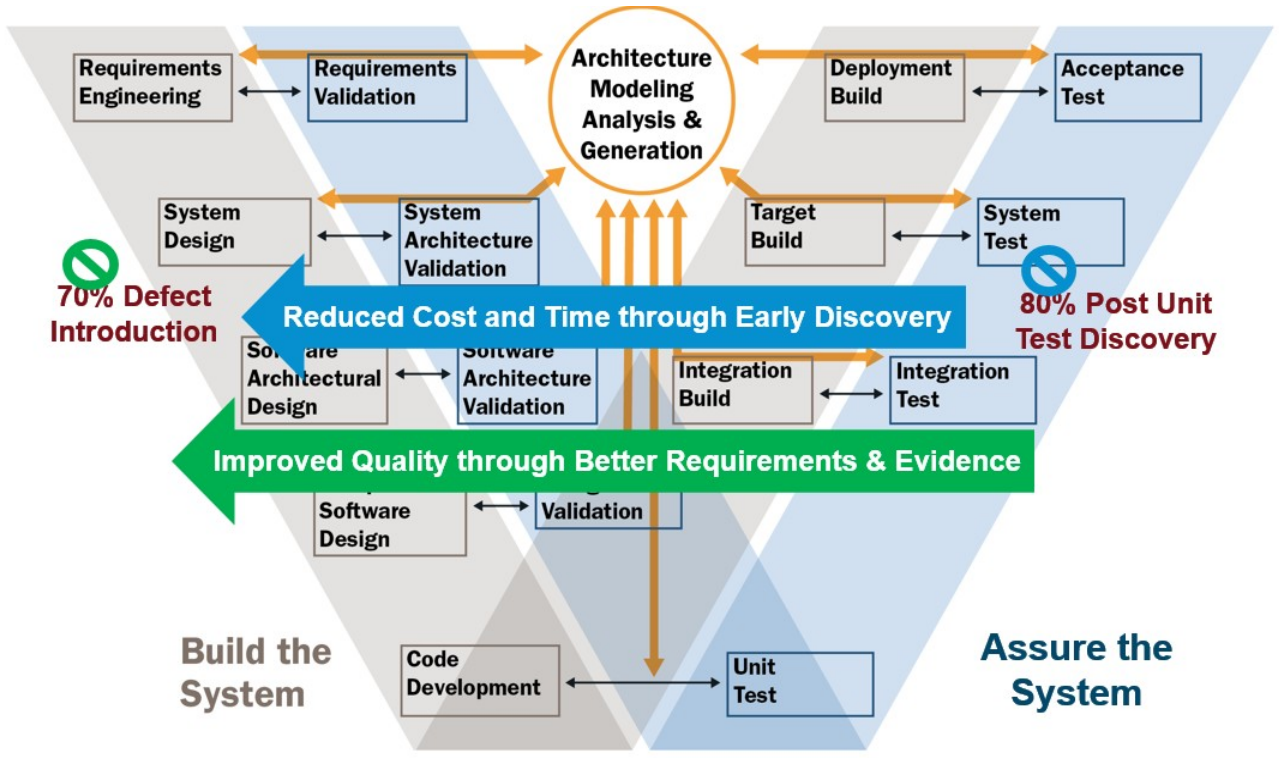
Introduction

Architecture-led Incremental System Assurance (ALISA) is an incremental life-cycle assurance workbench for high-assurance software-reliant systems. It utilizes architecture abstractions in models to manage requirements across multiple layer of a system architecture and the verification of a system implementation against these requirements. The workbench complements the capabilities of an architecture-centric virtual system integration workbench for the development of such systems.

Alisa utilizes SAE AADL as an architecture modeling language. Alisa provides several notations for users to specify requirements, verification plans, and execute verification plans incrementally throughout the life cycle.


Requirements Engineering
Requirements Validation
Architecture Modeling Analysis & Generation
Deployment Build
Acceptance Test
System Design
System Architecture Validation
Target Build
System Test
70% Defect Introduction
Reduced Cost and Time through Early Discovery
80% Post Unit Test Discovery
Software Architectural Design
Software Architecture Validation
Integration Build
Integration Test
Improved Quality through Better Requirements & Evidence
Software Design
Validation
Build the System
Code Development
Unit Test
Assure the System

Incrementally Evolving and Maintaining the Assurance Evidence

Incremental Lifecycle Assurance Concepts and Notations

We have introduced several notations in support of incremental lifecycle assurance.

- *ReqSpec:* a notation for specifying *stakeholder goals* and *system requirements*.
- *Verify:* a notation for registering *verification methods*, specifying *verification activities*, and *verification plans* with *claims* that requirements are satisfied by the results of verification activities.
- *Alisa:* a notation for defining *assurance cases* that consist of one or more *assurance plan*. An assurance plan configures the assurance of a particular AADL instance model and related artifacts and specifies how many subsystems are included in the given assurance case. *Assurance tasks* define a filtered view of the assurance case based on category labels that allows users to focus on a high priority subset of requirements and verification activities.
- *Assure:* a notation that represents an *assurance case result instance*, *i.e.*, the assurance evidence as the result of executing verification plans on one or more system instance models. Assure instances in this notation are automatically generated for an assurance case and assurance task.

These notations are supported by additional notations:

- *Organization:* a notation for defining the stakeholders of a system.
- *Categories:* a notation for defining requirement categories, verification categories, and selection categories

Users can define the following types of categories:

- *quality category labels:* These labels represent quality attributes and are used to categorize requirements, goals, and verification methods.
- *phase category labels:* These labels represent development phases and are used to categorize verification methods.
- *selection category labels:* These labels are used to introduce additional user-defined selection criteria and are used on requirements, goals, verification methods, and verification activities.

Grammars and Notational Conventions

The notations use a simple syntax and are case sensitive. Each construct introduces an instance of a particular Alisa concept, which we will refer to as *entity*, and identifies it with a unique name. The name is a simple identifier or a <dot> separated identifier sequence. Entities in a container, such as a requirement in a system requirements container, are qualified with the name of a container entity.

As the identifier may take the form of a label such as Req1, each entity optionally also has a title field that provides a short descriptive label. This label may be used instead of or an addition to the identifier for presentation in views.

An entity may identify a model element that it is associated with.

An entity may contain a number of attributes, i.e., labelled values or references to other model elements. These are enclosed in square brackets.

When describing the syntax of Alisa notations we will use BNF-like syntax markers to indicate optional constructs

- *(construct)?* for an optional construct,

- *(construct)** for a construct repeated zero or more times,

- *(construct)+* for a construct repeated one or more times, and

- *a* | *b* to represent altenatives.

Grammar rules are labeled by *<conceptname> ::=* and rules may be referred by *<conceptname>*. The reference means that the construct is to be inserted in place as nested entity within another construct.

We use the following approach for cross references:

- References to AADL classifiers use the AADL2 syntax to qualify the classifier name with the package name, e.g., my::pack::sys.impl for a system implementation reference.
- Within requirements the user may reference a feature within a classifier. this is done by the feature name, which is resolved relative to the classifier reference of the enclosing Requirement

specifications container.

- References to objects in the Alisa languages, e.g., requirements and goals, use a "." separator in the qualified name path, e.g., systemreqs.r1. This includes references to verification activities, verification methods, verification plans, and to stakeholders. In many cases the references do not have to be qualified as the context uniquely identifies to target. For example, the requirement reference of a claim is within the scope of the system requirements container identified by a verification plan.
- User defined categories are globally known by their name. Categories of different kinds (requirement, verification, selection) can use the same name as they only have to be unique within their kind.

Note that we will be providing a user interface that presents the Alisa information in a graphical or structured navigator view with the details of a selected entity shown in a properties view. The user will be able to create entities within the navigator view and edit its attributes in the properties view. In other words, in the future users of Alisa may not have to learn the details of the textual syntax of the Alisa notations.