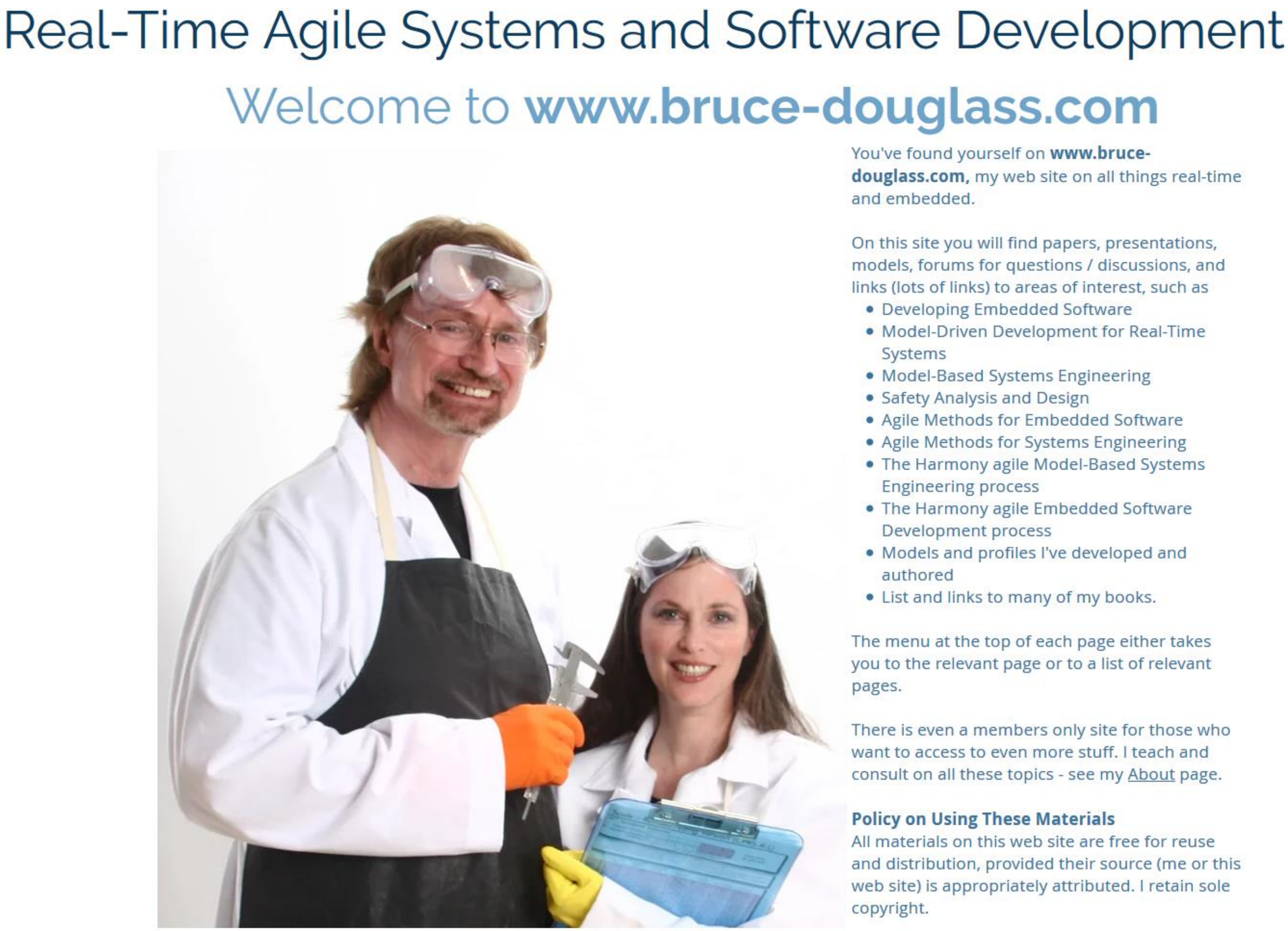
	requirement_3	requirement_4	requirement_5	requirement_6	requirement_7	requirement_
requirement_24						
requirement_25						
requirement_26						
requirement_28		requirement_4				
requirement_29	requirement_3					
requirement_30					requirement_7	
requirement_31					requirement_7	
requirement_32	requirement_3					
requirement_33	requirement_3					
requirement_34			requirement_5			
requirement_35						requirement_8
requirement_36				requirement_6		
requirement_37	requirement_3					
requirement_48						

Trace between stakeholder and system requirements

Allocation requirements

- Method: associate a traceable relation between the use case (pointing to) the requirement(s). Possibilities include:
  - «satisfy»
  - «allocate»
  - «trace»
  - Unadorned dependency
- Approach 1: Use case diagram
  - For each use case,
    - create a use case diagram,
    - add relevant requirements, and
    - add relations
- Approach 2: Use case allocation matrix
  - Create a use case-requirements matrix type (best if use cases are columns) of the appropriate cell relation type
  - Click in the cells to add/remove the relation
- Note: regardless of the approach taken, you will probably want to create a use case-requirements trace matrix to look for orphan requirements

Download Papers, Presentations, Models, & Profiles for Free



Real-Time Agile Systems and Software Development

Welcome to **www.bruce-douglass.com**

You've found yourself on **www.bruce-douglass.com,** my web site on all things real-time and embedded.

On this site you will find papers, presentations, models, forums for questions / discussions, and links (lots of links) to areas of interest, such as

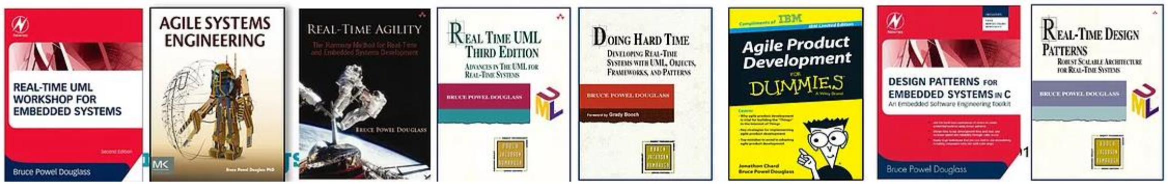
- Developing Embedded Software
- Model-Driven Development for Real-Time Systems
- Model-Based Systems Engineering
- Safety Analysis and Design
- Agile Methods for Embedded Software
- Agile Methods for Systems Engineering
- The Harmony agile Model-Based Systems Engineering process
- The Harmony agile Embedded Software Development process
- Models and profiles I've developed and authored
- List and links to many of my books.

The menu at the top of each page either takes you to the relevant page or to a list of relevant pages.

There is even a members only site for those who want to access to even more stuff. I teach and consult on all these topics - see my About page.

Policy on Using These Materials

All materials on this web site are free for reuse and distribution, provided their source (me or this web site) is appropriately attributed. I retain sole copyright.


REAL-TIME UML WORKSHOP FOR EMBEDDED SYSTEMS
Bruce Powel Douglass
AGILE SYSTEMS ENGINEERING
REAL-TIME AGILITY
Bruce Powel Douglass
REAL TIME UML THIRD EDITION
DOING HARD TIME
Agile Product Development for DUMMIES
DESIGN PATTERNS FOR EMBEDDED SYSTEMS IN C
Bruce Powel Douglass
REAL-TIME DESIGN PATTERNS