

Architecture-led Incremental System Assurance (ALISA) Tutorial

Peter Feiler

Sept 2018

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213



Copyright 2018 Carnegie Mellon University. All Rights Reserved.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM18-1065

Outline

ALISA Background and Overview

ALISA Notations

Verifiable System Requirements

Verification Plans

Categories and Filters

Configuration and Execution of Assurance Cases

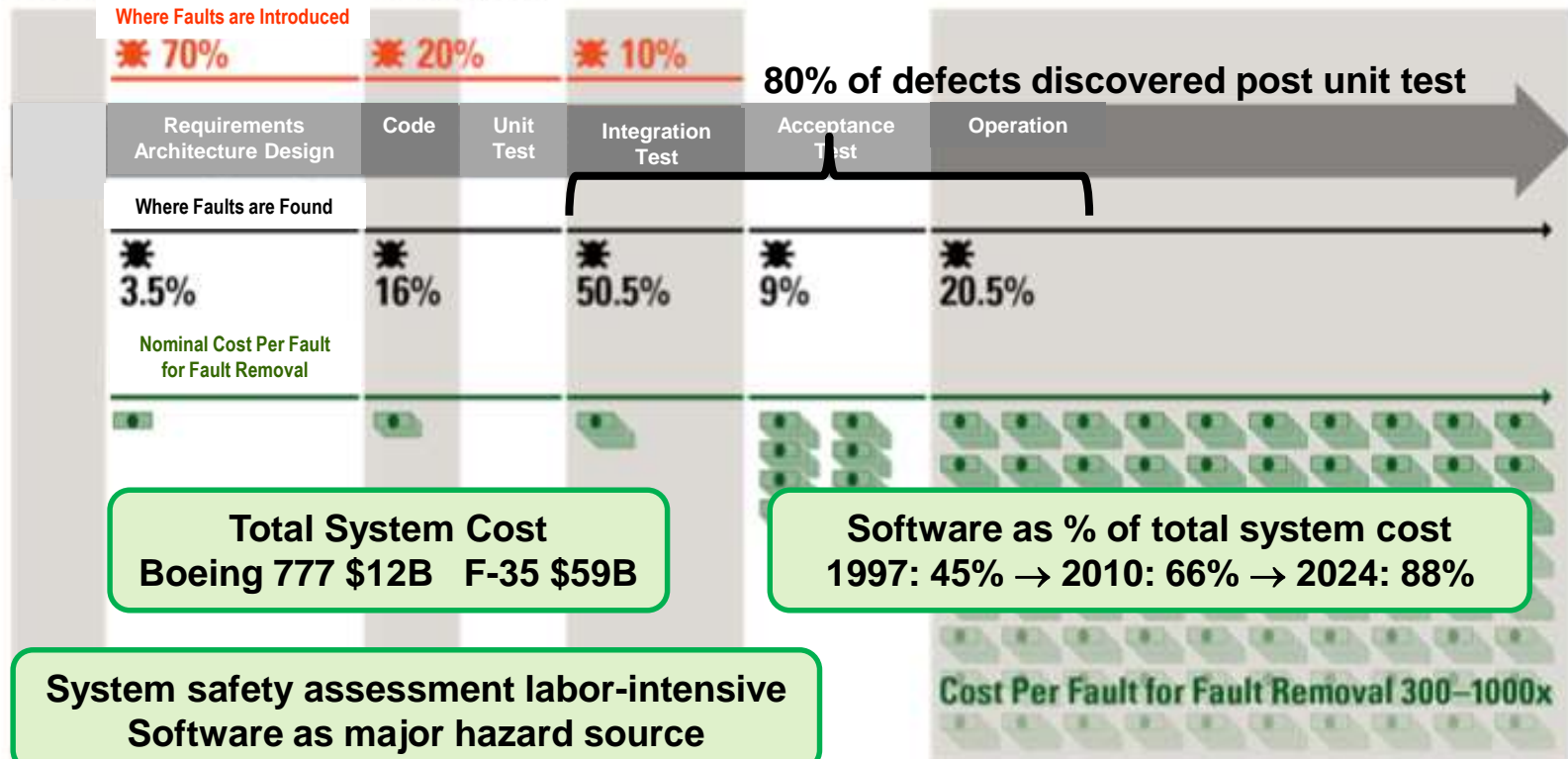
Reusable Global Requirements

Verification Method Registration

Stakeholder Goals and Requirements

Critical System Assurance Challenges

Software Development Life Cycle



Sources: Critical Code; NIST, NASA, INCOSE, and Aircraft Industry Studies

System safety assessment labor-intensive
Software as major hazard source

Post-unit test software rework cost
50% of total system cost and growing

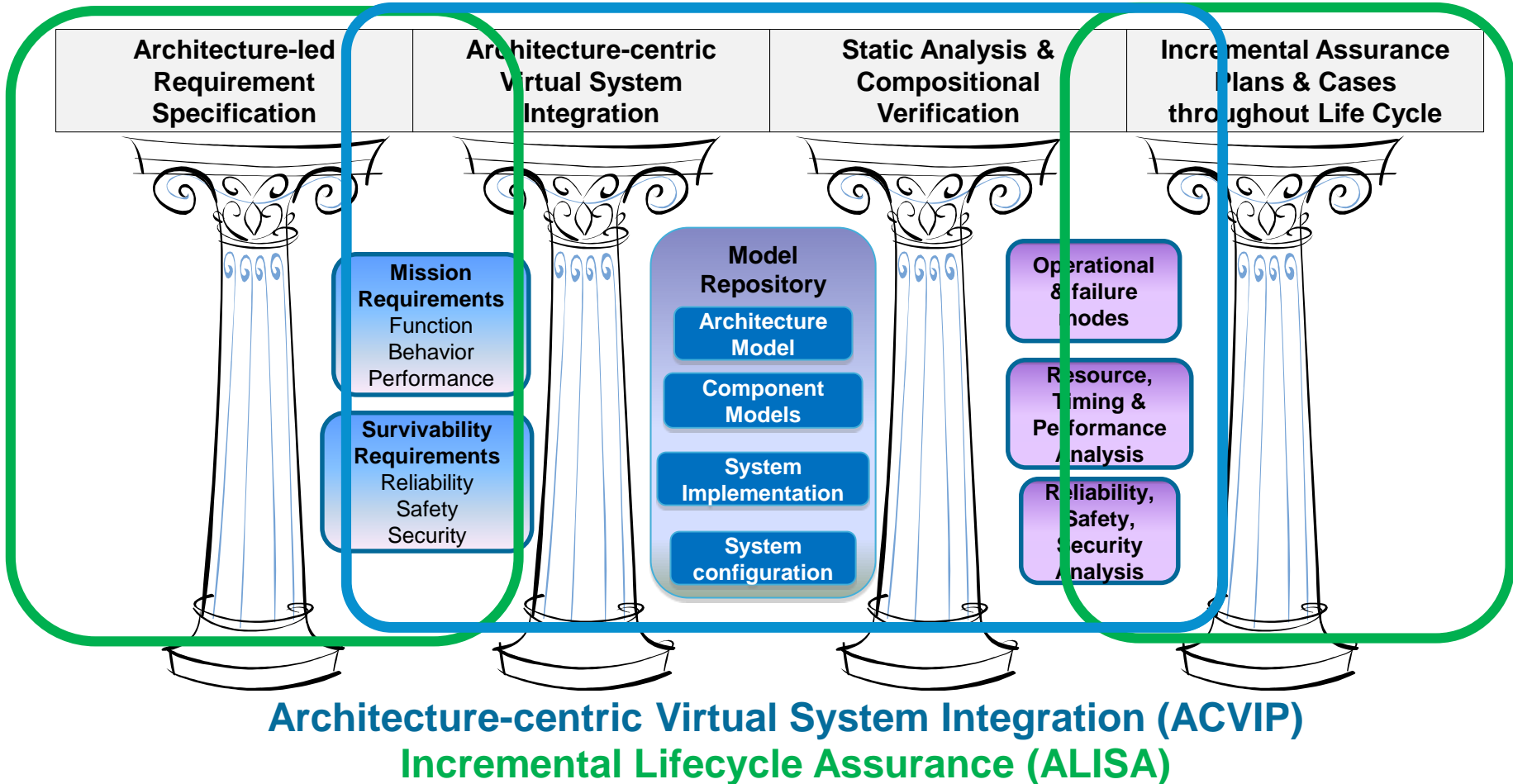
Recertification cost is not
proportional to system changes.

Assurance & Qualification Improvement Strategy



Assurance: Sufficient evidence that a system implementation meets system requirements

2010 SEI Study for AMRDEC
Aviation Engineering Directorate



Value of Requirement Uncertainty Awareness

Textual requirement quality statistics

- Current requirement engineering practice relies on stakeholders traceability and document reviews resulting in high rate of requirement change.

Requirements error	%
Incomplete	21%
Missing	33%
Incorrect	24%
Ambiguous	6%
Inconsistent	5%

NIST Study

Managed awareness of requirement uncertainty reduces requirement changes by 50%

- 80% of requirement changes from development team
- Expert requirement uncertainty assessment
 - Volatility, Impact, Precedence, Time criticality
- Focus on high uncertainty areas
- Engineer for inherent variability

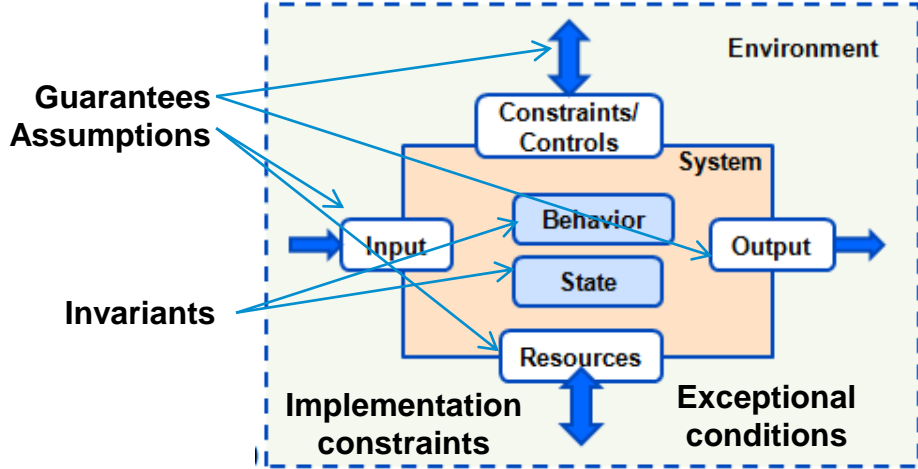
Selection	Weight	Precedence
Low Precedence	9	No experience of concept, or environment. Historically volatile
Medium Precedence	3	Some experience in related environments. Some historic volatility
High Precedence	1	Concept already in service. Low historic volatility

Figure 8. Precedence measurement scale

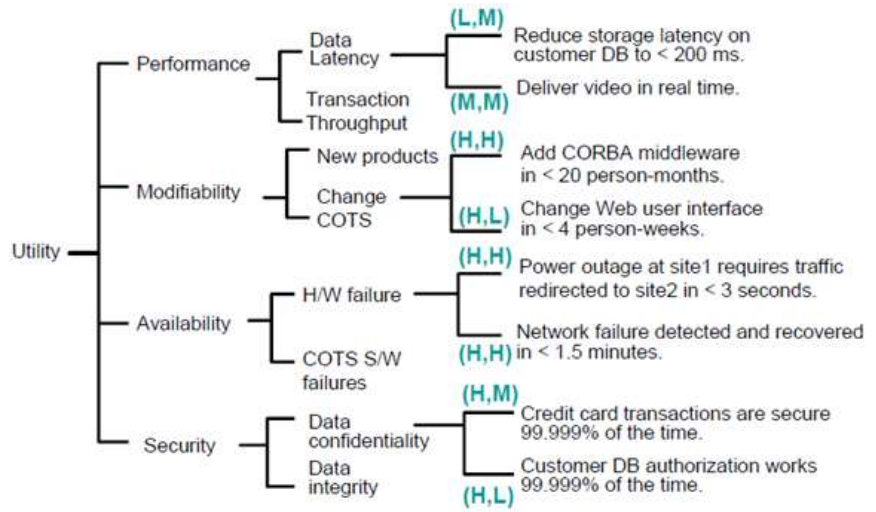
Rolls Royce Study

Three Dimensions of Requirement Coverage

System interactions, state, behavior



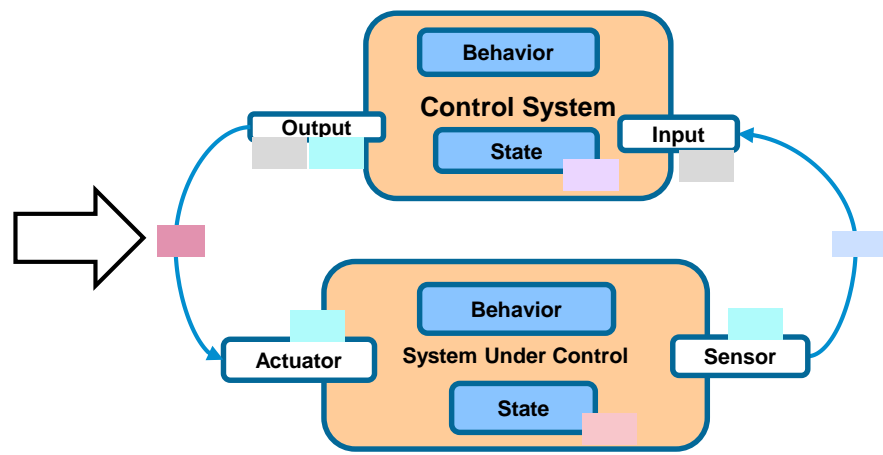
Design & operational quality attributes



Fault impact & contributors

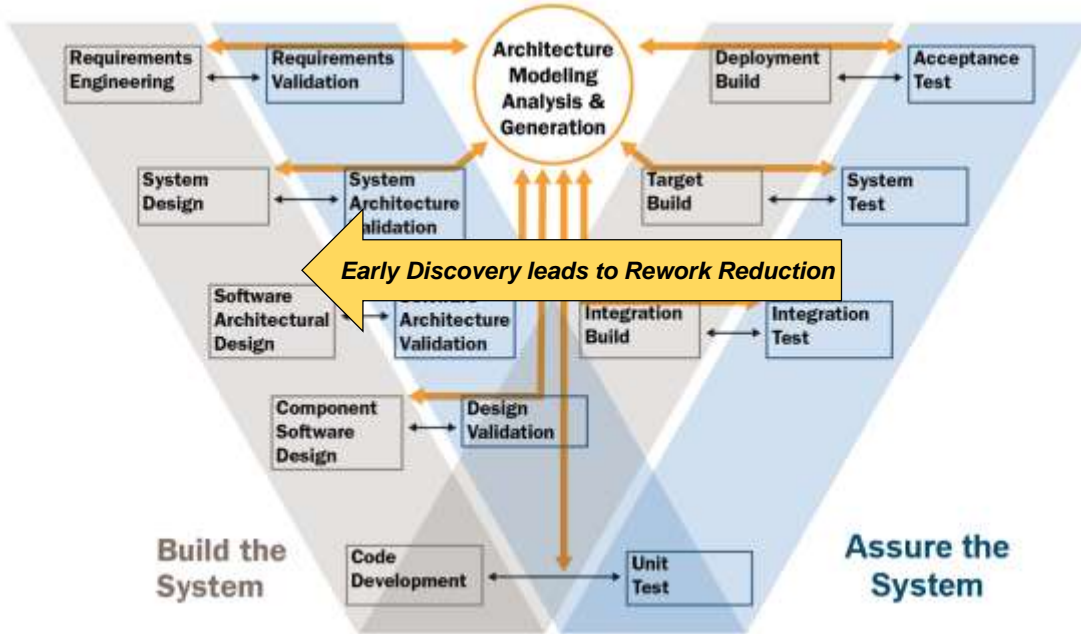
Omission errors	Commission errors
Value errors	Sequence errors
Timing errors	Replication errors
Rate errors	Concurrency errors
Authentication errors	Authorization errors

Fault Propagation Ontology

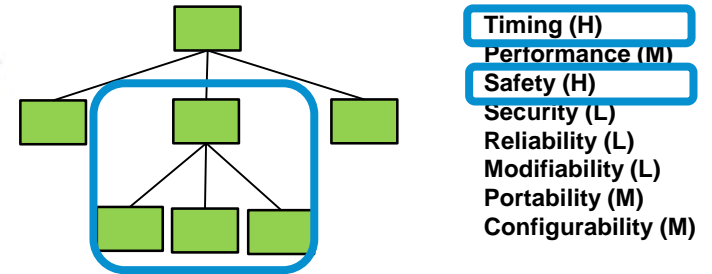


Three Dimensions of Incremental Assurance

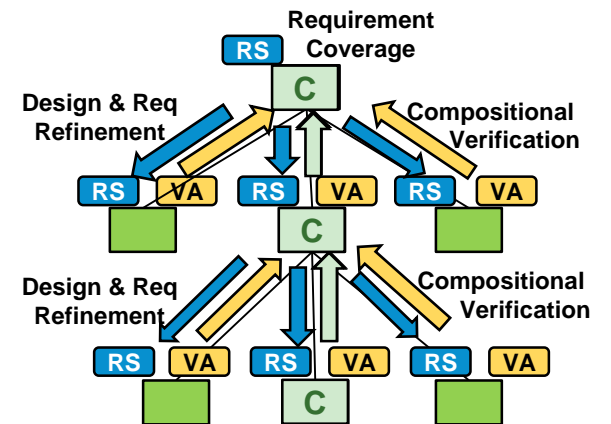
Incremental assurance through virtual system integration for early discovery
Return on Investment study by SAVI*



Priority focused architecture design exploration for high payoff
Measurable improvement (Rolls Royce)



Compositional verification and partitions to limit assurance impact



*System Architecture Virtual Integration (SAVI) Aerospace industry initiative

Incremental Lifecycle Assurance Benefits

Improved requirement quality through coverage and managed uncertainty

Improved evidence quality through compositional analytical verification

Measurably reduced certification related rework cost through virtual integration and verification automation

Outline

ALISA Background and Overview

ALISA Notations

Verifiable System Requirements

Verification Plans

Categories and Filters

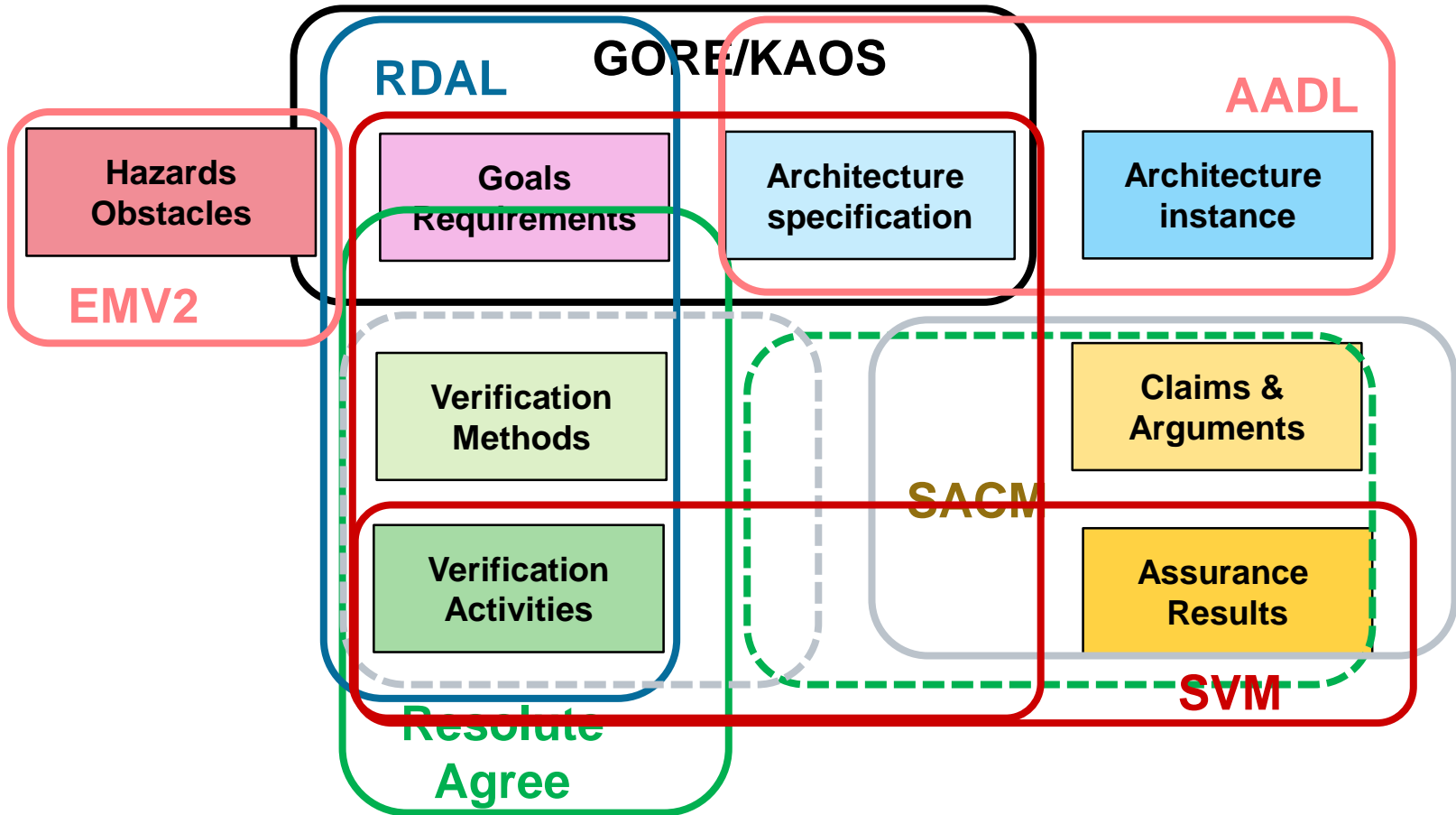
Configuration and Execution of Assurance Cases

Reusable Global Requirements

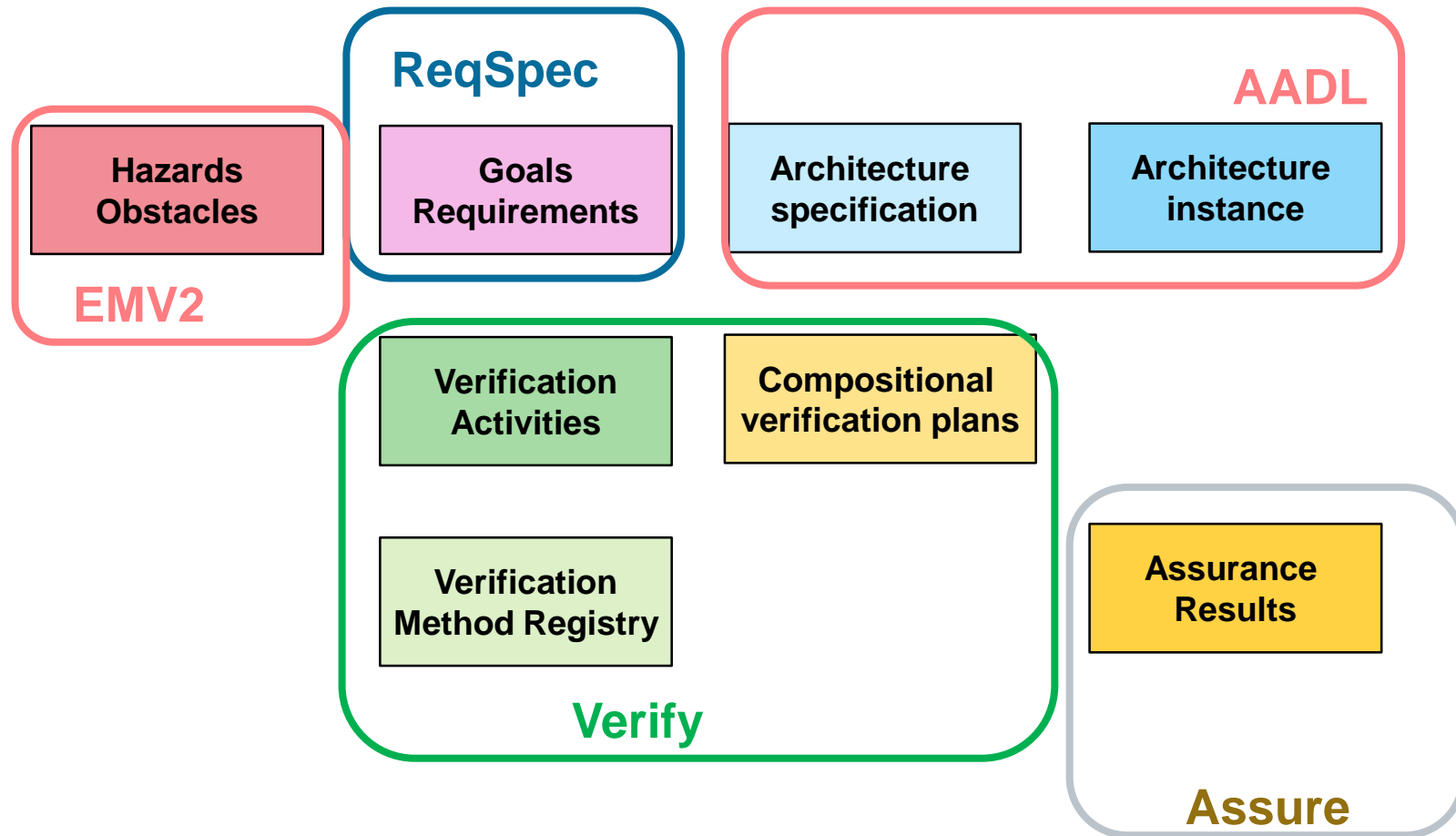
Verification Method Registration

Stakeholder Goals and Requirements

Relationship of Concepts/Meta models



ALISA Unified Concepts/Meta models



Verification and Computational Methods

Artifact: AADL models, Simulink models, source code

Available as: plugin, Java, Resolute/Lute, Junit tests

Java as wrapper method for external tools via Eclipse mechanisms

Modeling Notations in ALISA V1

ReqSpec: Represent stakeholder goals, system requirements

- Document-based & architecture-led
- Verifiable system requirements
- Coverage and uncertainty

Verify: Verification plans of verification activities against artifacts

- Reasoning logic of how verification activities satisfy requirement
- Via verification methods (manual, automated) on models/code
- Assumptions, preconditions on verification methods

Alisa: Composition of verification plans into assurance cases

- Verification of AADL model artifacts
- Across layers of system architecture
- Assurance tasks as filtered views of assurance plans

Assure: Manage assurance case instance execution and results

- Multi-valued logic evaluation of verification action & results
- Acceptable risk factors (e.g., design assurance levels)
- Filtered execution of assurance plans (based on category tags)

Outline

ALISA Background and Overview

ALISA Notations

Verifiable System Requirements

Verification Plans

Categories and Filters

Configuration and Execution of Assurance Cases

Reusable Global Requirements

Verification Method Registration

Stakeholder Goals and Requirements

Mixture of Requirements & Architecture Design Constraints

Textual Requirements for a Patient Therapy System

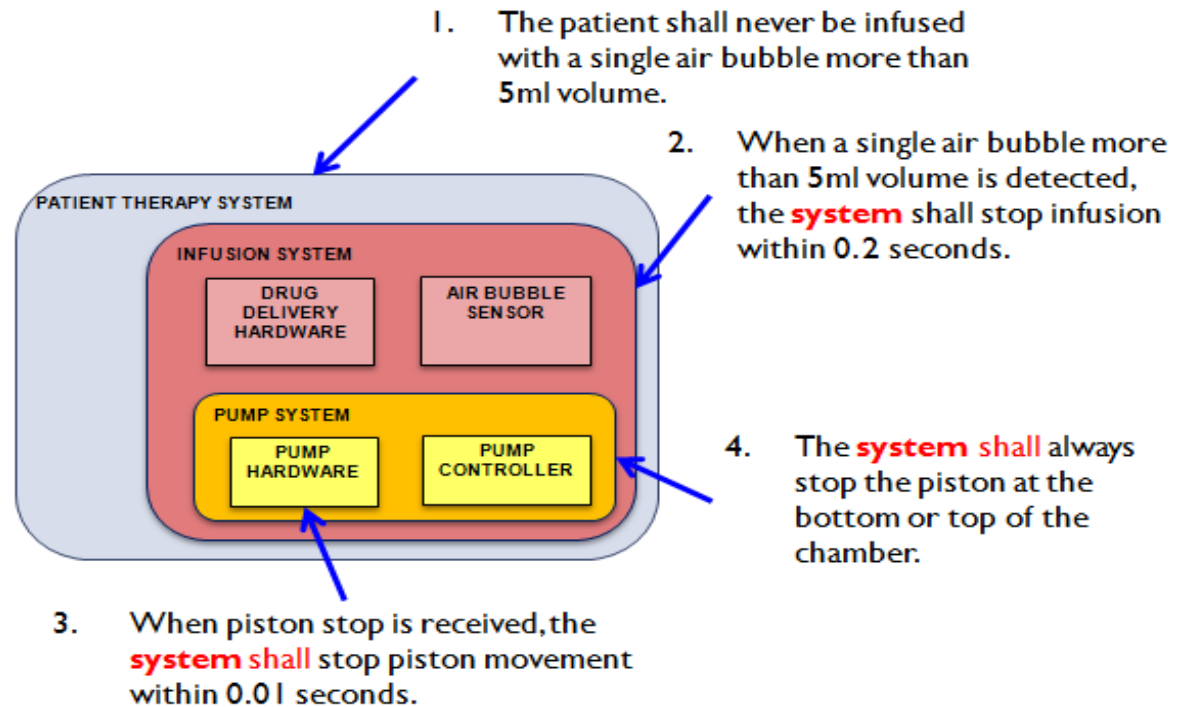
The patient shall never be infused with a single air bubble more than 5ml volume.

When a single air bubble more than 5ml volume is detected, the **system** shall stop infusion within 0.2 seconds.

When piston stop is received, the **system shall** stop piston movement within 0.01 seconds.

The **system shall** always stop the piston at the bottom or top of the chamber.

Same Requirements Mapped to an Architecture Model



Importance of understanding system boundary

We have effectively specified a system partial architecture

U Minnesota Study

ReqSpec: Textual Notation for Draft Requirement Definition & Analysis Language (RDAL) Annex

Objective

- Goal-oriented architecture-led requirement specification
- Explicit record of requirement decomposition, refinement, and evolution
- Integration with safety analysis via Error Model Annex V2
- Basis for incremental life-cycle assurance of evolving systems
- Bridge to existing requirements documentation practices

Stakeholder Goals

- Descriptions of intent that may be in conflict with each other
- Use cases: UML/SysML or ITU UCM

System Requirement Specification

- Specification of a consistent set of system requirements that must be met by a delivered system
- Basis for verification and assurance plans

System Requirement Specification Notation

Requirement specification

- Verifiable contract via verification plans
 - Satisfiable design goal
- Associated with AADL classifier or element in classifier
- Description, rationale
- Parameterization by variable
- Refinement, decomposition, inheritance, evolution
- Verifiable by verification activities
- Formal specification of predicate
- Traceability to goals, hazards
- Traceability to DOORS requirement and other documents

Requirement Sets

System Requirement Set

- One per component classifier (identified by **for** clause)
- List of system requirement declarations
- Inherited requirements according to extends hierarchy
- Inclusion of individual global requirements or global requirement sets

Global Requirement Set

- Reusable set of requirements
- For inclusion in system requirement sets
- Configurable into assurance plans

Minimize Impact of Frequent Requirement Changes

```
system requirements scsreqs for SimpleControlSystem::SCS [
  val MaximumLatency = 20 ms
  requirement R1 : "SCS weight limit" [
    val MaximumWeight = 1.2 kg
    category Quality.Mass
    description this " shall be within weight of " MaximumWeight
    // second condition verifies that MaximumWeight is same as the property value WeightLimit
    value predicate MaximumWeight == #SEI::WeightLimit
    see goal SCSgoals.ng2
  ]

  requirement R2 : "SCS sensor to actuator response time limit" [
    description this " shall have a sensor to actuator response time within " MaximumLatency
    category Quality.Latency
    see goal SCSgoals.g1
  ]

  verification plan scsvplan for scsreqs
  [
    claim R1 [
      activities
      actualsystemweight : Plugins.MassAnalysis()
      [ category Quality.Mass ]
      MaxWeight : Resolute.verifySCSReq1(MaximumWeight in kg) [ category Quality.Mass ]
    ]
  ]
]
```

Val constants can be defined for requirement set or individual requirements.

Val constants can be used in descriptions, predicates, and as parameters in verification activities

AADL Model as System Specification

```
system SCS
  features
    power: in feature PhysicalResources::Power;
    force: out feature;
  modes
    operational: initial mode;
    standby: mode;
  properties
    ACVIP::Aliases => ("SCS", "Simple Controller");
    SEI::PowerBudget => 2.5 w applies to power;
    SEI::WeightLimit => 1.2 kg;
    Physical::Voltage => 12.0V applies to power;
  annex emv2 (**
    use types ErrorLibrary;
    use behavior ErrorLibrary::FailStop;
    error propagations
      force: out propagation {ServiceOmission};
      flows
        es1: error source force;
    end propagations;**)
end SCS;
```

```
value predicate MaximumWeight == #SEI::WeightLimit
```

Scenario 1

Property association value changed by user
Referenced in description instead of val variable

Scenario 2

Val variable value automatically set as property
value for certain verification methods

Scenario 3

Consistency between val variables and
property values
Consistency check: explicit or as part of
verification method registry

Requirements Decomposition

Derive requirements on subcomponents from system requirements

- Associate requirement with subcomponent in the system implementation
 - Context specific requirement
 - Requirements on subsystem to be acquired (purchased, contracted out)
- Requirement on classifier of subcomponent
 - applies to all subcomponent instances referencing the classifier
 - Represented by property of subcomponent classifier
 - System specification used in implementation to verify that design meets system requirement

Outline

ALISA Background and Overview

ALISA Notations

Verifiable System Requirements

Verification Plans

Categories and Filters

Configuration and Execution of Assurance Cases

Reusable Global Requirements

Verification Method Registration

Stakeholder Goals and Requirements

Verification & Assurance Plan Notation

Method registry

- Reusable verification methods on models and other artifacts
- Method categories for filtering and coverage
- Currently supported: Java, Resolute, analysis plugin, manual

Verification plans

- Claims aligned with system requirement specifications
- Multiple verification activities on different artifacts
- Implementation and use verification
- Conditional and backup activities (pass, fail, error)
- User definable selection categories
- Description, Rationale

Compositional assurance plans and tasks

- Configuration of assurance responsibility
- Focused assurance view based in category filters

Verification Plans

Shared Predicates and “Compute” Variables

```
requirement R3 : "SCS inlet voltage" for power [  
  val volts = 12.0 V  
  compute actualvolt: Physical::Voltage_Type  
  value predicate volts == actualvolt  
  see goal SCSgoals.g3  
]
```

Requirement to be verified identifies target model element

```
claim R3 [  
  activities  
    hasvoltage: Alisa_Consistency.ConsistentVoltage(volts)  
    consistentvoltage: actualvolt = Alisa_Consistency.GetVoltage()  
    matchvoltage: actualvolt = Alisa_Consistency.SetGetVoltage()property values (volts)  
  ]  
]
```

Use of predicate (T/F) verification method
Use of compute verification method
Verification method with property values

```
claim R2_Lat [  
  activities  
    responsetime : Plugins.FlowLatencyAnalysis()  
    timing: Plugins.ResourceAllocationScheduling()  
  assert timing then responsetime  
]
```

Verification activity with assertion logic

Verification Plans

Shared Predicates and “Compute” Variables

```
requirement R3 : "SCS inlet voltage" for power [  
  val volts = 12.0 V  
  compute actualvolt: Physical::Voltage_Type  
  value predicate volts == actualvolt  
  see goal SCSgoals.g3  
]
```

Think of “compute” variable as free variable in lambda expression.

```
claim R3 [  
  activities  
    hasvoltage: Alisa_Consistency.ConsistentVoltage(volts)  
    consistentvoltage: actualvolt = Alisa_Consistency.GetVoltage()  
    matchvoltage: actualvolt = Alisa_Consistency.SetGetVoltage()property values (volts)  
  ]  
]
```

Verification activity result is bound as compute value

```
method ConsistentVoltage (feature, voltage: Physical::Voltage_Type in V)  
  : "Ensure Voltage property value is consistent with required voltage value" [  
    java alisa_consistency.ModelVerifications.hasVoltage  
    description "Verify that the Voltage property has the same value as specified in"  
  ]  
  
method GetVoltage (feature) returns (volts: Physical::Voltage_Type in V )  
  : "Ensure Voltage property value is consistent with required voltage value" [  
    java alisa_consistency.ModelVerifications.getVoltage  
    description "Verify that the Voltage property has the same value as specified in"  
  ]
```

Verification Activities

Multi-valued verification activity results

- Verification activity result states
 - Success, fail, error, tbd
- Compositional Argument Expressions
 - **All** [va+]: a collection of independent Vas
 - Va1 **then** Va2: Execution of Va2 dependent on success of Va1
 - Va1 **else** Va2: Execute Va2 only if Va1 produces negative result
 - Va1 **else** [fail: Va21 timeout: Va22 error: Va23]

Mode specific verification activities

Parameterized verification activities

- Data sets as input parameters

Why And/Or Assurance Logic is Insufficient

- Assurance case: focus on final result
 - Multiple pieces of evidence must be satisfied (“and”)
 - Design alternatives, different operational modes (“or”)
- Resolute experience during development
 - “and”: if first verification action fails, others are not evaluated
 - Equivalent to compiler stops on first error
 - “or”: any satisfied verification action is fine. Pick the cheapest one to execute
 - “implies”: failing first verification action results in success

Registry of Verification Methods

Uniform signature definition for different verification method implementations

Verification Method Registry Implementation

Existing OSATE Analysis plugins

Reflective execution of Java methods

Execution of Python Scripts (new)

Resolute claim functions

Execution of Junit-based code tests

Java Based Verification Methods

Method path starting with OSATE workspace

```
method ConsistentVoltage (feature, voltage: Physical::Voltage_Type in V)
: "Ensure Voltage property value is consistent with required voltage value" [
  java alisa_consistency.ModelVerifications.hasVoltage
  description "Verify that the Voltage property has the same value as specified in the requirement"
]
```

Predicate verification methods returns true or false

```
method GetVoltage (feature) returns (volts: Physical::Voltage_Type in V )
: "Return the voltage value of the feature in specified unit" [
  java alisa_consistency.ModelVerifications.getVoltage
  description "Verify that the Voltage property has the same value as specified in the requirement"
]
```

Compute method returns values

```
method SetGetVoltage (feature) properties(Physical::Voltage) returns (volts: Physical::Voltage_Type in V )
: "Ensure Voltage property value is consistent with required voltage value" [
  java alisa_consistency.ModelVerifications.getVoltage
  description "Verify that the Voltage property has the same value as specified in the requirement, and"
]
```

Sets property values for analysis

Access to Java methods in OSATE via Plugin dependency

Extension point to register Java class (new)

Other Verification Methods

OSATE Analysis Plugins are registered in AlisaPredefined/Plugins.methodregistry

```
method FlowLatencyAnalysis: "Verify end to end flow latency (max/min/jitter)" [  
  category Quality.Latency  
  description "Analysis of end-to-end flows reporting on latency (max/min/jitter).  
For details see https://wiki.sei.cmu.edu/aadl/index.php/Latency\_Analysis "  
  plugin FlowLatencyAnalysis  
]
```

Resolute Claim function path starting with package

```
method verifyDCSReq1 (max: real ): "Verify DCS weight is within specified maximum (Req1)" [  
  resolute BudgetResolute.BudgetResolute_public.Resolute.Resolute.DCSReq1  
]
```

Python script path within OSATE workspace without file extension

```
method DoCount : "Count model elements" [  
  python SimpleControlSystem.Python.modelstatistics2  
]
```

Outline

ALISA Background and Overview

ALISA Notations

Verifiable System Requirements

Verification Plans

Categories and Filters

Configuration and Execution of Assurance Cases

Reusable Global Requirements

Verification Method Registration

Stakeholder Goals and Requirements

User-extensible Categories

Categories on requirements and verification activities/methods for filtering and coverage

Can be organized into category sets

Users can add their own categories

Predefined categories can be found in
AlisaPredefined/predefined.cat

```
predefined.cat
1=Quality [
2   Behavior State Performance
3   Latency Timing Security Safety Reliability Availability
4   CPUUtilization MemoryUtilization
5   NetworkUtilization Mass ElectricalPower
6 ]
7=Phase [
8   StakeholderRequirements SystemRequirements PDR CDR
9   ArchitectureDesign DetailedDesign Implementation UnitTest
10  SystemTest
11 ]
12=Layer [
13  Tier1 Tier2
14  Tier3 Tier4 Tier5
15 ]
16=Kind [
17  Assumption Guarantee Consistency Constraint Exception
18 ]
```

Outline

ALISA Background and Overview

ALISA Notations

Verifiable System Requirements

Verification Plans

Categories and Filters

Configuration and Execution of Assurance Cases

Reusable Global Requirements

Verification Method Registration

Stakeholder Goals and Requirements

Assurance Plans and Cases Execution

Assurance plan

- Configuration of assurance case:
 - how much of architecture to be verified,
 - which component specific and reusable verification plans to be included

Assurance plan instantiation & execution

- Automated verification activity execution
- Execution states: To be done, in progress, done, Redo
- Tracking of result state and reports

Assurance Task

- Filtered assurance plan instances based on requirement, verification, and verification activity selection categories

Assurance Case & Plan

Assurance case defines which subsystems are to be verified

Multiple assurance plans for AADL models of different fidelity

Specification of which subsystems are assumed to have been verified separately

Assurance task = category filter

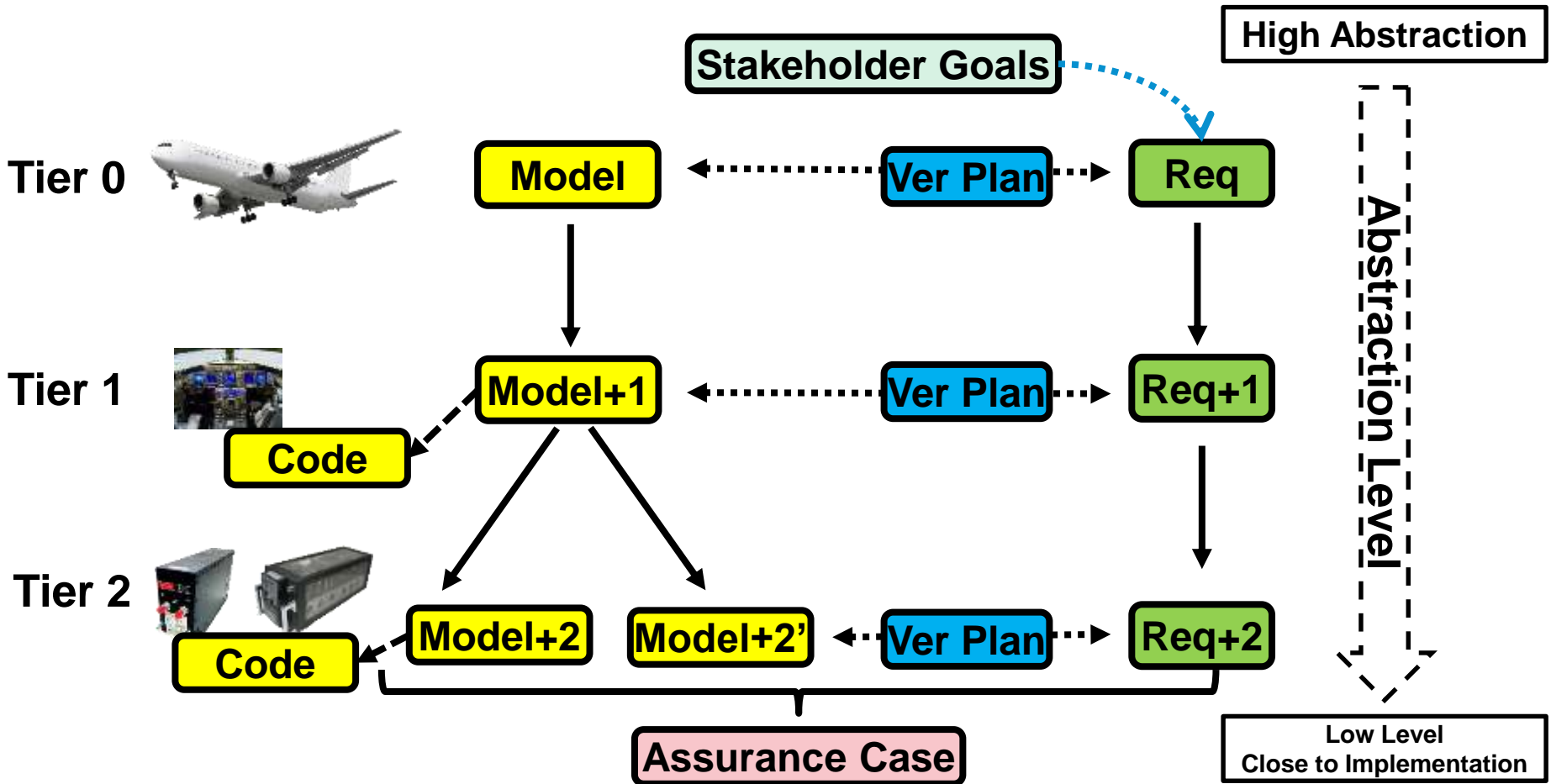
- Reusable category filters can be defined in .cat files

```
assurance case SCSCase for SimpleControlSystem::SCS [  
  assurance plan SCSPlan for SimpleControlSystem::SCS.tier1 [  
    description "my plan"  
    assure scsvplan scsimplvplan  
    assure global globalVPlan  
    assure subsystem dcs  
    assume subsystem sensor1 sensor2 actuator  
  ]  
  
  assurance task SCSTask [  
    category Quality.Mass  
  ]  
]
```

Automated Incremental Assurance Workbench



Identify Assurance Hotspots throughout Lifecycle



Original Aircraft Virtual Integration Demo

Aircraft: (Tier 0)

Aircraft system: (Tier 1)
 Engine, Landing Gear, Cockpit, ...
 Weight, Electrical, Fuel, Hydraulics,...

LRU/IMA System: (Tier 2)
 Hardware platform, software partitions
 Power, MIPS, RAM capacity & budgets
 End-to-end flow latency

System & SW Engineering:
 Mechatronics: Actuator & Wings
 Safety Analysis (FHA, FMEA)
 Reliability Analysis (MTTF)

Subcontracted software subsystem: (Tier 3)
 Tasks, periods, execution time
 Software allocation, schedulability
 Generated executables

OEM & Subcontractor:
 Subsystem proposal validation
 Functional integration consistency
 Data bus protocol mappings

Repeated Virtual Integration Analyses:
 Power/weight
 MIPS/RAM, Scheduling
 End-to-end latency
 Network bandwidth

Proof of Concept Demonstration and Transition by Aerospace industry initiative

- Propagate requirements and constraints
 - Higher level model down to suppliers' lower level models
 - Verification of lower level models satisfies higher level requirements and constraints
- Multi-tier system & software architecture (in AADL)
 - Incremental end-to-end validation of system properties

Assurance Case Execution & Results

Assurance View can be found under window/Show view/Other/Alisa/Assurance view

On the left it shows all assurance cases in the workspace

Users can select a filter for a subset view

The context menu lets you select and execute an assurance case

On the right we see the assurance case instance with a plan/claim/evidence, and result details

The counts provide insight into where the problem areas are

Evidence	Pass	Fail	Error	Todo	Description
Plan SCSPlan(SCS.tier1)	33	25	15		
Claim R1	2	4			shall be within weight of 1.2 kg
Evidence actualsystemwei 1			1		Perform full weight (mass) analysis. This includes net/gross we...
Evidence MaxWeight		2			sum of all subcomponent weights 1.2000000000000002 kg wi...
Failure					Failure: SCS_tier1_Instance : SimpleControlSystem::SCS.tier1All...
Validation verifyAllWei			1		
Evidence SCSReq1act		2			R1: SCS shall be no heavier than 1.2 kg[F]
Failure					assured sum of subcomponent budgets within budget
Failure					sum of all subcomponent weights 1.2000000000000002 kg wi...
Validation verifySCSRe			1		
Predicate	1				(MaximumWeight == #SEI::WeightLimit)

Assurance Case Execution and Metrics

Assurance Metrics

- Requirement coverage measures
- Multi-valued verification result measures and their aggregates
 - Pass, fail, incomplete, conditions, backups
- Weighted requirement claims, verification activity results
 - Reflect importance, uncertainty, effectiveness

Guidance throughout life cycle

- Measurement based uncertainty areas (hotspots)
- Change impact prediction

Outline

ALISA Background and Overview

ALISA Notations

Verifiable System Requirements

Verification Plans

Categories and Filters

Configuration and Execution of Assurance Cases

Reusable Global Requirements

Stakeholder Goals and Requirements

Global Reusable Requirements

```
global requirements globalReq
[
  requirement connected : "All features of a component are connected"[
    description "All features of a component are connected."
    when alisa consistency.ModelConditions.isLeafComponent()
    category Kind.Consistency
    development stakeholder sei.phf
  ]

  requirement Allconnected : "All features of all components are connected"[
    description "All features of leaf components are connected."
    category Kind.Consistency
    development stakeholder sei.phf
  ]
]

include Peter.req1 for self
include globalReq.Allconnected for self
include globalReq.connected

assurance case SCSCase for SimpleControlSystem::SCS [
  assurance plan SCSPlan for SimpleControlSystem::SCS.tier1 [
    description "my plan"
    assure scsvplan scsimplvplan
    assure global globalVPlan]
```

Reusable requirements, typically consistency constraints, completeness checks, or security issues.

Connected: the requirement is applied only to components that satisfy when condition

Inclusion in system requirement sets.
Applicable for component itself or recursively for all subcomponents.

Allconnected: the verification method is assumed to recursive the component hierarchy.

Global reusable verification plan configured into an assurance case

Outline

ALISA Background and Overview

ALISA Notations

Verifiable System Requirements

Verification Plans

Categories and Filters

Configuration and Execution of Assurance Cases

Reusable Global Requirements

Stakeholder Goals and Requirements

Use Scenarios

External stakeholder requirement documents

- Leave external: requirements reference document entry (**see document**)
- Import as goal document
- Capture concepts, functions, and architecture elements in AADL model
 - Basis for a system requirement specification

Use Scenarios

External system requirement documents

- Leave external
 - Capture concepts, functions, and architecture elements in AADL model
 - Create reqspec requirement proxies with reference to document entry (**see document**)
 - Annotate requirement with model reference (**for**)
- Import as requirement document
 - Capture concepts, functions, and architecture elements in AADL model
 - Annotate requirement with model reference (**for**)

Resulting ReqSpec specification is basis for verification and assurance plans

Stakeholder Goals Notation

Goal specification

- Associated with AADL model element
- Description, rationale
- Refinement, evolution, conflict resolution
- Traceability to stakeholders and DOORS requirement and other documents

- Document focused goal organization
 - Goals in document sections
 - Incrementally evolve model and identify model elements
- Model focused goal/requirement organization
 - Requirements for a model component

Names, Concepts, and Systems

```
document SCSgoals : "SCS stakeholder goals" [  
  description "This document contains the stakeholder requirements for the Simple Control System (SCS).  
  The SCS provides control for a simple device (SD).  
  The SCS system consists for software, hardware, and physical components."  
  section SystemFunctionality : "System Functionality" [  
    goal g1: "Feedback Control" [  
      [ description "The simple controller (SC) shall provide stable feedback control of the SD."  
        rationale "The SD is a safety critical device that cannot tolerate erratic behavior."  
        stakeholder sei.phf  
      ]  
    ]  
    goal g2: "Digital Feedback Control" [  
      description "The SCS system shall control the SCS device with a digital controller"  
    ]  
    goal g3: "electrical power"  
    [ description "The simple control system shall be supplied with 15V electrical power."  
      stakeholder sei.phf  
    ]  
  ]  
  section NonfunctionalProperties : "Nonfunctional system requirements"  
  [  
    goal ngl : "Safety" [  
      description "The system shall be safe"  
      rationale "This is a control system, whose failure affects lives. "  
      stakeholder sei.phf sei.dpg  
    ]  
    goal ngl_1: "Physical damage"[  
      refines ngl  
      description "The controller shall not cause the simple device to damage objects"  
    ]  
  ]  
]
```

Are the following the same:

Simple Control System

Simple Controller

System

Controller

We define SCS as system type.

As we add requirements we add features, e.g., power.

As we define a model of the operational context we are explicit about the necessary and possibly missing interactions.

Requirement Traceability Reports

BIRT based prototype

- Traceability to stakeholders
- Traceability between requirements and goals
- Context command invoked on instance model, reqspec/goal file
- Source code available as org.osate.reqtrace
- Extensible for additional report templates

ReqSpec compiler validations

- Reported via Eclipse Diagnostics/Markers
 - Missing stakeholder, missing goal, duplicate goal/requirement
 - Cycles in refine hierarchy
- Report generation from Eclipse Markers

References

ALISA

Online documentation (also built into OSATE):

<https://rawgit.com/osate/alisa/develop/org.osate.alisa.help/contents/00-Main.html>

Papers: <https://hal.archives-ouvertes.fr/hal-01289468/document> and <http://sigada.org/conf/hilt2016/paper-McGregor.pdf>

ReqSpec

SEI TR: <http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=464370>

JMR Shadow project:

http://resources.sei.cmu.edu/asset_files/specialreport/2015_003_001_447187.pdf

Model Examples

<https://github.com/osate/alisa-examples>