Model-Based Engineering with AADL: An Overview

Peter Feiler
phf@sei.cmu.edu
Outline

AADL: The Language
What’s New in AADL V2
Modeling with AADL
**AADL: Components and Connections**

- **Component type**
  - identifier
  - component category
  - extends {component_type}
  - features
    - port
    - port group
    - parameter
    - access
    - subprogram

- **Features**
  - flow specification
  - properties

- **Component implementation**
  - identifier
  - extends {component implementation}
  - refines type
  - subcomponents
  - connections
  - call sequences
  - modes
    - flow implementation & end-to-end flows
    - properties

- **Connections**
  - data
  - event
  - event data
  - port group
  - access

- **Component Category**
  - application
  - platform
  - processor
  - memory
  - device
  - processor/virt. bus
  - system
  - abstract

- **Package**
  - public
  - component classifier
  - private
  - component classifier

- **Properties**
  - standard
  - user defined

- **Property set**
  - property types
  - property definitions
  - property values

- **More details**
Application Components

System: hierarchical organization of components

Process: protected address space

Thread group: organization of threads in processes

Thread: a schedulable unit of concurrent execution

Data: potentially sharable data & datqa typing

Subprogram: callable unit of sequential code
Execution Platform Components

Processor – provides thread scheduling and execution services

Virtual processor – hierarchical schedulers & partitions

Memory – provides storage for data and source code

Bus – provides physical connectivity between hardware components

Virtual bus – virtual channels & protocols

Device – interface to external environment, physical components
system GPS
features
    speed_data: in data port metric_speed
       {SEI::BaseType => UInt16;};
    geo_db: requires data access real_time_geoDB;
    s_control_data: out data port state_control;
flows
    speed_control: flow path
       speed_data -> s_control_data;
properties SEI::redundancy => Dual;
end GPS;
System Implementation

```plaintext
system implementation GPS.secure
subcomponents
  decoder: system PGP_decoder.basic;
  encoder: system PGP_encoder.basic;
  receiver: system GPS_receiver.basic;
connections
  c1: data port speed_data -> decoder.in;
  c2: data port decoder.out -> receiver.in;
  c3: data port receiver.out -> encoder.in;
  c4: data port encoder.out -> s_control_data;
flows
  speed_control: flow path speed_data -> c1 -> decoder.fs1
                -> c2 -> receiver.fs1 -> c3 -> decoder.fs1
                -> c4 -> s_control_data;
modes none;
properties arch::redundancy_scheme => Primary_Backup;
end GPS;
```
Some Standard Properties

Dispatch_Protocol => Periodic;
Period => 100 ms;
Compute_Deadline => value (Period);
Compute_Execution_Time => 10 ms .. 20 ms;
Compute_Entrypoint => “speed_control”;
Source_Text => “waypoint.java”;
Source_Code_Size => 12 KB;

Thread

Thread_Swap_Execution_Time => 5 us.. 10 us;
Clock_Jitter => 5 ps;

Processor

Allowed_Message_Size => 1 KB;
Propagation_Delay => 1ps .. 2ps;
Bus_Properties::Protocols => CSMA;

Bus

Protocols is a user defined property

Thread

Code to be executed on dispatch

File containing the application code

AADL Introduction
Peter Feller, June 2009
© 2009 Carnegie Mellon University
Latency Impact of Partitions

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Request for new page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Display Manager</td>
</tr>
<tr>
<td></td>
<td>Page Content Manager</td>
</tr>
<tr>
<td>Flight Manager</td>
<td>Latency contribution:</td>
</tr>
<tr>
<td>Flight Director</td>
<td>Partition period per partition hop</td>
</tr>
</tbody>
</table>

AADL Introduction
Peter Feller, June 2009
© 2009 Carnegie Mellon University
device brake_pedal
features
  brake_status: out data port bool_type;
flows
  Flow1: flow source brake_status;
end brake_pedal;

system cruise_control
features
  brake_status: in data port;
  throttle_setting: out data port;
flows
  brake_flow_1: flow path brake_status -> throttle_setting;
end cruise_control;

device throttle_actuator
Features
  throttle_setting: in data port float_type;
flows
  Flow1: flow sink throttle_setting;
end throttle_actuator;
Architecture Execution Semantics Defined – Components to SoS

Nominal & recovery
Fault handling
Resource locking
Mode switching
Initialization
Finalization

Thread Example Diagram
Component Interactions & Modes

Completely defined interfaces & interactions

- Port-based flows
  - State data, events, messages
  - Flow specifications & connections
  - End-to-end flows
- Synchronous call/return
- Shared access

Modal & dynamically configurable systems

- Modeling of operational modes
- Modeling of fault tolerant configurations
- Modeling of different levels of service
A Partitioned Run-Time Architecture

Strong Partitioning
- Timing Protection
- OS Call Restrictions
- Memory Protection

Interoperability/Portability
- Tailored Runtime Executive
- Standard RTOS API
- Application Components
Domain Data Modeling

Domain types & base types
Data value range and units of measurement
Data transfer assumptions
  - Guaranteed delivery
  - Ordered delivery
Data stream characteristics
  - Computational error
  - Miss rates
  - Freshness
AADL and Safety-Criticality

Fault management

- Architecture patterns in AADL
  - Redundancy, health monitoring, …
- Fault tolerant configurations & modes

Dependability

- Error Model Annex to AADL
- Specification of fault occurrence and fault propagation information
- Use for hazard and fault effect modeling
- Reliability & fault tree analysis

Behavior validation

- Behavior Annex to AADL
- Model checking
- Source code validation

Consistency checking of safety-criticality levels
Architecture Redundancy Pattern

External and internal mode control
Pattern variants with different latency & fault propagation characteristics
Leverage Connectivity in AADL Models

Fault propagation at the application logic level, at the hardware level, and between the two levels.

- Provides compositional model specification approach
- Architecture defines propagation paths for software and hardware

Generation of stochastic process models & fault trees
Large-Scale Development

Component evolution

- Component templates & refinement
- System families
- Component variants
- Components as extensions of other components
- Model configuration by property values

Large models & team development

- Components organized into AADL packages
- Public & private package sections
- Independently developed packages
- Version management of AADL packages
- Model integration
AADL Language Extensions

Model annotation through properties and sublanguages
New properties defined through property sets
Standard compliant sublanguage syntax in annex subclauses
Project-specific language extensions
Language extensions as approved SAE AADL standard annexes

Examples

• Error Model
• Interaction behavior
• System partitions (e.g., ARINC 653)
COTRE Annex Extension

THREAD t
FEATURES
  sem1 : DATA ACCESS semaphore;
  sem2 : DATA ACCESS semaphore;
END t;

THREAD IMPLEMENTATION t.t1
PROPERTIES
  Period => 13.96ms;
  cotre::Priority => 1;
  cotre::Phase => 0.0ms;
  Dispatch_Protocol => Periodic;

ANNEX cotre.behavior {**
STATES
  s0, s1, s2, s3, s4, s5, s6, s7, s8 : STATE;
  s0 : INITIAL STATE;
TRANSITIONS
  s0 -> s1 { PERIODIC_WAIT };
  s1 -> s2 { COMPUTATION(1.9ms, 1.9ms) };
  s2 -> s3 { sem1.wait ! (-1.0ms) };
  s3 -> s4 { COMPUTATION(0.1ms, 0.1ms) };
  s4 -> s5 { sem2.wait ! (-1.0ms) };
  s5 -> s6 { COMPUTATION(2.5ms, 2.5ms) };
  s6 -> s7 { sem2.release ! };
  s7 -> s8 { COMPUTATION(1.5ms, 1.5ms) };
  s8 -> s0 { sem1.release ! };
**};
END t.t1;

COTRE thread properties

COTRE behavioral annex

 Courtesy of cotre

Software Engineering Institute  Carnegie Mellon
Outline

AADL: The Language
What’s New in AADL V2
Modeling with AADL
What’s New in AADL2

Packages and classifier visibility

New classifiers
Classifier matching and substitution
Explicit parameterization of classifiers (prototypes)
Abstract features
Connection improvements
Feature groups
Arrays of components
Property improvements
Threads and communication timing
Subprograms and subprogram groups
Packages and Classifier Visibility

Component implementation in public and private section of packages

- Public: represents component variant: only properties
- Private: represents realization

With statement

- Specifies the set of packages whose classifiers can be referenced

Renames statement

- Represents a local alias for long qualified classifier references to component types
- Renames provides short names for fully qualified package names
Abstract Components

Example: A conceptual architecture

```
abstract car
    end car;

abstract implementation car.generic
    subcomponents
        power_train: abstract PowerTrain;
        exhaust_sys: abstract ExhaustSys;
    end car.generic;

abstract PowerTrain
    end PowerTrain;
```

```
system carRT extends car
    end car;

system implementation carRT.impl extends car.generic
    subcomponents
        power_train: refined to process PowerTrain;
        exhaust_sys: refined to process ExhaustSys;
    end carRT.impl;
```

Concrete category extends abstract type/implementation

Concrete category supplied by refinement
New Classifiers

Subprogram group classifier

- Represents subprogram library
- Provides subprogram features
- Provides and requires access to subprogram group

Virtual processor classifier

- Represents hierarchical scheduler, virtual machine, processor partition, rate group task
- Virtual processor as subcomponent
- Virtual processor is bound to virtual processors and processors
New Classifiers - 2

Virtual bus classifier

- Represents logical/virtual channel, protocols and protocol stacks
- Required_Virtual_Bus_Class and Provided_Virtual_Bus_Class property associated with buses, processors and virtual buses
- Required/Provided_Connection_Quality_Of_Service property
  - Example value: guaranteed_delivery
- Virtual bus subcomponents
- No requires virtual bus access and provide virtual bus access
Refinement Substitution Rules

**Classifier_Match**
1. Goto type
2. Select an implementation

**Type_Extension**
1. Goto type
2. Select an extension
3. Select an implementation (optional)

**Signature_Match**
1. Goto type
2. Select a type with a superset of features and flow specifications
3. Select an implementation (optional)
Prototypes – Consistent Refinement

Example: Type of data on a port

```plaintext
system GpsGeneric
  prototypes
  dt: data;
  features
  pos_1: out data dt;
  pos_2: out data dt;
end GpsGeneric;

system Gps
  extends GpsGeneric(dt=>PosData)
end Gps;
```

Compare to refinement

```plaintext
system GpsBasic
  features
  pos_1: out data;
  pos_2: out data;
end GpsBasic;

system GpsRef extends GpsBasic
  features
  pos_1: refined to out data PosData;
  pos_2: refined to out data OtherData;
end GpsRef;
```

No enforcement of consistency possible
Relaxation of Port Connection Rules

Connection syntax

- Only **port** keyword

Connections between data port, event data ports, event ports

- event port -> event port
- data port -> data port, event data port, event port
- event data port -> event data port, data port, event port

Connections between ports and data components

- data, data access -> data port, event data port, event port
- data port, event data port -> data, data access
Relaxation of Port Connection Rules

Connection syntax

- Only **port** keyword

Connections between data port, event data ports, event ports

- event port -> event port
- data port -> data port, event data port, event port
- event data port -> event data port, data port, event port

Connections between ports and data components

- data, data access -> data port, event data port, event port
- data port, event data port -> data, data access
Feature Group Improvements

For any feature

- Not just ports

**Inverse of** in feature declaration

- No need to define both a feature group type and an inverse feature group type

Matching of independently declared feature group type

- Ability to define *equivalence*

Prototypes and refinement

- Classifier and feature prototypes
- Substitution rules
Arrays in AADL

Arrays of subcomponents
Multiple array dimensions
  • Of same subcomponent
  • Of subcomponent and ancestor component

Single array dimension for features
  • Example applications: routers, voters
Arrays in AADL – Example

Fan-in, fan-out at process port

Thread connectivity across processes
T1, t2, t3, t4 : thread controller [4];
Connection_Pattern => ( one_to_one);
t1 -> t3 arrays ; t1 -> t4 arrays; t2 -> t3 arrays; t2 -> t4 arrays with pairwise connected elements

Feature group in process
• t1 -> t3 array; t2 -> t4 array
Connection patterns – one dimension

Identity aka. One_To_One

next

Cyclic_Next

previous

Neighbor = (next, previous)

Cyclic Neighbor = (CyclicNext, CyclicPrevious)

next, one_to_one
Connection patterns – two dimensions

S[3,3]; D[3,3]; Port S.p1 -> D.p2;

(Identity, identity)

(next, next)

(Identity, next)
Connection patterns – two dimensions

self

M[3,3]; port M.p1 -> M.p2;

(identity, next)  (next, identity)  (next, next)

(next, next), (next, previous)  (identity, previous), (identity, next)  (identity, identity)
Mode Improvements

Modes in component types

- Modes common to all implementations
  - No additional modes in implementation
- Mode-specific property values on component type, flow spec, and features

Named mode transitions

Improved specification of mode transition activities

- Based on Rolland TLA specification

Inherited modes

- **Requires modes** to indicate which modes on an enclosing component the component is sensitive to
- Mapping of mode names
Flow Improvements

End-to-end flows composed of end-to-end flows

- Ete12: **end to end flow** ete1 -> conn12 -> ete2
- Ete13: **end to end flow** ete1 -> conn13 -> ete3

Flow through data components

End-to-end: T1.fs1 -> c1 (write) -> d -> c2 (read) -> T2.fs2

Did you know
Multiple end-to-end flows from feature group connections and connections involving arrays
System Modeling

Layered architectures

- Use of system implementation declarations
- Implemented_As property

Asynchronous systems

- Synchronization domains
- Reference_Time property
What’s New in AADL2

Packages and classifier visibility
New classifiers
Classifier matching and substitution
Explicit parameterization of classifiers (prototypes)
Abstract features
Connection improvements
Feature groups
Arrays of components
Property improvements
Threads and communication timing
Subprograms and subprogram groups
Outline

AADL: The Language
What’s New in AADL V2
Modeling with AADL

- What is your system?
- Modeling multiple tiers
- Finding the right abstraction
- Filling in the software architecture
- Team support & Variants
- Modeling a reference architecture
Towards Architecture Centric Engineering

Build on architecture tradeoff analysis (e.g., SEI ATAM)
  • Provides focused evaluation method
  • MBE/AADL provides quantitative analysis & starter models to build on

Facilitate pattern-based technical architecture root cause analysis
  • Identify systemic risks in technology migration and refresh
  • AADL provides semantic framework to reason about technical problem areas and potential mitigation strategies

Scalability through architecture extraction
  • Leverage existing design data bases
  • Challenge: abstract away from design details
  • Focus on “what” instead of “how”

Support system and software assurance
  • Provides structured approach to safety/dependability assurance
  • MBE/AADL provides evidence based on validated models
Software & System Engineering

System Engineering

Physical System Model

Operational Environment

Physical System

Computing Platform

Embedded Software

Application Domain

Embedded Software System

SAE AADL

AADL – SysML coordination

Embedded Software System Engineering

SysML

© 2009 Carnegie Mellon University
Modeling an Embedded System Architecture

Elements of an embedded system architecture

- Software Architecture PLUS
- Hardware Architecture PLUS
- Physical system/environment PLUS
- Logical interface between software and physical system PLUS
- Physical interface between hardware and physical system PLUS
- Deployment of software on hardware
Outline

What is your system?

**Modeling multiple Tiers**

Finding the right abstraction

Filling in the software architecture

Team support & Variants

Modeling a reference architecture
Multiple Tiers of a System

Aircraft system: (Tier 1)
Engine, Landing, Cockpit, … weight, electrical, fuel, hydraulics, …

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

Subcontracted software subsystem: (Tier 3)
Tasks, periods, execution time
Software allocation, schedulability
Abstraction in System Architecture

Bus to represent resource such as hydraulics

Bus to represent resource connections such as hydraulic lines
Embedded Software Architecture

Computing Platform

Application Software
Outline

What is your system?
Modeling multiple Tiers
Finding the right abstraction
Filling in the software architecture
Team support & Variants
Modeling a reference architecture
Hardware as an Abstraction

Memory
- Individual memory modules
- Processor with memory

High-speed Ethernet switch
- As bus with latency properties
- As collection of hardware components with processing queues

Multi-core processors
- As single AADL processor
- Individual cores as processors

Digital camera
- As AADL device
- \textit{Implemented as} system with hardware and software

Challenge: abstract away from design details
Focus on “what” instead of “how”
Intended Data Flow in Task Architecture

From other Partitions

Pr 1

20Hz

Periodic I/O

Navigation

Sensor Processing

Pr 2

20Hz

Integrated Navigation

Pr 3

10Hz

Shared data area

Priority inversion & non-determinism through preemption

To other Partitions

Pr 4

20Hz

Guidance Processing

Pr 6

5Hz

Flight Plan Processing

Pr 9

2Hz

Aircraft Performance Calculation

Decreasing Priority

Intended flow documented in design document table

Priority assignment achieves desired data flow
Modeling the What

- Navigation Sensor Processing
  - 20Hz
  - Nav signal data
  - Nav sensor data

- Integrated Navigation
  - 10Hz
  - Nav data

- Guidance Processing
  - 20Hz
  - Guidance

- Flight Plan Processing
  - 5Hz
  - FP data

- Aircraft Performance Calculation
  - 2Hz
  - Performance data

- From Partitions

- To Partitions

- Input-compute-output (ICO) AADL thread semantics
- Immediate and delayed data port connections for deterministic sampling

AADL Introduction
Peter Feller, June 2009
© 2009 Carnegie Mellon University
Types Vs. Properties

Example: Power supply

Type enforcement by AADL semantics

Multiple properties mapped into types

```aadl
bus LAN
  features
    Power: requires bus access PowerSupply.V9_0 { 
      SEI::PowerBudget => access 0.1 W;
    };
  properties
    SEI::NetWeight => 2.5 kg;
end LAN;

bus PowerSupply
  properties
    SEI::PowerCapacity => 10.0 W;
    SEI::NetWeight => 25.0 kg;
end PowerSupply;

bus implementation PowerSupply.V9_0
end PowerSupply.V9_0;

bus implementation PowerSupply.V4_5
end PowerSupply.V4_5;
```
Flow Use Scenario through Subsystem Architecture

Display -> IOProcessor ->
Command -> Comm -> Nav ->
IOProcessor -> Modem ->
IOProcessor -> Nav -> Comm ->
Command -> Display

Latency = Partition hops +
processing + transfer
Independent clock per
processor

Multiple rates and
processors with
independent clocks
Outline

What is your system?
Modeling multiple Tiers
Finding the right abstraction
Filling in the **software architecture**
Team support & Variants
Modeling a reference architecture
Systems Context Diagrams

Defines all of the entities that are within the scope of an application

Shows the data flows that are included in the scope of a application

Focuses on relationships with external entities and identifies the information that is exchanged between these external entities and the system under review.

Context Diagram at the project level is the root of the Functional Decomposition at the conceptual level of detail. (Level 0)

Subsequent decompositions (1 to n) identify components and the logical flow of data between the components
A Typical Context Diagram
Cruise Control
Functional Decomposition-Level 0

- Engine
- Speed Sensor
- Brake sensor
- User Input Device
- Display
- Throttle Actuator
- Other Control elements
Feature Groups

Feature Groups are collections of individual ports and port groups such that

- a feature group can be connected to individually
- a component feature group can be connected as a single unit
Compute Desired Speed
Functional Decomposition-Level 2

Previous sample instantaneous velocity

OK to run
Instantaneous velocity

Functional specification in domain notation
Outline

What is your system?
Modeling multiple Tiers
Finding the right abstraction
Filling in the software architecture

Team support & Variants
Modeling a reference architecture
Use of AADL Packages

AADL Packages as name spaces
  • Qualify references by package name: BaseTypes::uint16
  • Nested package names: edu::cmu::sei::MySystem::App1

Component libraries
  • Component types and implementations
  • Hardware & application libraries
  • Subsystem details in separate packages

Data dictionary
  • Data types
  • Domain information on data types

Interaction specifications
  • Port group type specifications
Model Repository

Packages as versioned units
Eclipse projects as Configuration Items

Development Repositories

Subcontractor1

Public

Subsys Spec rev
Subsys Spec
Subsys ICD rev
Subsys Impl
Subcontractor data dictionary
Test harness

Internal

Subsys Impl
Subsys ICD
Subsys Spec
Subsys Spec rev

Public

Subsys data dictionary

Subcontractor2

Public

System spec
System configurations
Integration test harness
Integrator data dictionary

Public

System Impl

Integrator

Public

Subsys Impl
class dictionary
Subsys ICD
Possibly Public Per Sub

Public

Subsys Spec
Subsys Spec rev
Subsys Impl
Subsys ICD rev
Subsys data dictionary
Test harness

Internal

Subsys Impl
Subsys data dictionary

Public

System spec
System configurations
Integration test harness
Use of extends to specify configurations

Use of contained property association to keep deployment information in one place
Variants in System Families or Product Lines

Multiple interface variants
- AADL component types with extends

Multiple realizations
- Multiple AADL component implementations per type

Variation in component structure and communication
- Parameterized component implementations (AADL V2 Prototype)
- Dynamic variation through mode-specific subcomponents and connections

Source code variations
- Different source files as Source_Text property
- Conditional compilation flags as properties or property constants

Seed & calibration values
- As property values on data components

AADL V2 supports multiple data sets
Outline

What is your system?
Modeling multiple Tiers
Finding the right abstraction
Filling in the software architecture
Team support & Variants

Modeling a reference architecture
The Mission Data System

A reference architecture

- To be instantiated for different applications

An embedded systems architecture

- Consists of physical system, computing hardware, application software

A control systems architecture

- Feedback loops in application architecture
- Feedback loops in data management system

A multi-layered architecture

- From low-level control loops to goal-oriented planning and plan execution
A Reference Architecture

Generic Architecture Pattern With Connection Topology

Instantiation of Application Architecture Computing Platform, and Physical System
Mission and Control Processing

Goal-oriented Mission Tasks

Time-sensitive Continuous Control Tasks

System Under Control

State variables
- Intent
- Knowledge

Elaboration, projection, & scheduling

Execution

state-based model
System Under Control

measurements
commands
Excerpt from the Textual Specification:

process implementation MDSControlSystem.basic
subcomponents
  GoalPlanner: thread group ControlSoftware::GoalPlanner;
  GoalExecutive: thread group ControlSoftware::GoalExecutive;
  GoalMonitor: thread group ControlSoftware::XGoalMonitor;
  StateEstimation: thread group ControlSoftware::estimator;
  StateControl: thread group ControlSoftware::controller;
  OperatorConsole: thread group
    ControlSoftware::OperatorConsole;

Textual & Graphical Representations

Focus on Information Flow
Mission Data System (MDS) Architecture*

Closed loop
Goal-Directed
Explicit models
Separation of Concerns
Integral Fault Protection

* M. Bennett, R. Knight, R. Rasmussen, M. Ingham, "State-Based Models for Planning and Execution, 2006-08-11.
Separation of Concerns: Data Management System

Role of state history and state variables

- Information flow through application architecture
  - Access to multiple data stream values
- Historical log of data stream for post mortem analysis
  - Storage management through compression
- Distributed processing between space and ground system
  - Proxies & telemetry

Managed data history variables
Flow-Oriented Model of Temperature Control

Flow path
Reference Architecture Instantiation

package ControlSoftware::Camera

public

thread group estimator
  extends ControlSoftware::estimator
  features
    StateEstimates: refined to port group MDSData::Camera::StateVariables;
    measurements: refined to port group MDSData::Camera::RequestedCommands;
    PreviousCommands: refined to port group MDSData::Camera::RequestedCommands;
  flows
    StateFlow: flow path measurements -> StateEstimates;
    cmd estimator;

thread group implementation estimator.camera
  subcomponents
    TemperatureEstimator: thread TemperatureEstimator;
    TemperatureSensorHealthEstimator: thread TemperatureSensorHealthEstimator;
    HeaterSwitchEstimator: thread HeaterSwitchEstimator;
  connections

Instantiation of reference architecture through refinement of AADL model

Deployment on different computing hardware platforms
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.

Use of any trademarks in this presentation is not intended in any way to infringe on the rights of the trademark holder.

This Presentation 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.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.