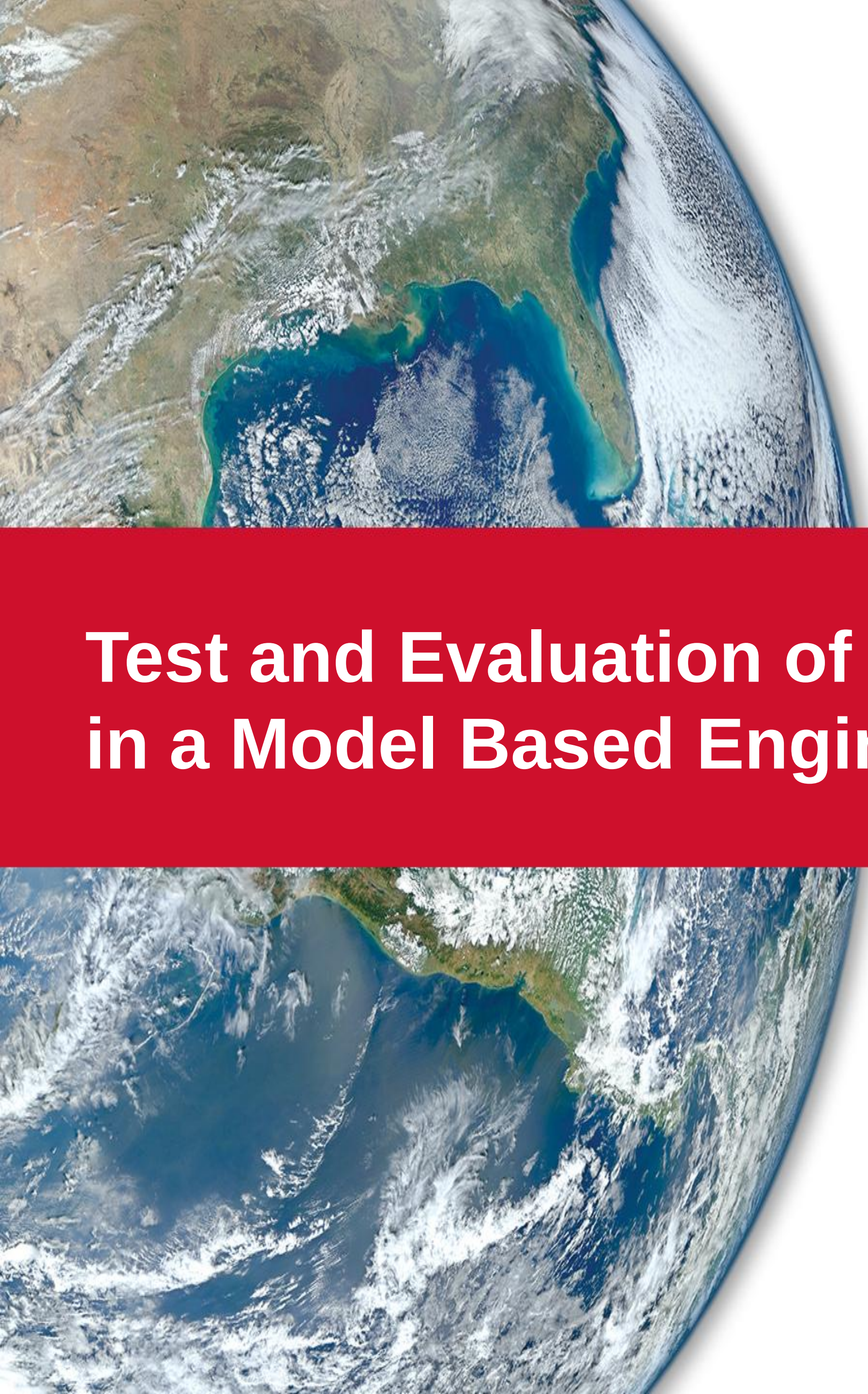


Raytheon

# Test and Evaluation of Autonomous Systems in a Model Based Engineering Context


**Raytheon**

Michael Nolan

**USAF AFRL**

Aaron Fifarek
Jonathan Hoffman

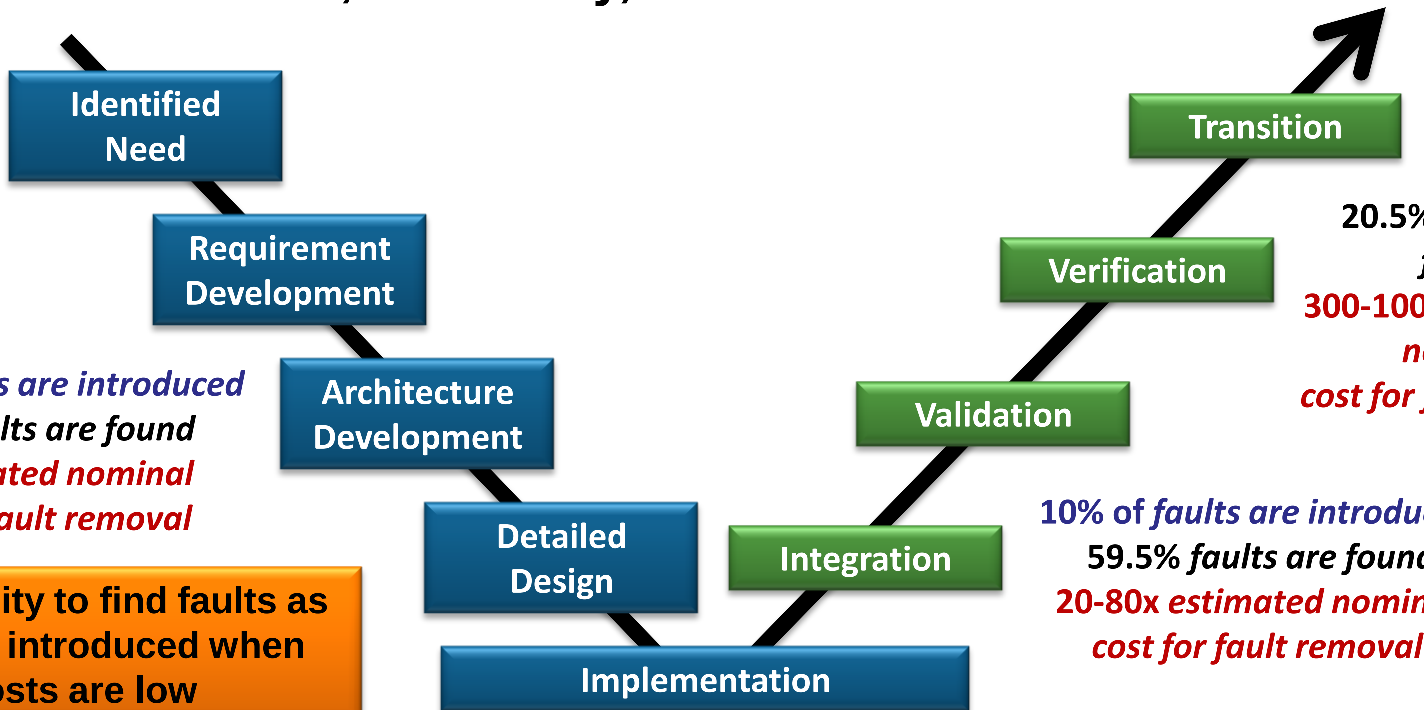

3 March 2016

# Agenda

- Motivation
- Trust and Certification Process
- Background
- Formal Analysis
- Requirements Analysis
- Architecture
- Model Traceability
- SysML Representation of Autonomous System and Autonomous System Development
- Basic example of Autonomous Systems T&E in MBE context
- Summary

# Motivation

## Introduction, Discovery, and Cost of Software Faults[1,2,3]




70% of *faults are introduced*
3.5% *faults are found*
1x *estimated nominal cost for fault removal*

20% of *faults are introduced*
16% *faults are found*
5x *estimated nominal cost for fault removal*

10% of *faults are introduced*
59.5% *faults are found*
20-80x *estimated nominal cost for fault removal*

20.5% *faults are found*
300-1000x *estimated nominal cost for fault removal*

**Opportunity to find faults as they are introduced when costs are low**

**Rework and certification is 70% of SW cost.[4]**


1. NIST Planning report 02-3, *The Economic Impacts of Inadequate Infrastructure for Software Testing,* May 2002.
2. D. Galin, *Software Quality Assurance: From Theory to Implementation*, Pearson/Addison-Wesley (2004)
3. B.W. Boehm, *Software Engineering Economics*, Prentice Hall (1981)

# Trust and Certification

## Products / Process

∑Individual Certifications
System of Systems Certification

Compositionally Verified Systems of Systems

## Design

- Requirements
- Architecture
- Models

## Validation

- Simulation
- Testing

New Auto y Need

Multiple
V&V
Technology
Paths

Certified
Assurance
Case

VERIFIED & VALIDATED
CERTIFIED AUTONOMY
LICENSE

System Design and Safety Requirements
(ARP 4761, ARP 4754/A, MIL-HDBK-882E)

Testable Requirements & Verification Plans
(DO-178C/254, MIL-HDBK-516)



AFRL

# Formal Analysis

***Formal Methods*** *refers to mathematically rigorous techniques and tools for the specification, design and verification of software and hardware systems.*

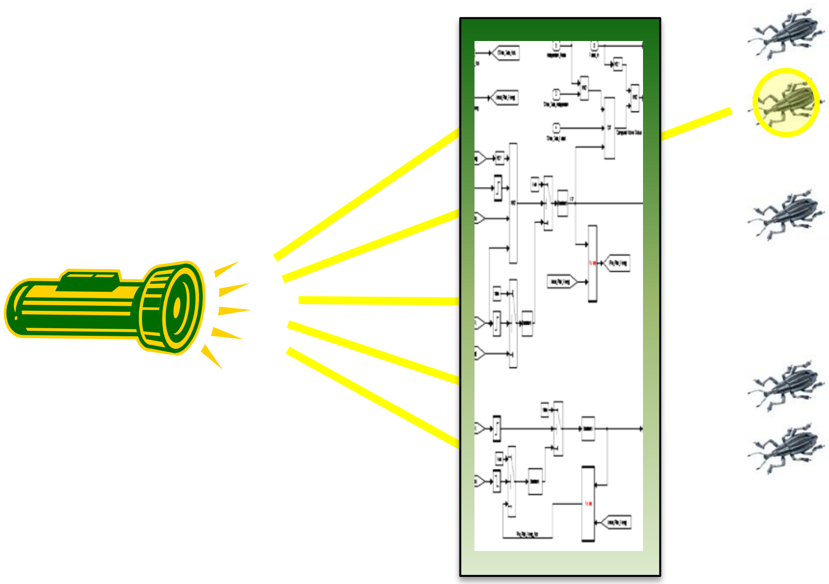
*- Langley Formal Methods (http://shemesh.larc.nasa.gov/fm/fm-what.html)*

- What is Formal Analysis?
  - Analysis performed on mathematically precise models utilizing elegant Computer Science algorithms and tools
    - Model-Checking
    - Theorem Proving

- Why do we want to do it?
  - We can exhaustively search the behavior of models to prove or disprove desired properties
  - Removal of ambiguity due to required mathematical rigor
  - Can identify unintended and unspecified behaviors

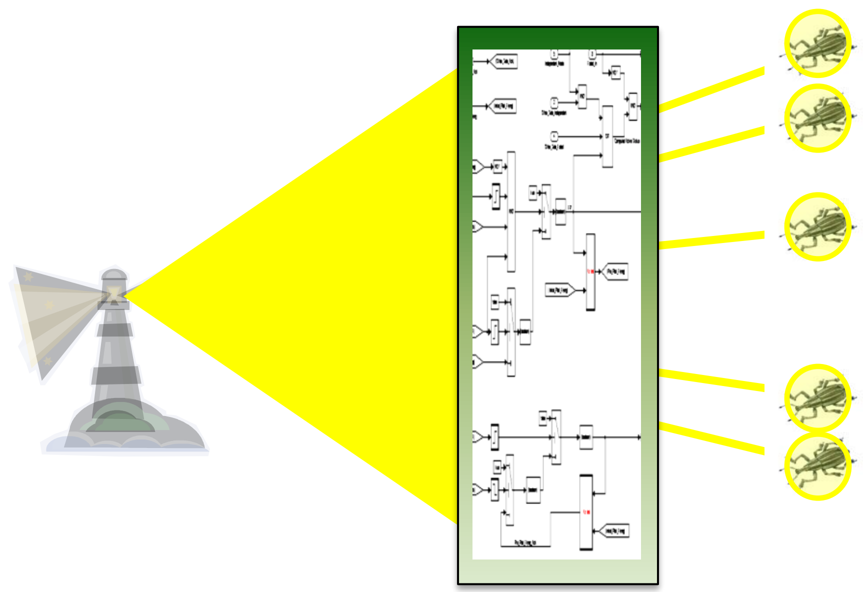
# Analysis
## Advantage of Model Checking

*Testing Checks Only the Values We Select*



*Even Small Systems Have Trillions (of Trillions) of Possible Tests!*

*Model Checker Tries Every Possible Value!*



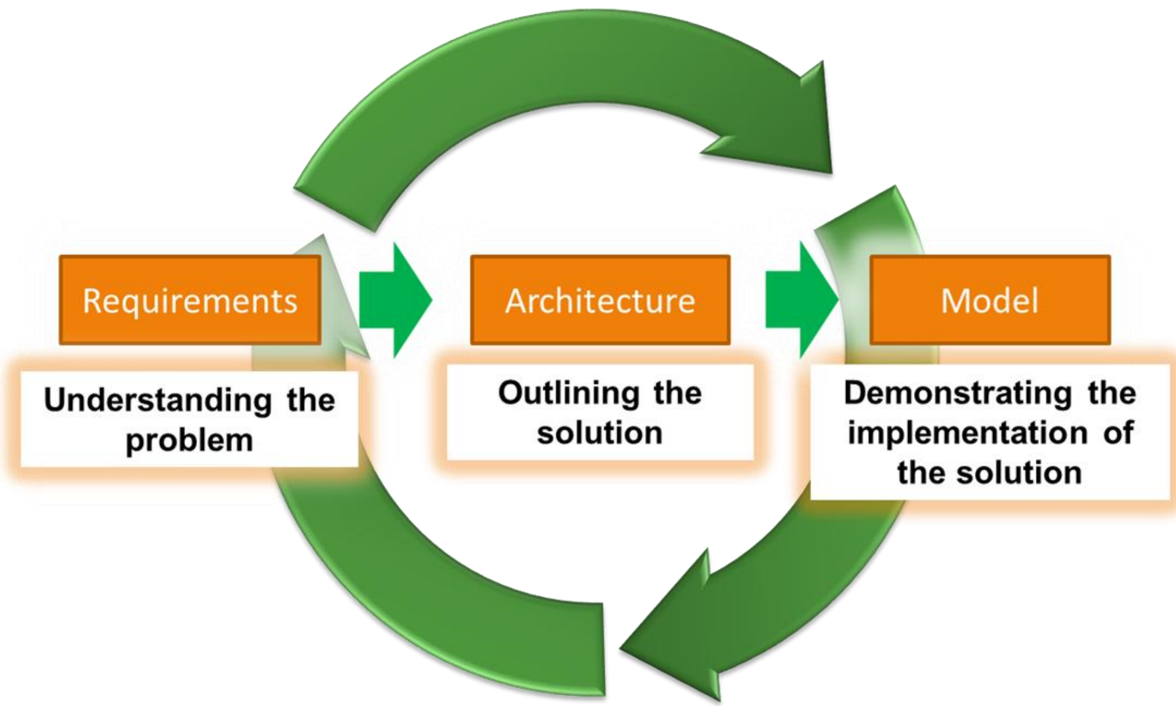
*Finds every exception to the property being checked!*

# Requirements Development & Analysis

**REQUIREMENTS DEV. AND ANALYSIS**

Precise, structured standards to automate requirement evaluation for testability, tractability, and deconfliction

***Precise, structured standards to automate requirement evaluation for testability, traceability, and de-confliction***



# Formal Requirements Analysis

- Natural language requirements are difficult to process logically and mathematically especially if they are not written with a formal basis
  - *"The flight control function that performs the automatic avoidance maneuver shall be of a level of redundancy equivalent to the primary flight control system"*
    - What is the formal definition of this constraint on the system?
    - Not a trivial definition on the system

> ***Formal Methods*** *refers to mathematically rigorous techniques and tools for the specification, design and verification of software and hardware systems.*
>
> *- Langley Formal Methods (http://shemesh.larc.nasa.gov/fm/fm-what.html)*

Temporal logic definitions are not obvious to write for most individuals and takes years of practice to master effectively

| |
|---|
| $\square(p \rightarrow a)$ |
| $\square(p \rightarrow \Diamond a)$ |
| $\square(p \rightarrow (\neg b\ U\ ((a \vee \neg p) \vee \square \neg b)))$ |
| $\square(p \rightarrow ((b \rightarrow (p\ U\ (a \wedge p)))\ U\ (\neg p \vee \square((b \rightarrow (p\ U\ (a \wedge p)))))))$ |

What does that mean?

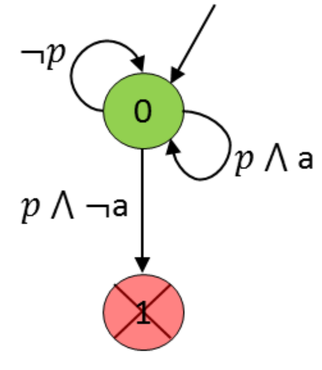
There may be logical basis but it's not accessible to others.

# Formal Requirements Analysis

- Our Approach – ***Pattern Implementation***
  - Constrain natural language to patterns which contain a scope and a predicate
  - Enforces the formal basis necessary to ensure mathematical rigor

- Can requirements be defined and verified compositionally?

| | |
|---|---|
| *while p :: always a* | $\square(p \rightarrow a)$ |
| *while p :: exists a* | $\square(p \rightarrow \Diamond a)$ |
| *while p :: a proceeds b* | $\square(p \rightarrow (\neg b\ U\ ((a \vee \neg p) \vee \square \neg b)))$ |
| *while p :: a responds to b* | $\square(p \rightarrow ((b \rightarrow (p\ U\ (a \wedge p)))\ U\ (\neg p \vee \square((b \rightarrow (p\ U\ (a \wedge p)))))))$ |




$\square(p \rightarrow a)$

while p :: always a

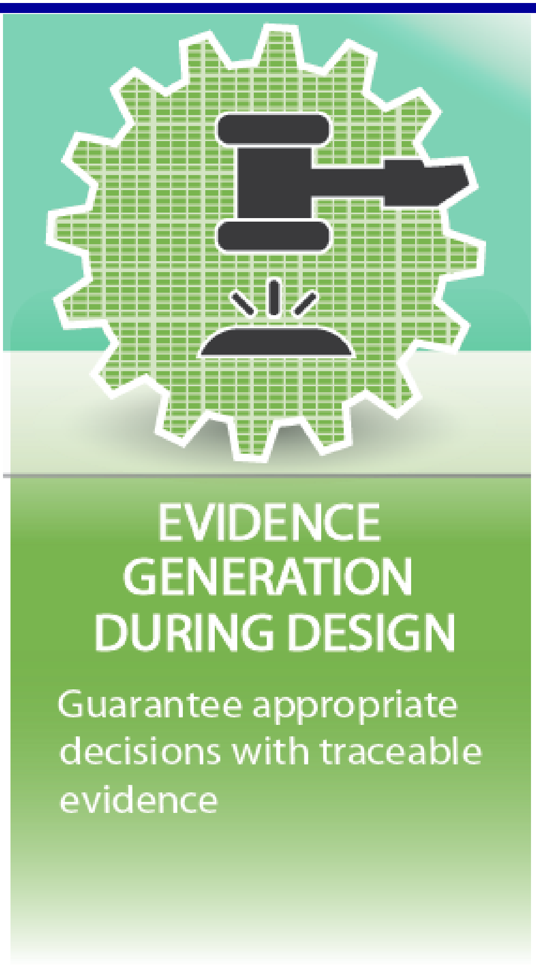
$\underbrace{\text{While the pump is ON}}_{\text{Scope}}\ \underbrace{\text{the pump outflow shall be the maximum flow rate.}}_{\text{Predicate}}$

**while (pump_state == ON) :: always (pump_flow == MAX_FLOW)**

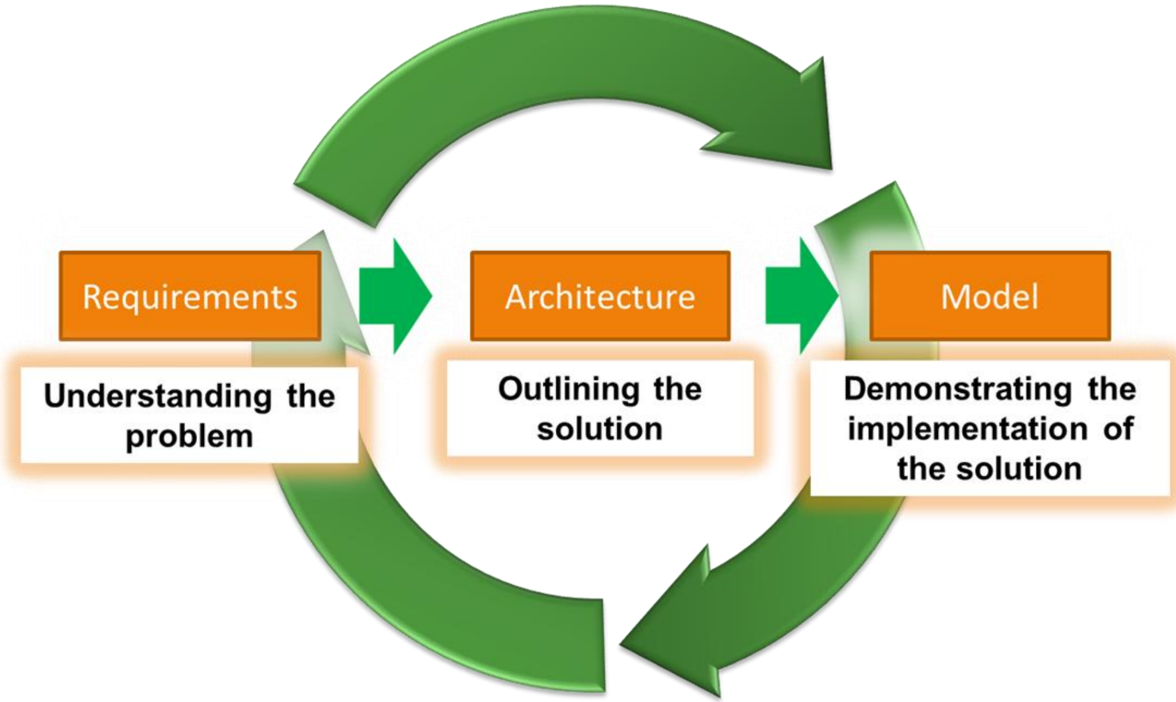
Property Patterns Classes

| | |
|---|---|
| **Occurrence** | Absence |
| | Universality |
| | Existence |
| | Bounded Existence |
| **Order** | Precedence |
| | Response |
| | Chain Precedence |
| | Chain Response |

# Architecture




***Guarantee appropriate decisions with traceable evidence during the system architectural design***



# Architecture: AADL and AGREE

- The Architecture Analysis & Design Language (AADL)
  - Developed by SAE
  - Architecture modeling notation with well-defined semantics
- Assume Guarantee REasoning Environment (AGREE) plugins
  - Developed by University of Minnesota and Rockwell Collins
  - Part of the DARPA High-Assurance Cyber Military Systems (HACMS) program[1]

Assumptions → System Implementation → Guarantees

<u>Assumption</u>: something a system assumes about it's environment (inputs)

<u>Guarantee</u>: what you can assume about the system and the performance of the system (outputs)

1. Kathleen Fisher, "Using Formal Methods to Enable More Secure Vehicles: Tufts University", 16 September, 2014 DARPA's HACMS Program, URL: http://wp.doc.ic.ac.uk/riapav/wp-content/uploads/sites/28/2014/05/HACMS-Fisher.pdf [cited 27 Jul. 2015].

# AGREE

## Assume Guarantee REasoning Environment

- **Assume-Guarantee Contract -** Verifiable set of Assumptions and Guarantees that abstracts the behavior of a system component implementation

- **Assumptions**

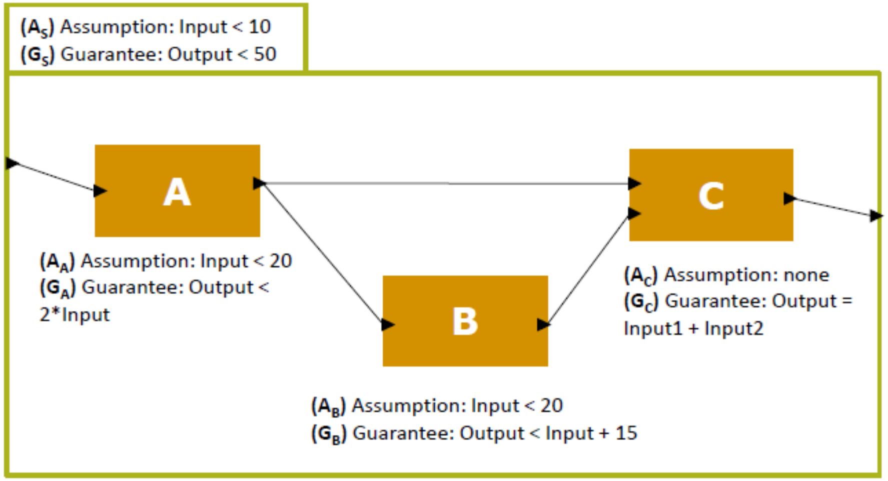
  Constraints over what a component expects to see from its environment

- **Guarantees**

  Constraints over how a component behaves in response to its environment

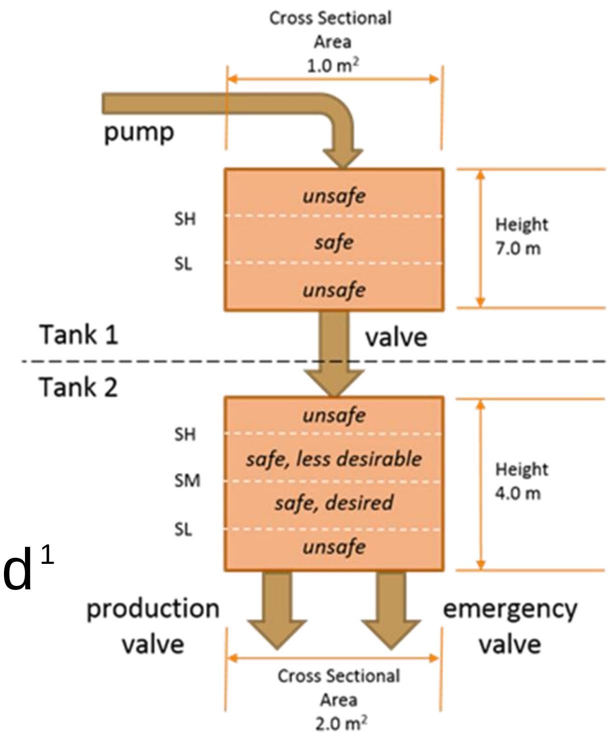
**Example (to prove)**

$A_S \rightarrow A_A$
$A_S \wedge G_A \rightarrow A_B$
$A_S \wedge G_A \wedge G_B \rightarrow A_C$
$A_S \wedge G_A \wedge G_B \wedge G_C \rightarrow G_S$

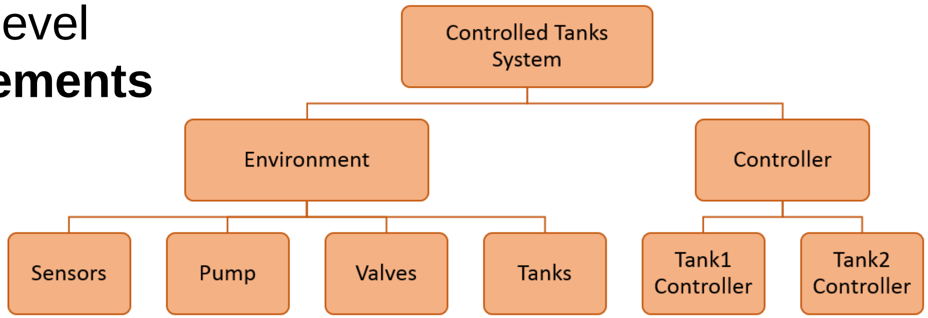


# Compositional Verification

- A series of techniques to allow for systems to be decomposed into less complex modules to be enforce a hierarchical structure that can be leveraged for compositional techniques




- Systems can be hierarchically organized[1]
  - Requirements vs. architectural design must be a matter of perspective
  - Need better support for *N*-level decompositions for **requirements** and **architectural design**



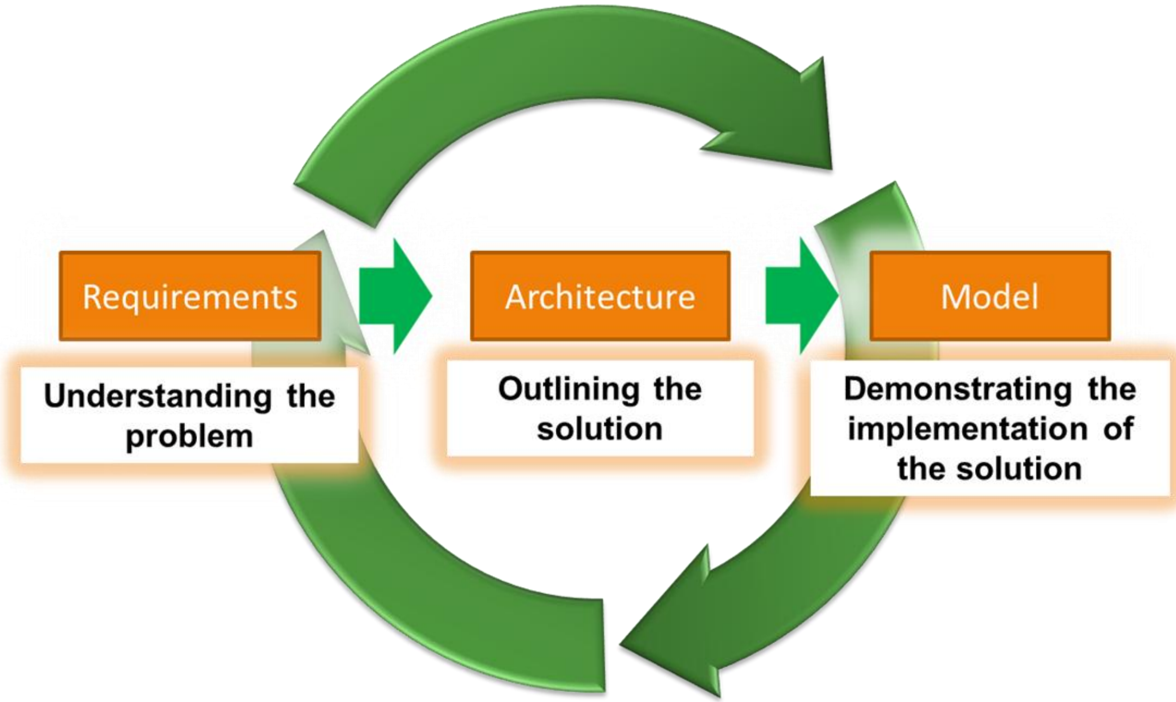

1. Whalen, Michael W., et al. "Your "What" Is My "How": Iteration and Hierarchy in System Design." Software, IEEE 30.2 (2013): 54-60.

# Model Development

CUMULATIVE EVIDENCE THROUGH RDT&E, DT & OT

Progressive sequential modeling, simulation, test and evaluation

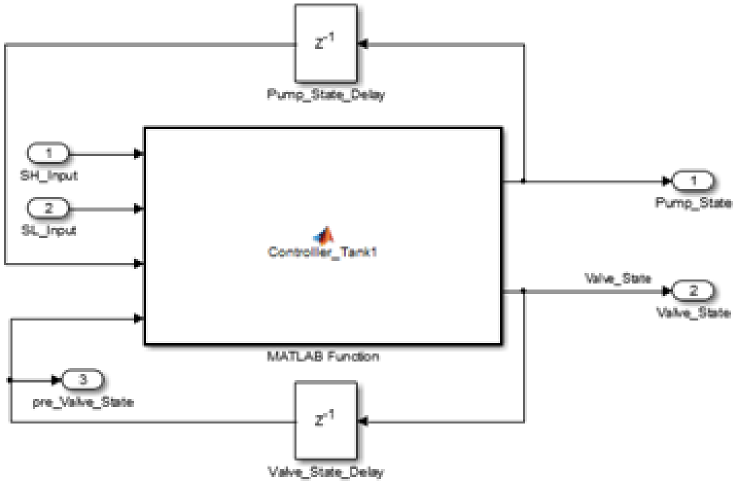
***Cumulative Evidence Through Research, Developmental, and Operational Test***

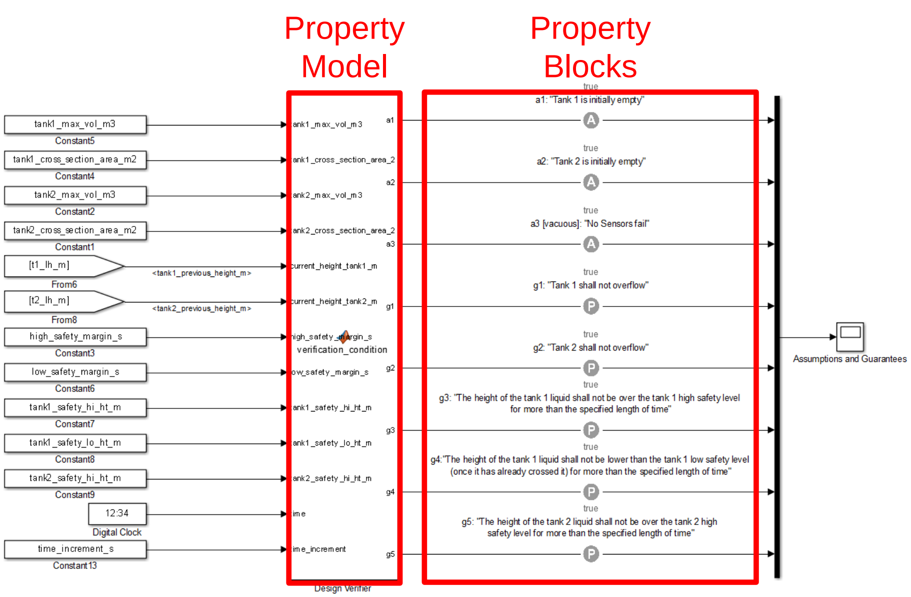


# Introduce Simulink and SLDV


- Uses formal methods to find **violations of design properties and assumptions**
- Formal Analysis techniques from:
  - Prover Plug-In
  - Polyspace formal analysis engine from MathWorks

# SLDV Analysis




## Property Model

```
%% Guarantees (Proof Objectives)
% G01: guarantee "The pump is initially off"
g1 = pump_initially_off(pre_Pump_State, time);
function result = pump_initially_off(Pump_State, time)
    UnderThisCondition = (time == 0.0);
    ResultShouldBe = (Pump_State == 0);
    result = implies(UnderThisCondition, ResultShouldBe);

% G02: guarantee "The valve is initially closed"
g2 = valve_initially_closed(pre_Valve_State, time);
function result = valve_initially_closed(Valve_State, time)
    UnderThisCondition = (time == 0.0);
    ResultShouldBe = (Valve_State == 0);
    result = implies(UnderThisCondition, ResultShouldBe);

% G03: guarantee "After the initial time step, When SL_Input is False,
the Pump shall be on and Valve shall be Closed"
g3 = sl_input_false_cond(SL_Input, Pump_State, Valve_State, time);
function result = sl_input_false_cond(SL_Input, Pump_State, Valve_State,
time)
    UnderThisCondition = (SL_Input == 0) && (time > 0);
    ResultShouldBe = (Valve_State == 0) && (Pump_State == 1);
    result = implies(UnderThisCondition, ResultShouldBe);

% G04: guarantee "When the SH_Input is true, the Pump shall be off and
Valve shall be open"
g4 = sh_input_true_cond(SH_Input, Pump_State, Valve_State);
function result = sh_input_true_cond(SH_Input, Pump_State, Valve_State)
    UnderThisCondition = (SH_Input == 1);
    ResultShouldBe = (Valve_State == 1) && (Pump_State == 0);
    result = implies(UnderThisCondition, ResultShouldBe);

% G05: guarantee "When the SL_Input is True and the SH_Input is False,
the Pump and Valve stay in their previous state"
g5 = sl_input_true_sh_input_false_cond(SL_Input, SH_Input, Pump_State,
Valve_State, pre_Pump_State, pre_Valve_State);
function result = sl_input_true_sh_input_false_cond(SL_Input, SH_Input,
Pump_State, Valve_State, pre_Pump_State, pre_Valve_State)
    UnderThisCondition = (SL_Input == 1) && (SH_Input == 0);
    ResultShouldBe = (Pump_State == pre_Pump_State) && (Valve_State ==
pre_Valve_State);
    result = implies(UnderThisCondition, ResultShouldBe);
```

# Requirements Traceability

## Requirement - SpeAR Property

```
g04 = while sensor_high
          :: always not pump_state and valve_state;
```

## Architecture - AGREE Guarantee

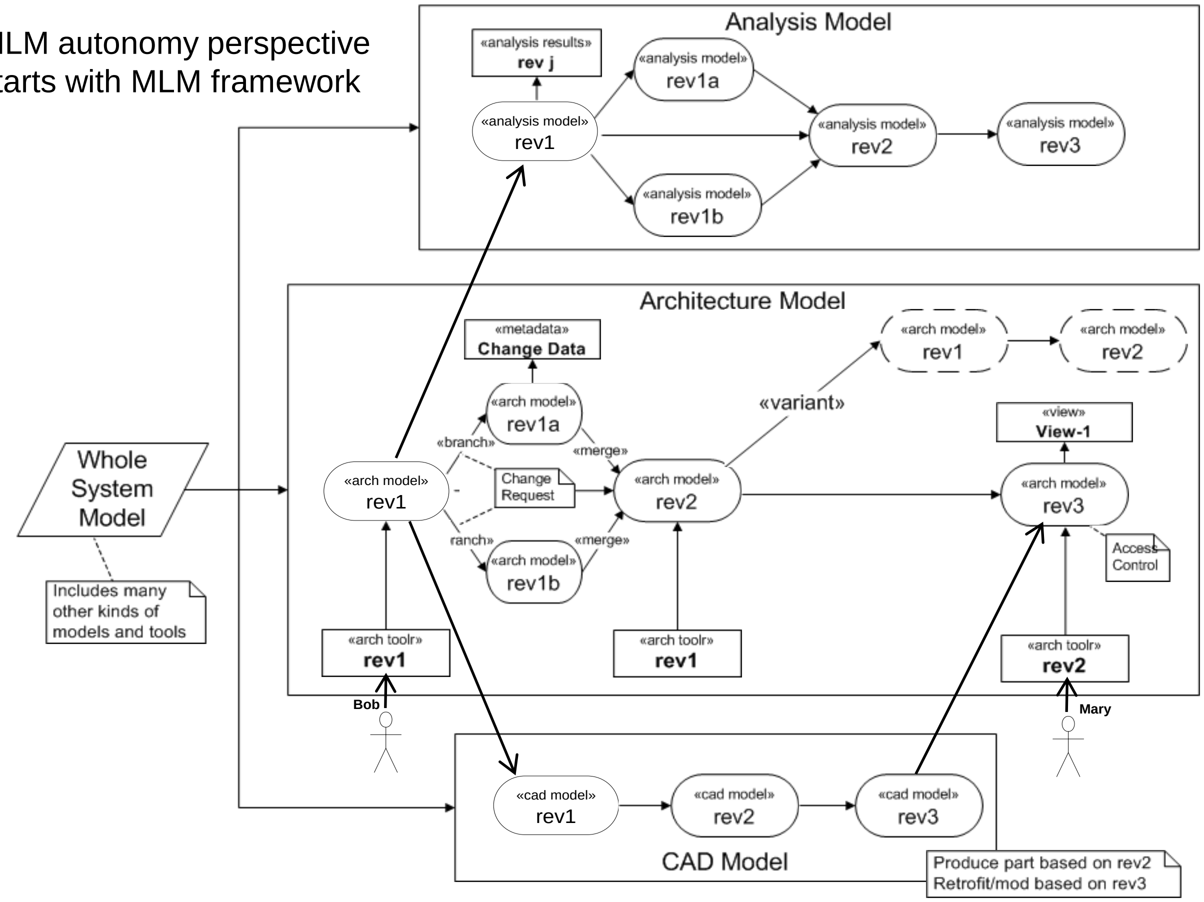
```
guarantee "G04: After the initial time step,
           When the SH_Input is true,
           the Pump shall be off and Valve shall be open" :
    true -> ((tank1_SH_value = 1.0) =>
             ((Valve_State = 1.0) and Pump_State = 0.0));
```

## Modeling - Simulink Design Verifier Property

```
g4 = sh_input_true_cond(SH_Input, Pump_State, Valve_State);
function result = sh_input_true_cond(SH_Input, Pump_State, Valve_State)
    UnderThisCondition = (SH_Input == 1);
    ResultShouldBe = (Valve_State == 1) && (Pump_State == 0);
    result = implies(UnderThisCondition, ResultShouldBe);
```

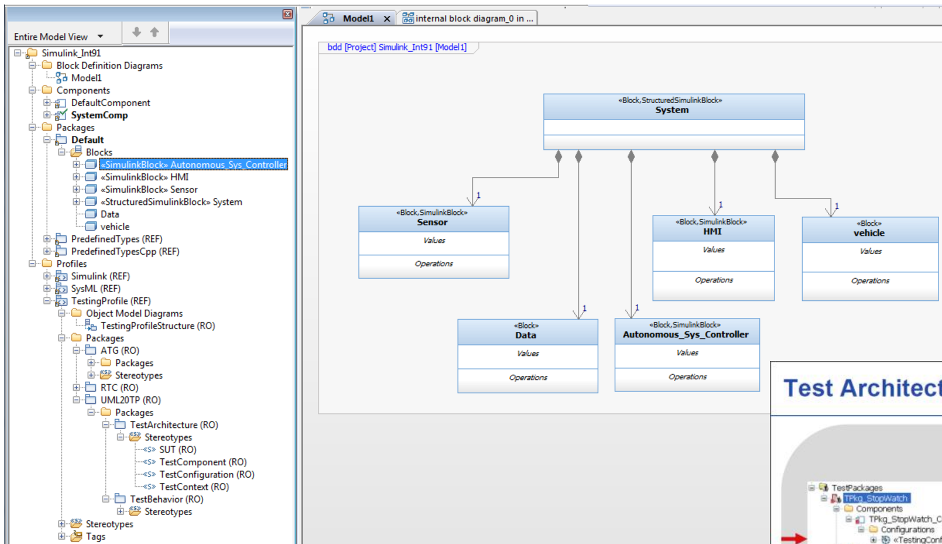
# Model Lifecycle Management Perspective

MLM autonomy perspective starts with MLM framework

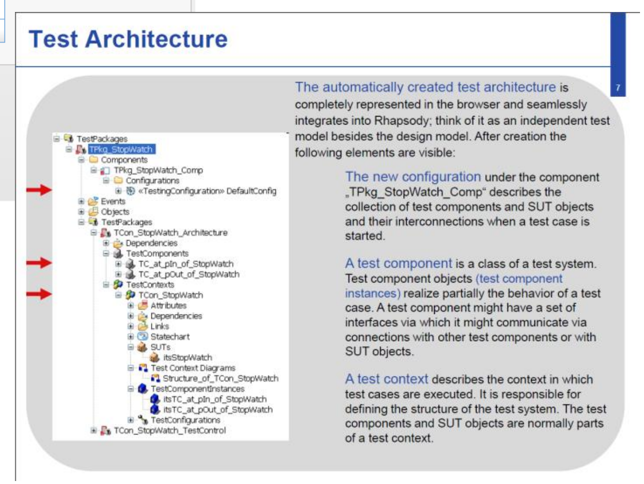


# SysML Representation of Autonomous System and Autonomous System Development

**Raytheon**
**Integrated Defense Systems**




- Building on the MLM framework
- Nominal autonomous system modeled in SysML (Rhapsody example)




- UML Test Protocol or similar utility is used
- Enables efficient pairing of requirements, test straps, procedures, reports, and other artifacts with each member of a product family
- Models are executable within modeling environment at chosen level of fidelity

# Basic example of Autonomous Systems T&E in MBE context

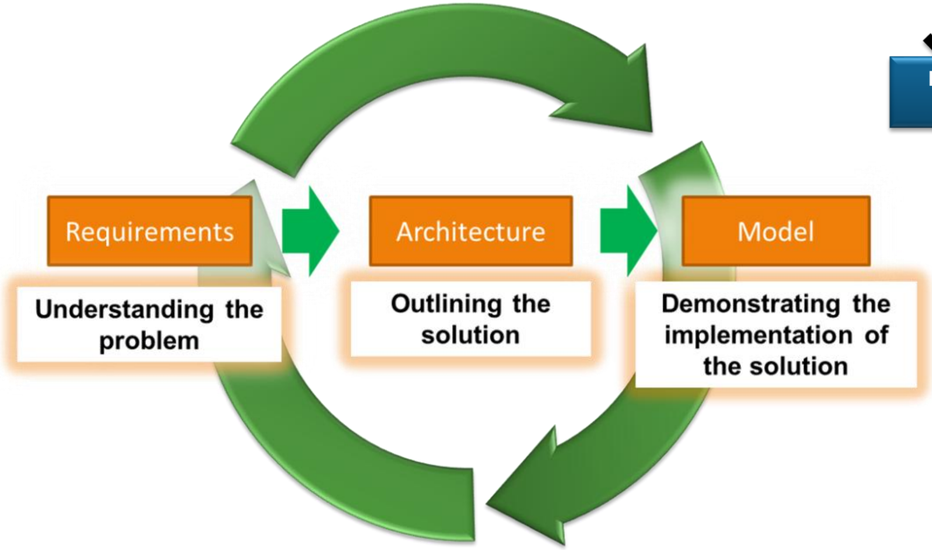
Raytheon
Integrated Defense Systems

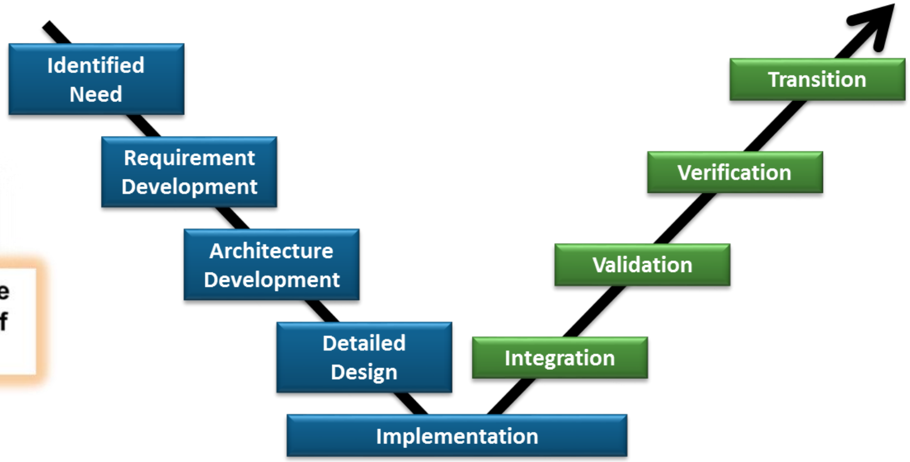
- Basic Machine Learning algorithm hosted in Simulink
- Data sets for nominal autonomous system developed
- Simulink components integrated within Rhapsody (SysML)
- Model executed in the SysML environment
- SysML test utilities placed around test and test results
  - IBM Test Conductor or potentially RQM wrapper
- Systems trained with different data sets behaved differently
- MBE considerations
  - Configuration management, Data management
  - Flexibility, product family architecture support
  - Training Data is paired with the autonomous system
    - Ability to trace system development back to the training data set used

**Autonomous systems development requires additional MBSE considerations**

# Summary

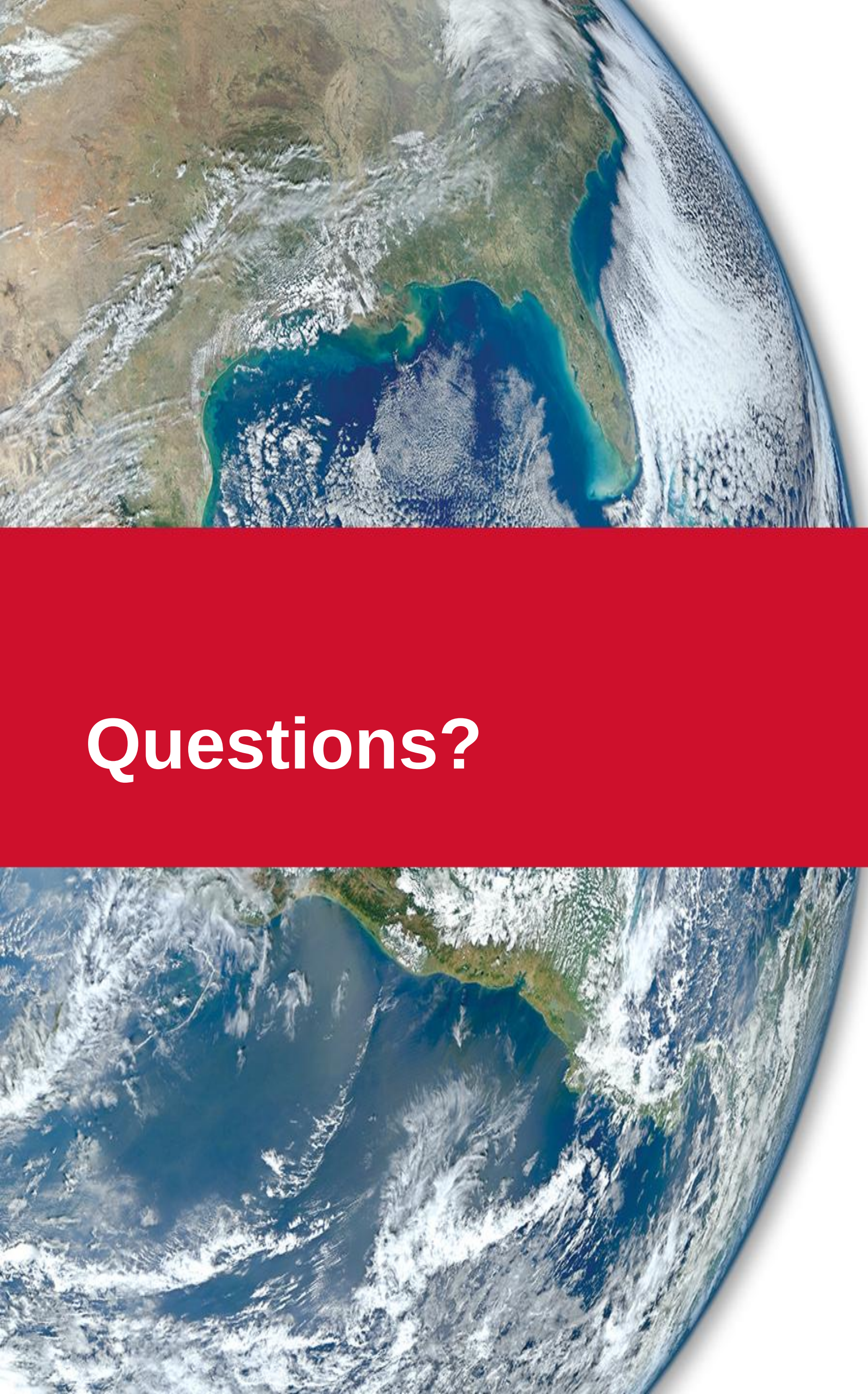
- Discovery of critical flaws early in the design process can save time and money
- Formal requirement traceability throughout design process
- Composability for reuse and modular verification
- Autonomous systems development requires additional MBSE considerations



# Future Directions of Work

- Continued research in the Development Process
  - Requirements
    - Realizability arguments could identify early conflicts
    - Natural language masking of formal representations
  - Architecture
    - Abstraction of different compositional levels across different teams
  - Modeling
    - Bounding nonlinear behavior within discrete defined systems
- Assurance Case Construction
  - Utilize the artifacts from the Development Process to provide evidence of behavior
    - Move the formulation forward with these artifacts
- Implementing the Development Process on more complex systems
  - Testing the scalability of the techniques
  - Designing challenges that approach the complexity of Air Force domain systems
  - Potentially build on MBSE – autonomy structure
- Run-time Assurance for nonlinear autonomy
  - If we can't formally prove or test can we bound?
  - How can we safely bound a system?



# Questions?

aaron.fifarek.ctr@us.af.mil
jonathan.hoffman.2@us.af.mil
mknolan@raytheon.com