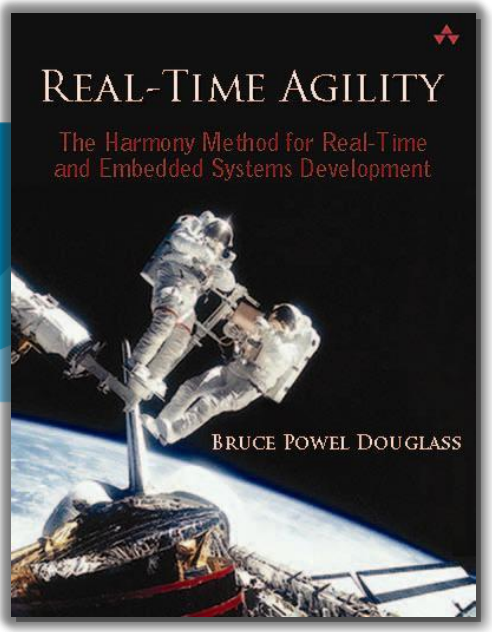
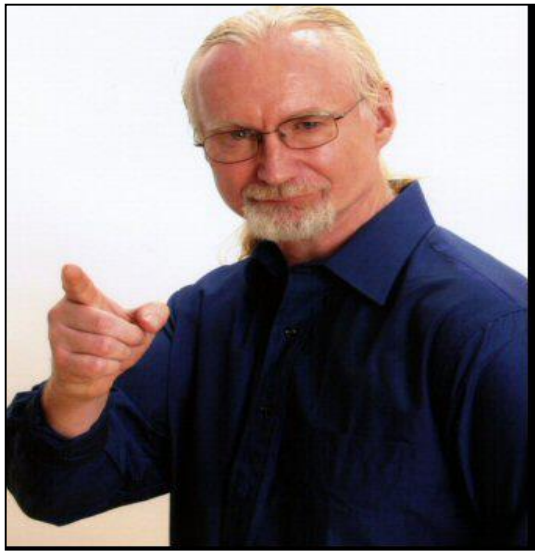
Overview of Agile Practice for Embedded Software Development

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





Being “Agile” means being flexible enough to do what makes sense. The notion of “strict Agile” is an oxymoron.

Law Of Douglass #149

Embedded Concerns vs Agile Practices

Enhanced Optimization

Broadened "Definition of Done"

3 Schedules

Test Coverage Analysis

Incremental Dependability Analysis

Simulation

Computable Models

Planning for Conformance

Hand off from Systems Engineering

Cross functional teams

Design Patterns

CONCERNS

Specialized Hardware

HW-SW Co-Development

Language/Tools Constraints

Development vs Target Environments

Predictability & Timeliness

Resource Constraints

Safety & Reliability

Standards Conformance

Stand-Alone Product Release

Fixed Price Bid

Follows Systems Engineering

Missing Architecture

Test-Driven Development

Continuous Integration

5 Views of Architecture

Defensive Design

Incremental Traceability

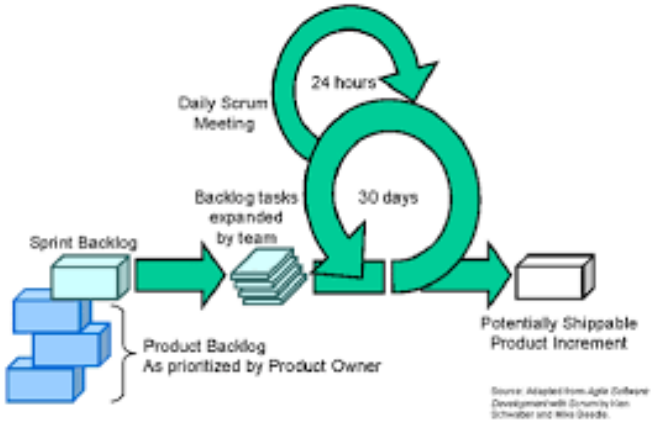
Work Product Audits

Short development cycles

Mapping Concerns to Practices

Practice ➔ / Concern ⬇	Incremental Dependability	Simulation	Computable Models	Plan for Conformance	Hand off from SE	Cross Functional teams	Design Patterns	Short Dev Cycles	Work Product Audits	Incremental trace	Defensive Design	5 View of Architecture	Continuous Integration	Test Driven Development	Test Coverage	3 Schedules	Definition of Done	Enhanced Optimization
Specialized Hardware		✓			✓		✓					✓	✓	✓				✓
HW-SW Co-Development		✓	✓		✓	✓		✓			✓	✓	✓	✓			✓	✓
Language/Tools Constraints		✓	✓	✓			✓				✓		✓	✓	✓			✓
Dev / Target Environments		✓	✓			✓	✓	✓	✓	✓			✓	✓	✓		✓	✓
Predictability and Timeliness	✓	✓	✓	✓	✓		✓				✓	✓		✓	✓		✓	✓✓
Resource Constraints		✓	✓		✓	✓	✓				✓	✓	✓	✓	✓			
Safety & Reliability	✓✓		✓	✓	✓		✓		✓	✓	✓	✓			✓		✓	
Standards Conformance	✓			✓✓	✓		✓		✓	✓		✓			✓		✓	
Stand-Alone Release		✓	✓								✓		✓	✓			✓	
Fixed Price Bid									✓							✓	✓	
Follows SE					✓✓	✓							✓					
Missing Architecture						✓				✓		✓						

Practice: Short Development Cycles



Purpose

Reduce project risk by producing partially-complete versions of the system to ensure that the stakeholders needs are being met.

Description

The software is developed in 1-4 week iterations or “sprints”; at the end of each iteration, a tested version of the system running on at least one platform, is produced

Key points

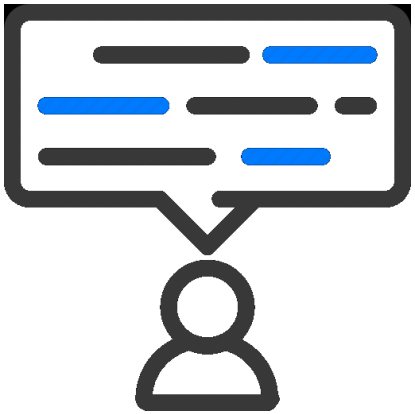
1. Perform iteration planning
2. Incrementally elaborate the software by adding the design and implementation of a small set of use cases or user stories
3. Deliver (not necessarily to the stakeholders) a verified version of the software
4. Perform an iteration review
5. Perform an iteration retrospective

It’s all about the work items

Work items include

- Addition of new functionality as described in user stories or use cases
- Performance of one or more risk-reduction spikes
- Resolution of previously identified defects
- Addition of architectural feature
- Support for one or more platform
- Refactoring of software basis

Use Case & Use Story



Canonical form for a user story:

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

Purpose

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

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

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.

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.**

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,

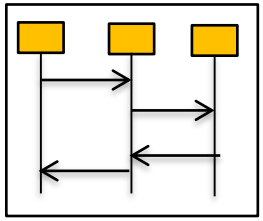
As a **pilot**, I **want the pedal to control the rudder in a range of -30 to +30 degrees** so **that I can steer left or right**.

- 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’

Epics, use cases, and user stories, Oh My

Months

(multiple iterations)

EPIC

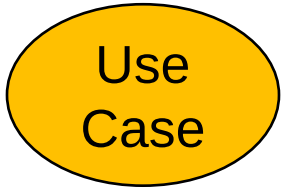
An epic is a coherent set of features, use cases, and user stories at a strategic level.

Weeks

(single iteration)

Feature

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



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)

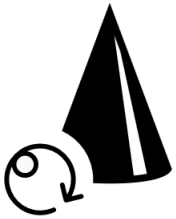
Work Item

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

User Story

A kind of (large scale) 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 iteration)

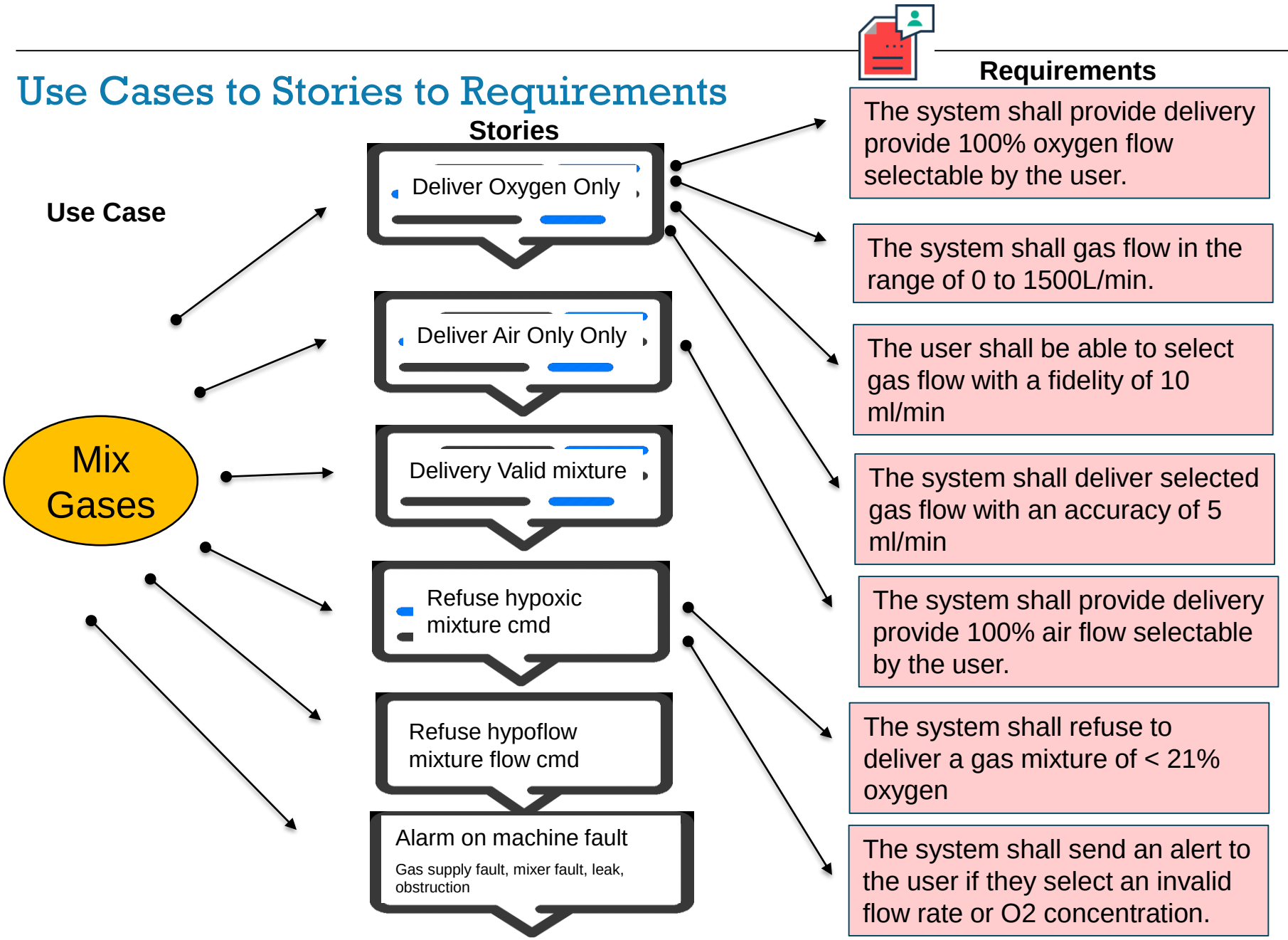
Task

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

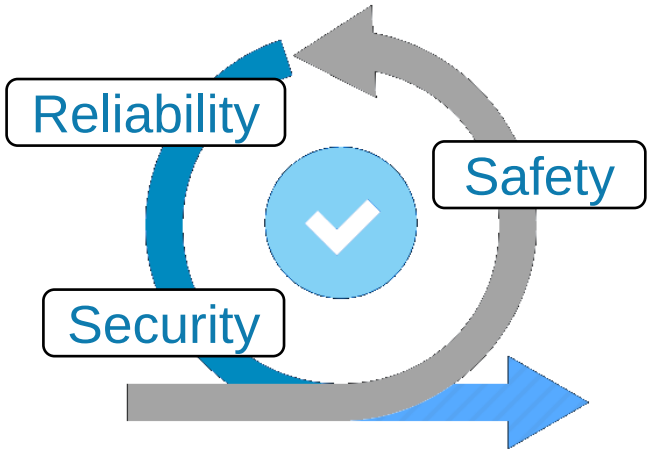
Requirement

A simple, testable statement of what a system must do.

Use Cases to Stories to Requirements



Practice: Incremental Dependability



Purpose

Address safety, reliability, and security concerns in an incremental, iterative fashion

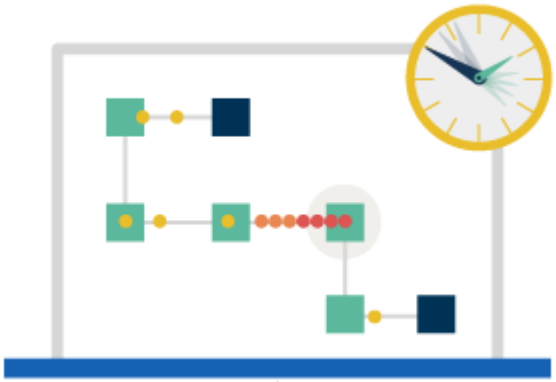
Description

- Use cases / user stories group requirements including *how dependably* those requirements will be met. Introduce these concerns in that context.
- As technical decisions are made, the system dependability is reevaluated and new dependability requirements are added.

Key points

1. **Safety** is “Freedom from loss”
   Reliability is a “measure of the availability of services”
   Security is “protections against intrusion or attack”
2. Concerns can be either
   1. Intrinsic – a functional of the nature of the system and its role, irrespective of implementation
   2. Technical – resulting from technology and design choice
3. This usually is used in conjunction with practices
   1. Incremental Traceability practice
   2. Planning for Conformance

Practices: Simulation and Computable Models



Purpose

Understand a process, system, or design without having to construct the actual thing.

Description

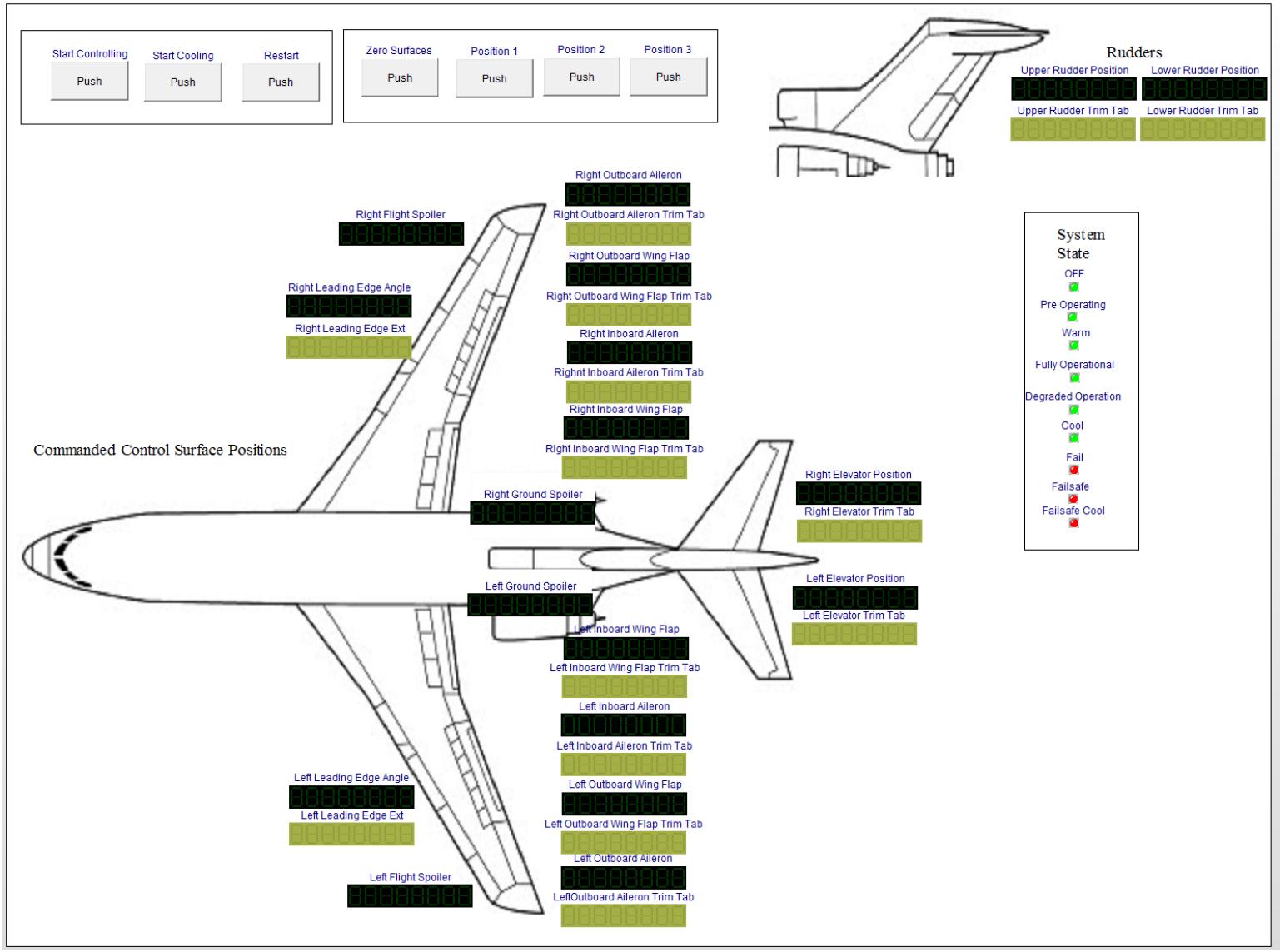
A simulation is an approximate imitation of the operation of a process or system; the act of simulating first requires a model is developed. This model is a well-defined description of the simulated subject, and represents its key characteristics, such as its behavior, functions and abstract or physical properties

Key points

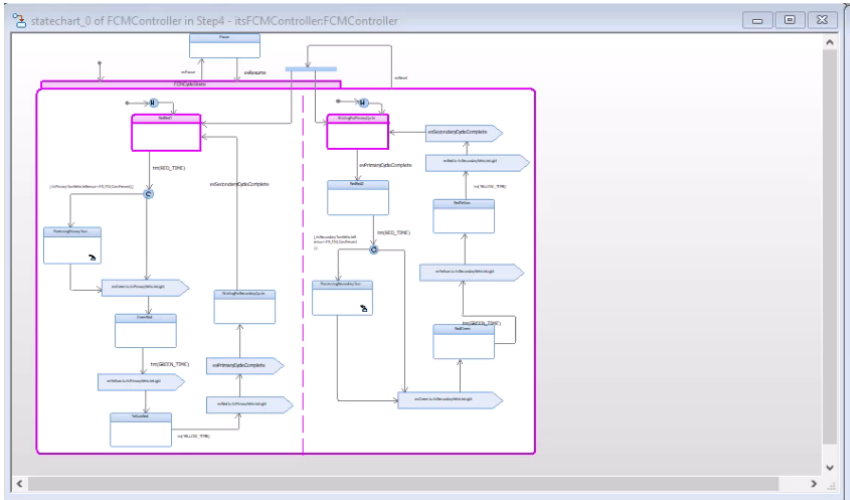
In this context, the uses of simulation and computable models are many:

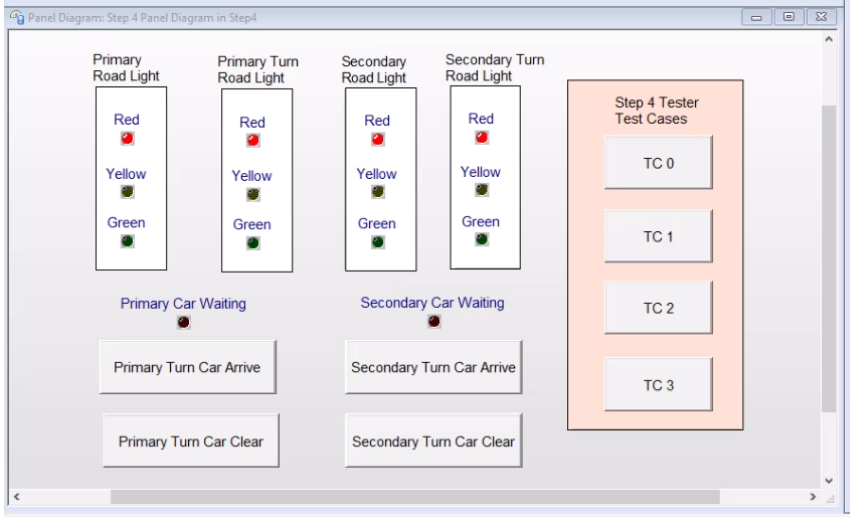
- Verify consistency and correctness of requirements for a use case or user story
- Construct / evaluate / understand an architecture or design
- Automatically generate code from a design model
- Evaluable quality of service properties of a design or use case such as
    - Performance
    - Safety
    - Reliability
    - Security

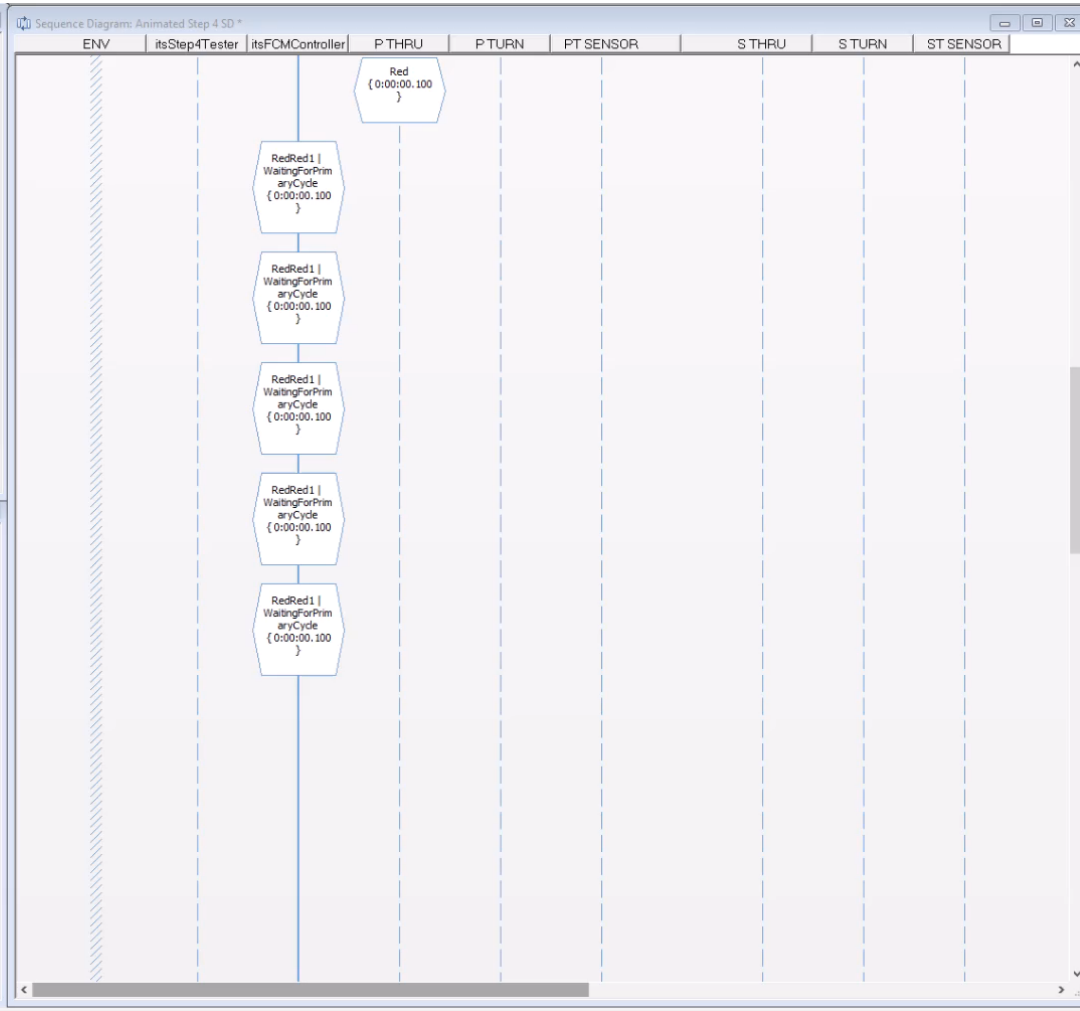
Control Surfaces System Simulation Control Panel Diagram



Computable Models







Practice: Planning for Conformance

Purpose

In regulated industries, where standards compliance is required, planning early for conformance is crucial risk reduction

Description

Conformance to standards doesn't happen accidently or incidentally. It requires planning and execution. A Plan for Conformance ensures that all the required work products and process adherence objectives for a development will be met. Ideally, the plan would be developed in conjunction with the regulator.

Key points

A *Plan for Software Aspects of Certification* (PSAC) is a common example in which part of project planning includes identification of how each of the regulatory objectives of the DO-178 standard will be met. These identified tasks become part of the development process; the required engineering data becomes part of the deliverables; the acceptance criteria for these work products becomes part of their "definition of done"

Commonly used with practices

- Incremental Dependability Analysis
- Work Process Audits
- Test Coverage Analysis
- Incremental Traceability

Practice: Hand Off From Systems Engineering

Purpose

Properly hand off important engineering data from systems engineering to downstream engineering (including software development) is a fashion which is both timely and usable.

Description

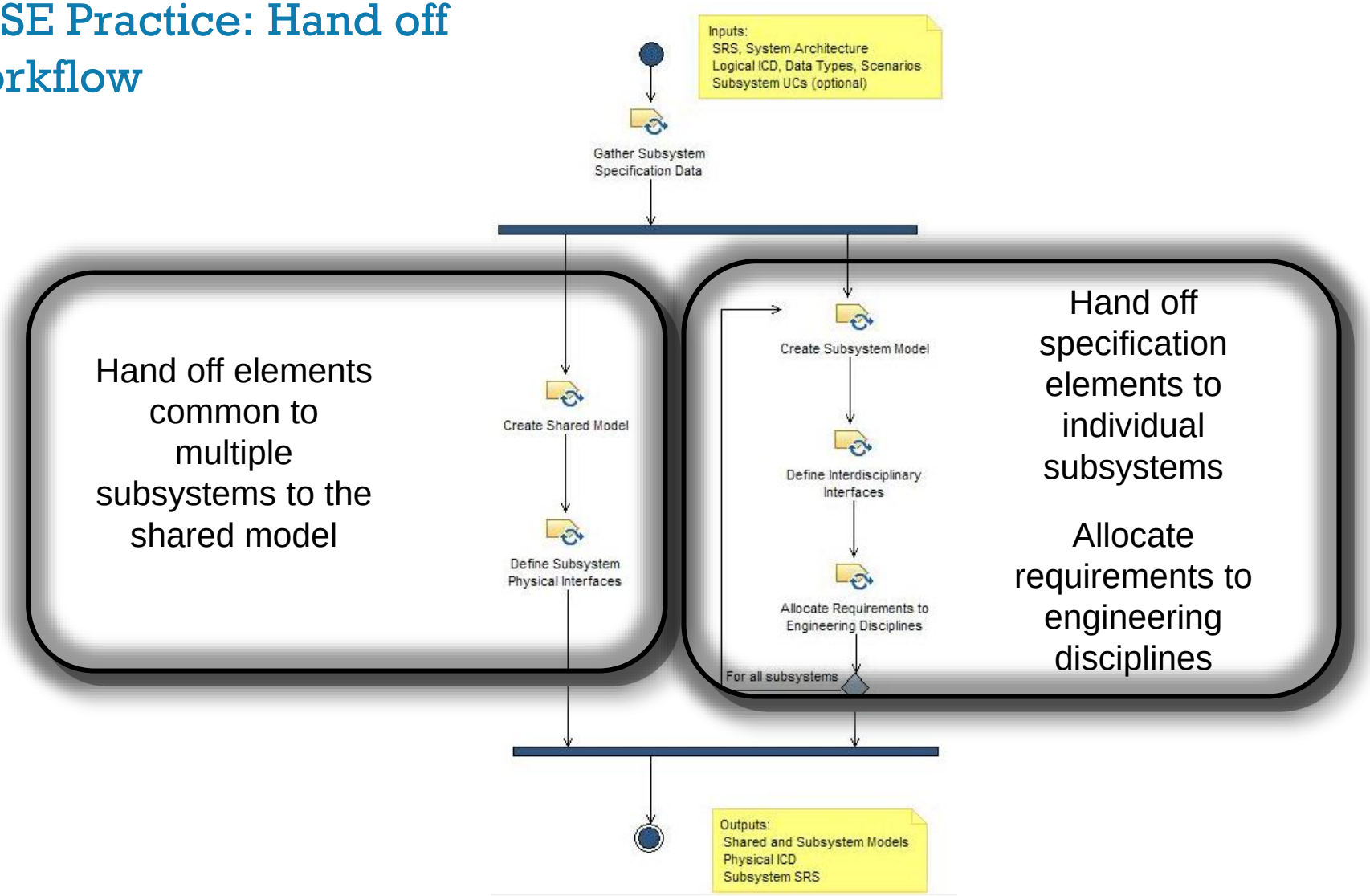
Systems engineering is a completely different discipline from software development; it deals with the definition of system properties which are then decomposed and derived into hardware & software requirements, architectures, and designs. Because of the differing concerns, skills, and tools, care must be taken to ensure that the information is handed to software well.

Key points

The hand off can be done monolithically, but it is best done incrementally as a small set of use cases or user stories and related engineering data in each iteration. Hand off should generally include:

- Physical system and subsystem interfaces and physical data schema for data carried by the interfaces
- Subsystem Architecture
- Deployment Architecture within the subsystem
- Subsystem Software Requirements, use cases, and user stories
- Inter-disciplinary interfaces (specifically software-electronic interfaces)

MBSE Practice: Hand off Workflow



Practice: Cross Functional Teams

Purpose

The problems of working in silos are well-documented. The purpose of this practice is to allow team members with different skill sets to effectively contribute to the development process.

Description

When software is embedded it is even more important to have a representatives of different areas working with the team to ensure effective development with fewer errors of understanding and omission.

Key points

Practice has shown that a diverse team background results in improved quality. In an embedded environment, such a team will often include

- Customer or customer representatives (such as technical marketing)
- Systems engineering
- Electronics
- Subject matter experts
- Management
- Testing
- Manufacturing
- Maintenance personnel
- Sales

Practice: Design Patterns

Purpose

The purpose of design pattern is to reuse design approaches that have previously proved to be effective and thereby result in better designs. .

Description

Design patterns are

- generalized solutions to recurring optimization problems
- Parameterized collaborations of objects, where the object roles are the formal parameters and the objects that play those roles are the actual parameters when the pattern is instantiated

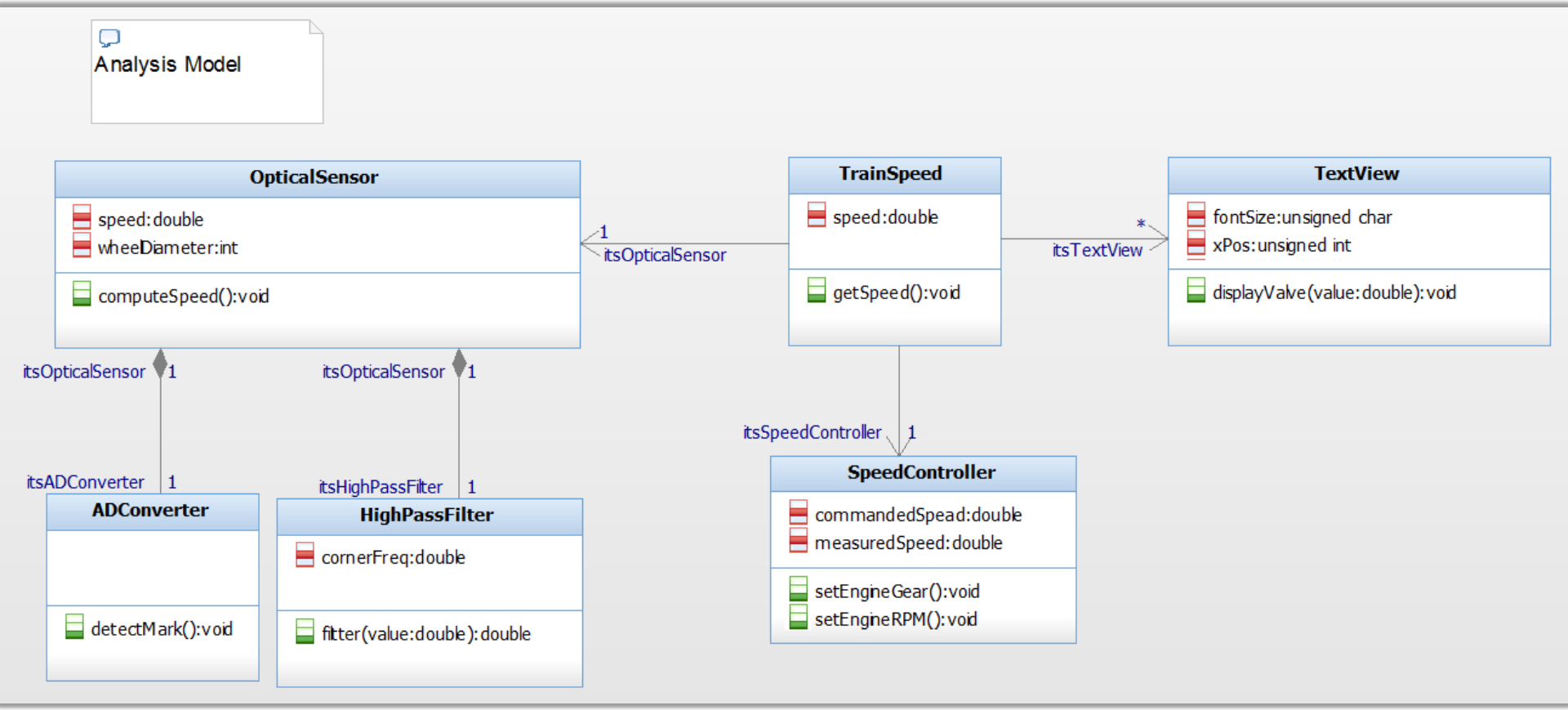
Key points

- Design patterns optimize some properties of a solution at the expense of others.
- Design patterns contain two kinds of elements
  - Formal parameters – for which you substitute elements in your own specific design
  - Glue elements – elements that orchestrate the behavior of the pattern
- Patterns have some important aspects
  - Applicability
  - Optimization
  - Solution
  - Consequences – pros and cons

Example Analysis Model Collaboration

Optimization Criteria:

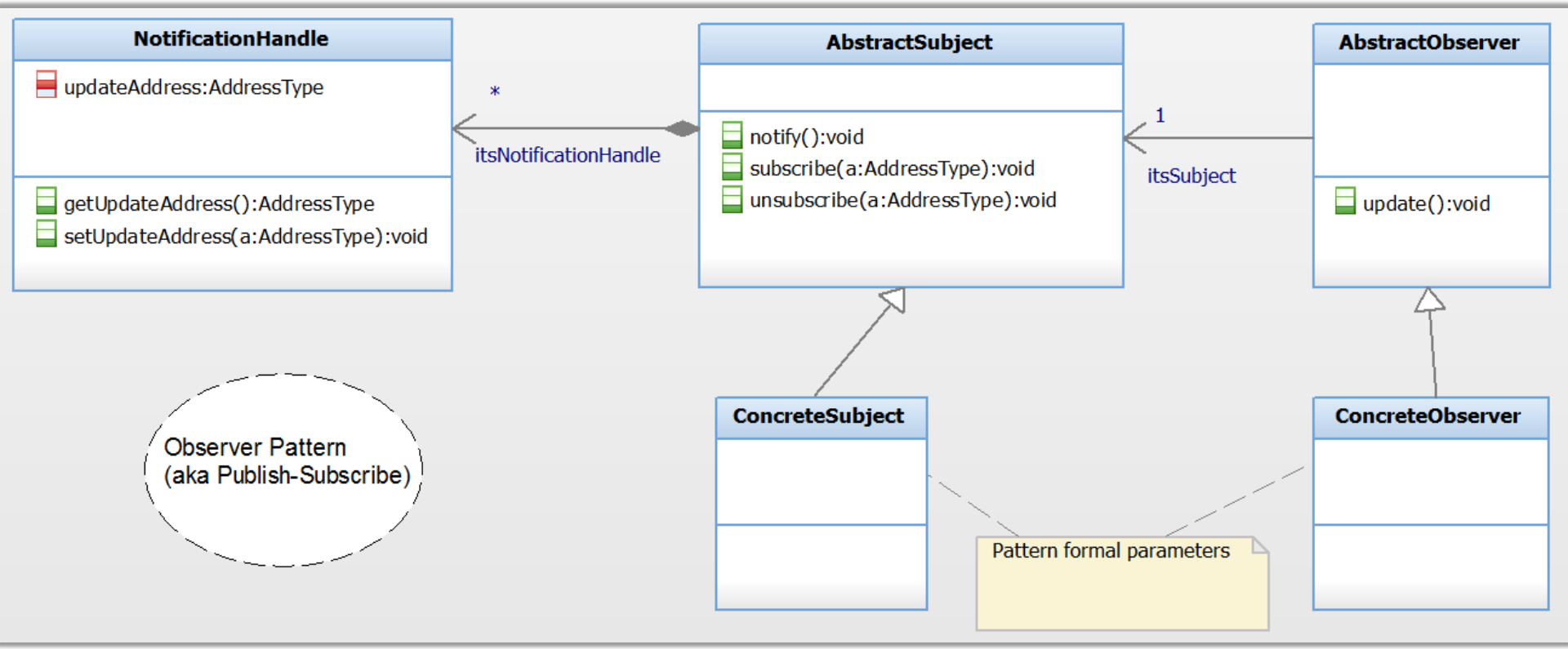
1. Ease of adding new **TextView** (and other) clients
2. Efficient use of bus bandwidth (only send data on bus when necessary)



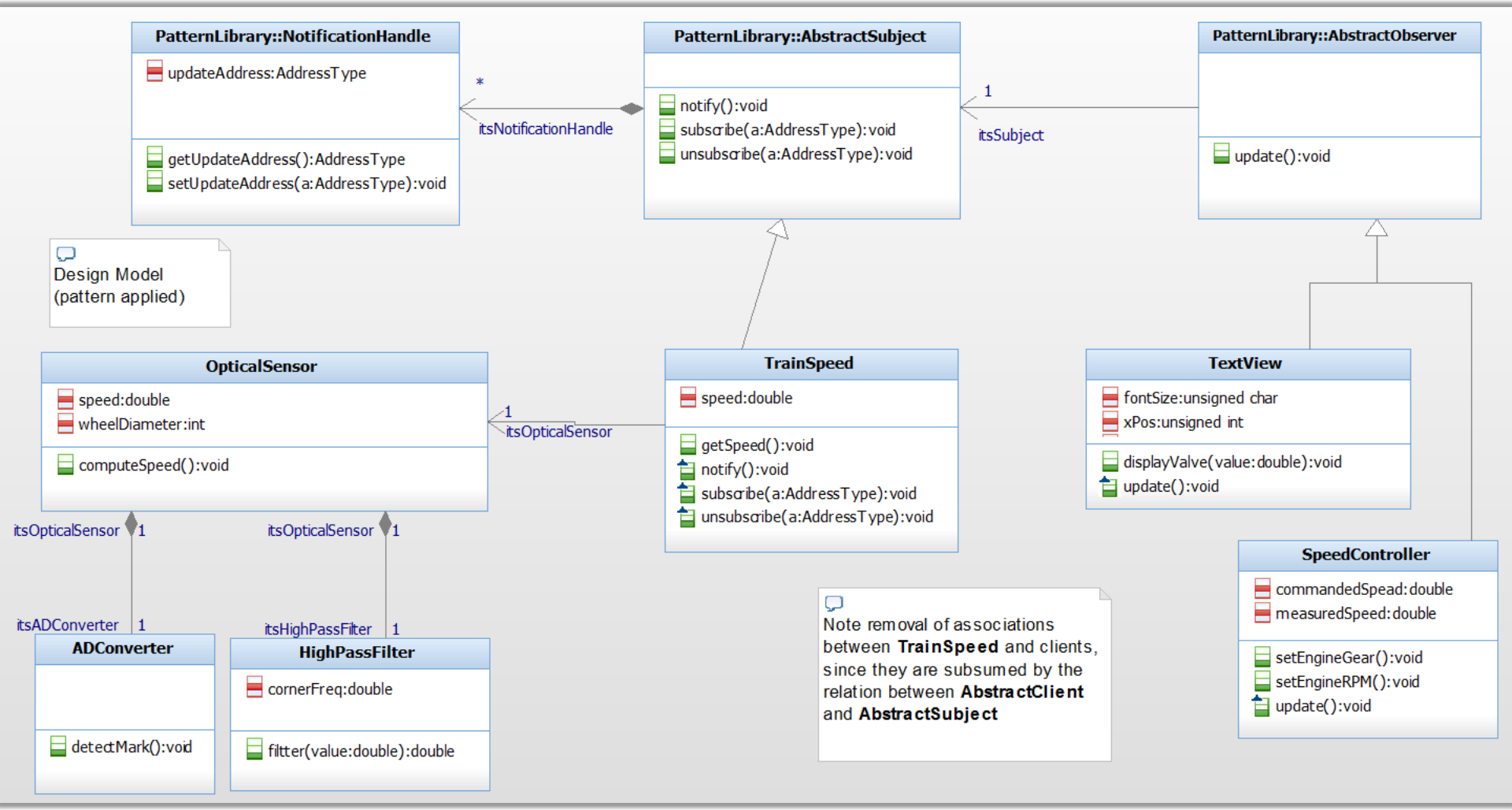
Selecting Patterns using Design Tradeoff Analysis

Design Solution	Design Criteria				Total Weighted Score
	Execution Efficiency	Maintainability	Run-time Flexibility	Memory Usage	
	Weight = 7	Weight = 5	Weight = 4	Weight = 7	
	Score	Score	Score	Score	
Client Server	3	7	8	5	123
Push Data	8	4	7	9	167
Observer Pattern	**8**	**7**	**9**	**9**	**190**

Design Pattern: Observer Pattern Specification



Design Pattern: Observer Pattern Instantiation



Practice: Work Product Audits

Purpose

Work product audits are to ensure compliance of a work product to a standard: so-called “syntactic correctness”. It also ensures consistency among work products produced by different teams and team members.

Description

Each important work item should have a standard by which it can be judged to be “well-formed” , Aspects assessed typically include:

- Product format
- Product organization
- Product content

Key points

- Used in conjunction with the “Broadened Definition of Done” practice
- Syntactic correctness does not ensure semantic correctness or vice versa.
- This is more important in regulated industries but is important in any project for product quality consistency.
- This is often performed by Quality Assurance personnel using a checklist in which each objective identified in the related standard is represented
- Commonly applied to
    - Requirements collections
    - Designs
    - Models
    - Code
    - Test plans and procedures

Practice: Incremental Traceability

Purpose

Traceability's purposes include:

- Completeness assessment
- Allows the demonstration of the consistency of information capture in work products that may span different disciplines, tools, and representation,
- Support impact analysis
- Provide justification (why is this design element here?)

Description

Traceability means that a navigable relation exists between all related project data, without regard to the data's location (within work products) or format. This is commonly done through spreadsheets linking semantic statements in one work product to related statements in another, or by tools specializing in creation and maintenance of trace relations, such as DOORS™ or RELM™.

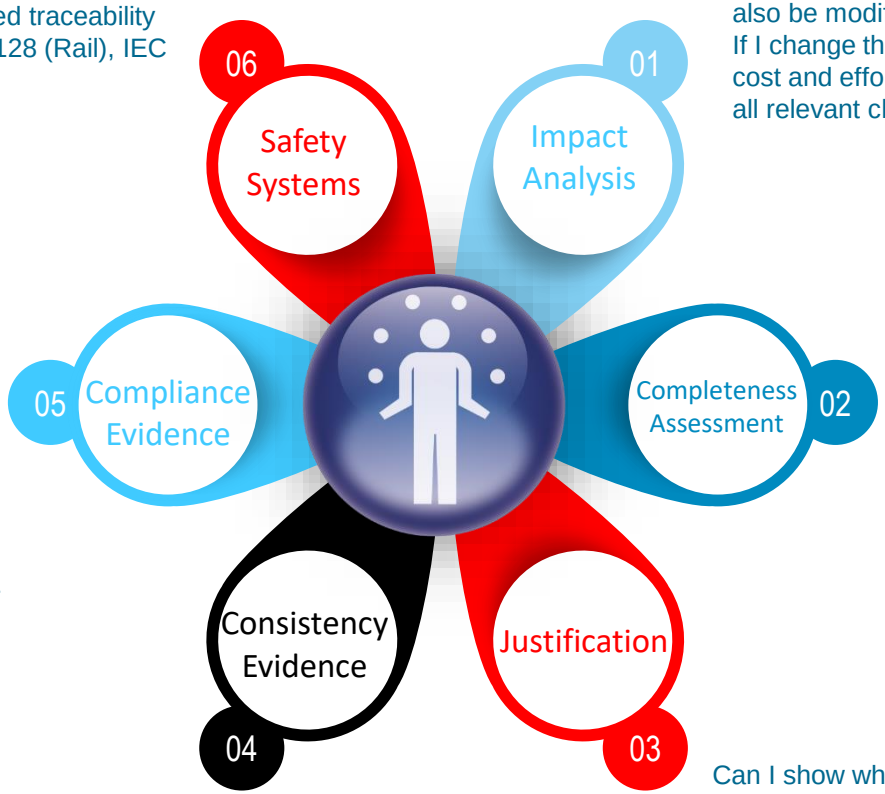
Key points

- For systems in regulated industries or when the "cost of system failure is high", traceability provides the means for demonstrating the consistency of engineering data across word products, tools, and processes
- Impact Analysis is "Identifying the potential consequences of a change, or estimating what needs to be modified to accomplish a change"
- **Impact analysis is more important for incremental development because refactoring is an ongoing, rather than occasional, activity**

Why Traceability

Safety standards require detailed traceability
E.g. DO-178 (Avionics), EN 50128 (Rail), IEC 62304 (Medical)

If I change this datum, what other project data is affected and must also be modified?
If I change this datum, what is the cost and effort required to make all relevant changes?



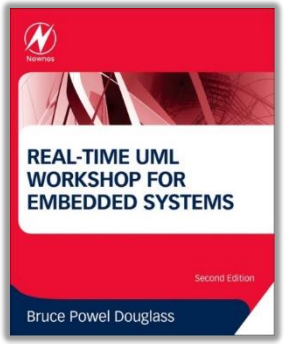
Does the data comply with internal and external standards
E.g. QA Records show process was followed (audit) or work product is well-formed (review)

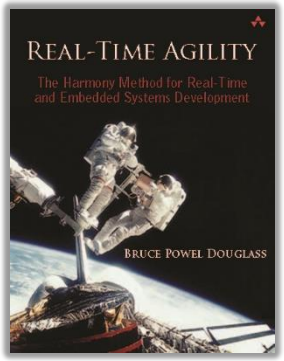
Have I (demonstrably) realized each datum?
E.g. Safety Objective, Requirement or design element

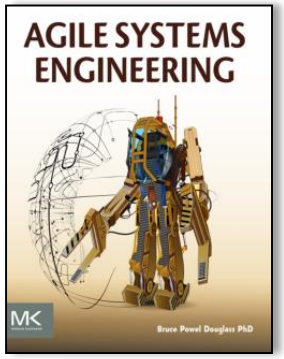
Are the data within different aspects of the project data consistent?
E.g. Are the requirements consistent with the safety assessment? Does the implementation meet the design?

Can I show why this data is present?
E.g. Is this design element present to meet a requirement?

I was able to find some books on Agile and Traceability …







- Discusses traceability with respect to requirements, safety analysis, architecture, design and test

Establishing Traceability to the Stakeholder Requirements and Use Cases

Traceability is useful for both change impact analysis and to demonstrate that a system meets the requirements or that the test suite covers all the requirements. It is also useful to demonstrate that each design element is there to meet one or more requirements, something that is required by some safety standards, such as DO-178B.[9]

To: Requirement **Scope:** RequirementsPkg

From: UseCase **Scope:** RequirementsPkg

	REQ_pt1	REQ_pt2	REQ_pt3	REQ_pt4	REQ_pt5	REQ_pt6	REQ_pt7	REQ_pt8	REQ_b1
Cargo Transport		REQ_pt2	REQ_pt3		REQ_pt5	REQ_pt6	REQ_pt7		REQ_b1
Personnel Transport	REQ_pt1	REQ_pt2	REQ_pt3	REQ_pt4	REQ_pt5	REQ_pt6	REQ_pt7	REQ_pt8	
Transport									
Biomaterials Transport									
Detoxification Submode				REQ_pt4		REQ_pt6		REQ_pt8	
Biofilter Submode						REQ_pt6		REQ_pt8	
Targetting									
Scanning									
Pattern Storage									
Transmission									
Configure System									
Lifesign Scanning									
Filtering									
Configure Biofilter									
Configure Hazadous Materials Filter									
Configure Operational Preferences									
Install and Initalize System									
Diagnostics and Built In Test									

Figure 6.19 *Requirements Traceability Matrix*

Practice: Defensive Design

"If we built houses the same way we build software, the first woodpecker would have destroyed civilization

- Grady Booch, IBM Fellow

Purpose

To improve the robustness of the software to unexpected events and conditions

Description

All software functionality depends on some assumptions (preconditional invariants). Defensive design codifies these assumptions and designs in the system behavior to ensure that these preconditional invariants are met at run-time, and if not, appropriate measures are activated.

Key points

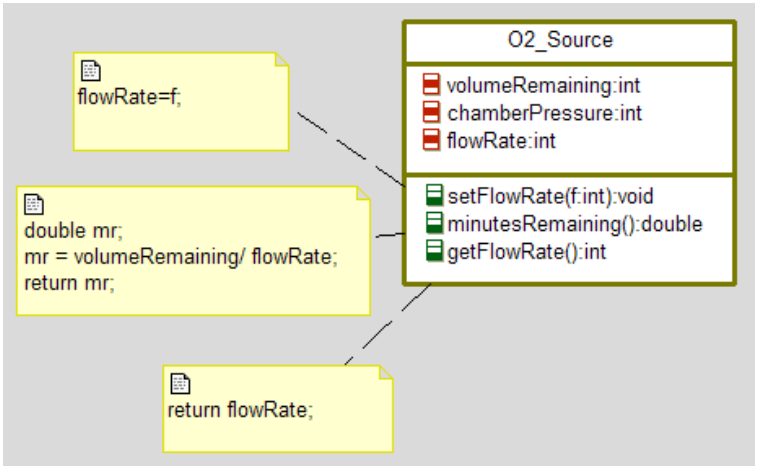
This practice does require additional compute cycles, but provides enhanced system robustness.

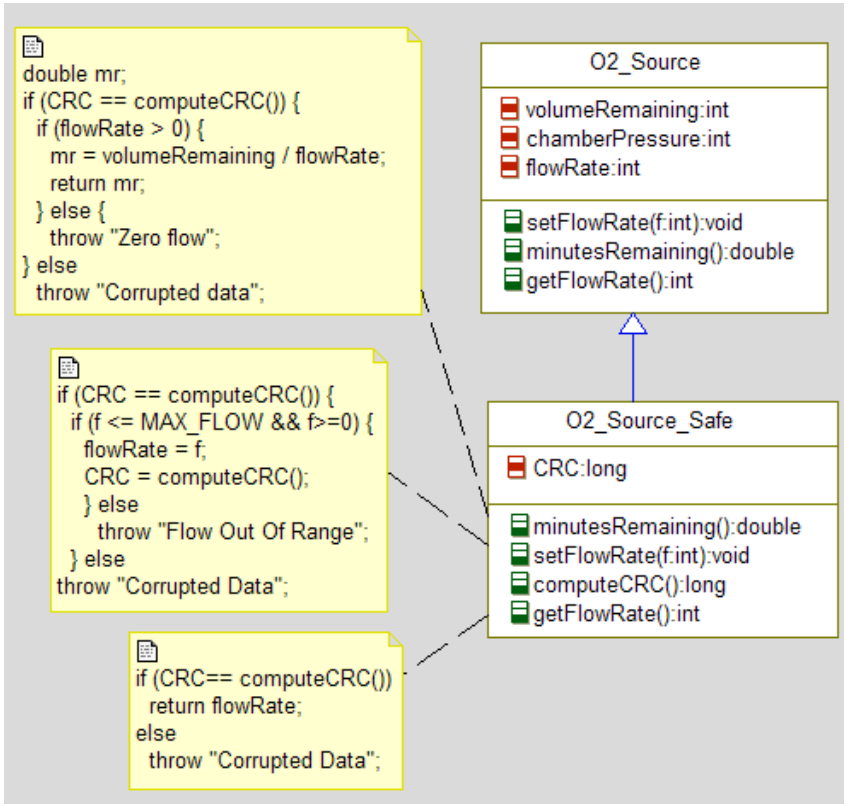
Examples

- Input parameters in range
- Output results in range
- Value sets are consistent
- Resources (ex. Memory) are available
- Object or system state is compatible with request
- CRCs are valid
- Exceptions / error return codes are caught / handled
- Determine and execute appropriate corrective measures if preconditions not met

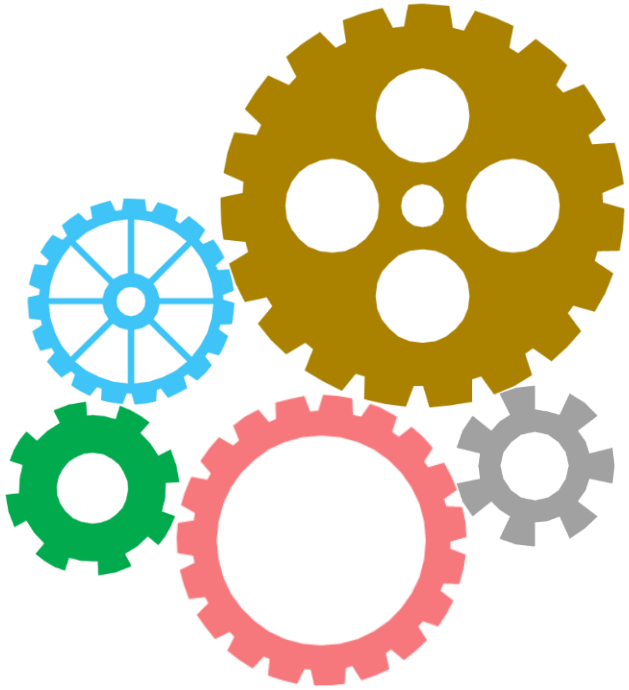
Redundancy "in the small" with Defensive Design

- Redundancy "in the small" adds low-level checking on
  - Data value range
  - Data value consistency
  - Computational accuracy
  - Explicit pre- and post-condition checks





Practice: 5 Views of Architecture



Purpose

To provide structure to the definition of architecture as a collection of design patterns from different aspects.

Description

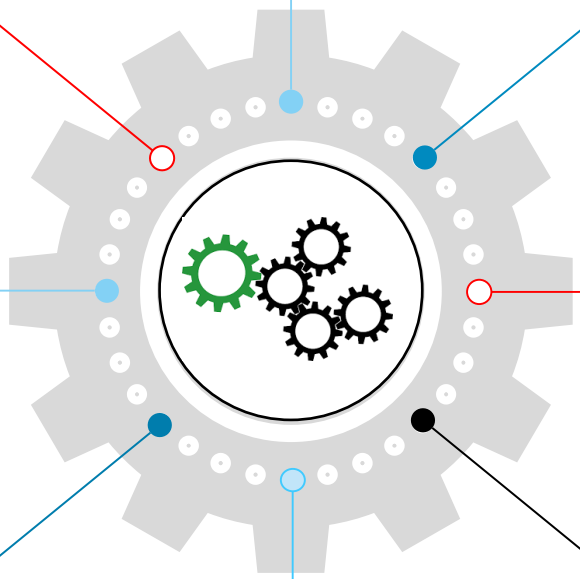
Architecture is the set of system-wide design optimization and organization decisions. In the Harmony process, it is divided up into 5 key views:

- Subsystem and Component
- Concurrency and Resource
- Distribution
- Dependability
- Deployment

Key points

- Avoid Big Architecture Up Front, but DO define your architecture as a set of interlocking design patterns from these different views
- Evolve your architecture by adding aspects as technical work items when needed in an iteration
- The architecture is rarely put completely in place before developers begin their work. **Architecture 0** is the name of the initial collection of implemented architectural concepts.
- The architectural concepts are represented as a set of Technical Work Items and put into the project backlog. Their priority reflects when that aspect is expected to be needed in the evolving product
- UML/SysML is a great way to capture architecture

Architecture Guidelines



Architecture provides a framwork into which design and code embeds. It should be available early, starting in iteration 0.

Architect early

Keep current

It is critical to update your architecture representation as it changes in the product. An old, incorrect architecture provides little value.

Evolve often

Some decisions can be delayed and others will turn out to be wrong. Evolve your architecture as needs arise. All architectures need refactoring from time to time.

Write it down

Don't rely on an oral history to document your architecture. Write it down or model it.

Simpler is better

Your architecture should be as simple as possible to meet all the needs – but no simpler than that.

Optimize

Architecfure, like design, is all about optimizing aspects of the system. And this means trade-offs. Many architectures can meet the same functional requirements.

Technical WI

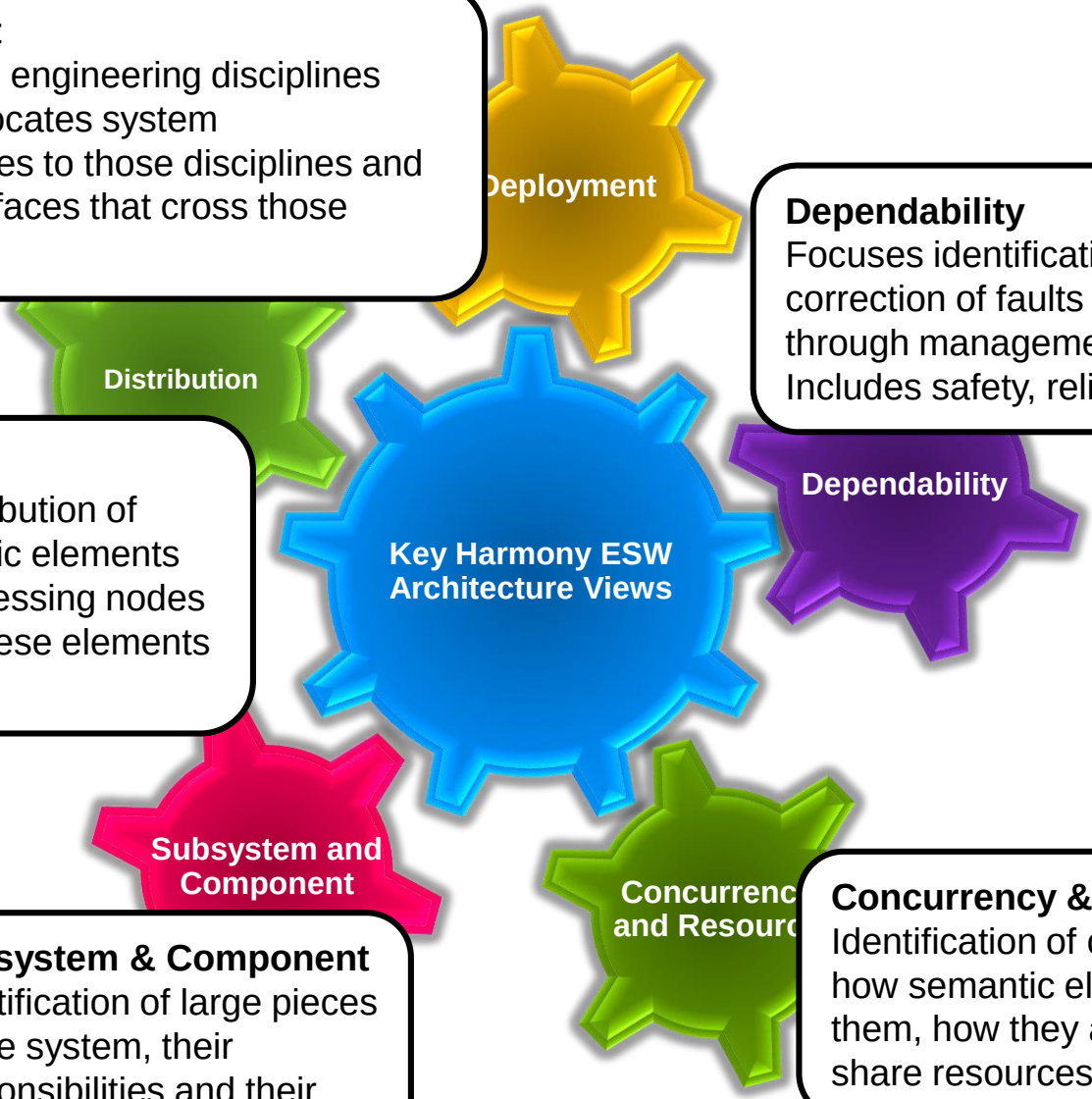
Addition and evolution of the architecture is planned, resulting in Techical Work Items. Evolution of the architecture introduces more TWIs over time.

Pattern-based

Architecture is the integration of many different aspects – called *architectural views*. These are defined in the selected architecture patterns.

Five Key Views of Architecture

Deployment
Identifies the engineering disciplines involved, allocates system responsibilities to those disciplines and defines interfaces that cross those boundaries



Dependability
Focuses identification, isolation and correction of faults as the system runs through management of redundancy. Includes safety, reliability, & security

Distribution
Focuses on the distribution of services and semantic elements across different processing nodes and identifies how these elements collaborate

Concurrency & Resource
Identification of concurrency units, how semantic elements map to them, how they are scheduled & share resources

Subsystem & Component
Identification of large pieces of the system, their responsibilities and their interfaces

Practice: 3 Schedules

Purpose

Schedules serve different purposes for different stakeholders. To this end, I propose we develop and maintain 3:

- The customer schedule (pessimistic, met 80% of the time)
- Working schedule (most accurate; met 50% of the time)
- Goal schedule (optimistic, may be incentivized; met only 20% of the time)

Description

For each estimated work item, get three estimates:

- $Est_{20\%}$
- $Est_{50\%}$
- $Est_{80\%}$

Compute the working schedule using EST_{best}

$$Est_{best} = \frac{Est_{20\%} + 4 * Est_{50\%} + Est_{80\%}}{6} * E_c$$

Customer schedule is constructed with $Est_{80\%} * E_c$

Goal schedule is constructed with $Est_{20\%} * E_c$

Key points

- Use absolute estimates on little things (2-8 hours)
- Use relative estimates on large things (user stores, use cases, epics)
- Use velocity to produce large scale estimates from relative measures when needed
- Remember: *All schedules are inaccurate*
- **Plan to replan** – update schedules and plans frequently, at least once per iteration
- "Optimistic" is just another word for "WRONG". Never use an optimistic schedule for planning purposes.

Planning Poker & Story Points

01

Purpose

Planning poker is a quick and easy design game for estimating effort for work items.

02

Description

1. The moderator will also need to prepare the list of use cases to size, and a set of planning cards to provide to each player.
    - The number of cards in this set depends on the number of estimating categories. Commonly the cards have values like 0, 1, 2, 3, 5, 8, 13, 20, 40 and 100
2. Estimation is performed:
    1. The estimators discuss the feature, asking questions of the product owner as needed.
    2. Each estimator privately selects one card to represent his or her estimate.
    3. When all participants have made their choice, all the cards are turned over at the same time.
    4. If all estimators selected the same value, that becomes the estimate. If not, the estimators discuss their estimates. The high and low estimators should especially share their reasons.
    5. Repeat this process until consensus is achieved or it is decided that more information is required.
3. Repeat for all items to be estimated

Practice: Broadened Definition of Done

P

Purpose

Provide all team members with a common understanding and expectation of what it means to complete each important work task, work item or work product

D

Description

It is crucial that all team members understand and adhere to what it means to be done with each important work task, work item or work product. This is best arrived at via team consensus of ideas.

Pr

Detailed procedure

1. Brainstorming session - No ideas are rejected out of hand, just recorded
2. Categorization session – how best to group the ideas into categories
3. Sorting and consolidation session
4. Definition of Done creation session

Ex

Example of Definition of Done

Work task: Release an implementation

- Unit/integration tests passed
- System verification tests passed
- Deployed on customer-accessible target platform
- Acceptance (validation) tests passed
- Release notes / customer documentation written
- No increase in technical debt

Accounting for stuff that isn't writing code

- The term **Definition of Done** is often applied solely to the increment produced by an iteration, however, there is benefit to using it in a broadened fashion
- All tasks should have a **Definition of Done**, and all work to achieve that state are included in the task estimate
  - Analyzing the user story and breaking it into requirements
  - Feature design
  - Documentation of design
  - Writing code
  - Write test cases
  - Executing test cases
  - Adding trace links among related elements (typically use cases, user stories, requirements, test cases, test results)
  - Performing refactoring
- Materials visible to customers – such as user documentation – should be prioritized, estimated and planned like a technical work item or use case

Download Papers, Presentations, Models, & Profiles for Free