Tools for Formally Reasoning about Systems

June 9 2015

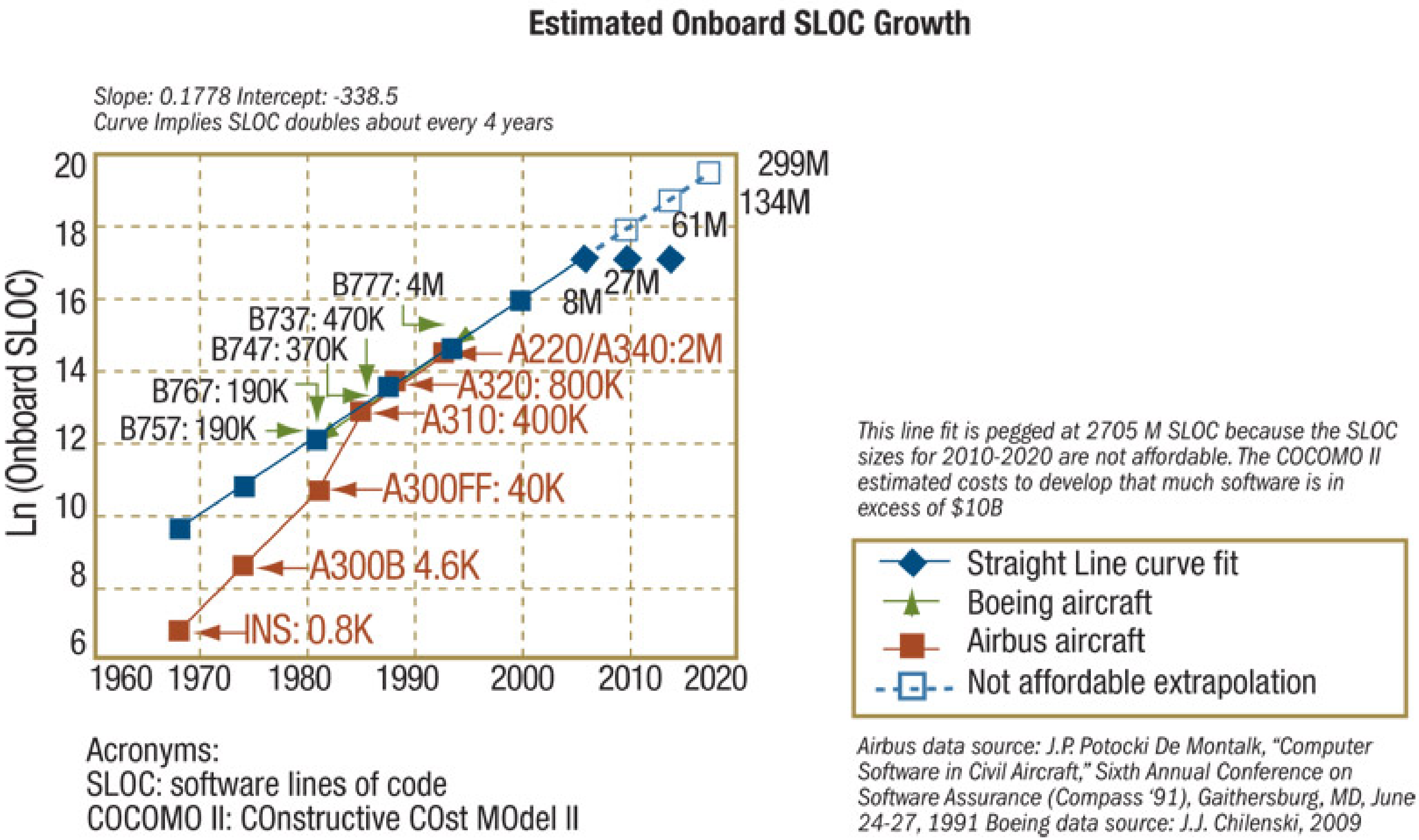
Prepared by Lucas Wagner

Rockwell Collins

Building trust every day

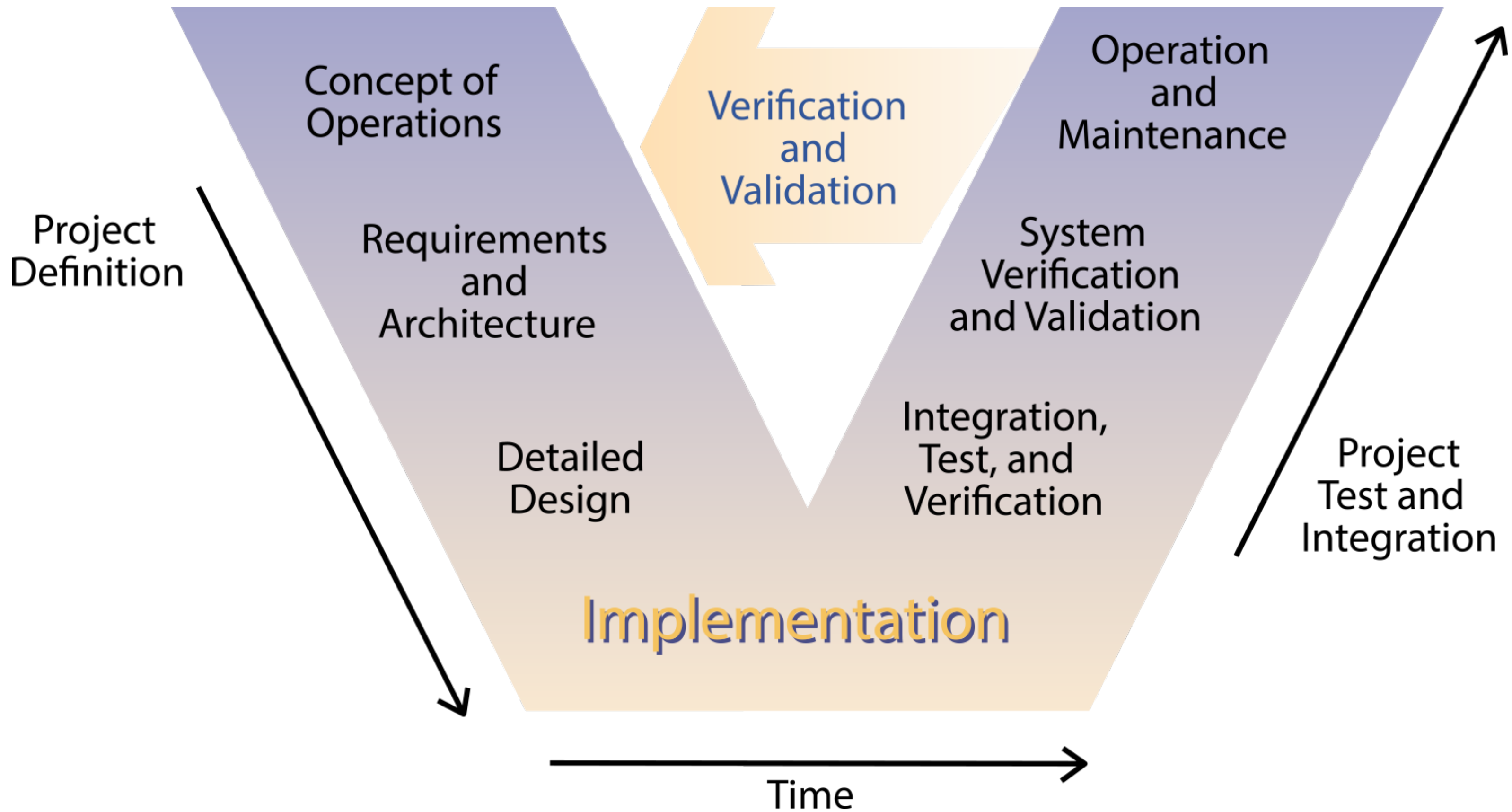
Rockwell Collins

Complex systems are getting more…complex



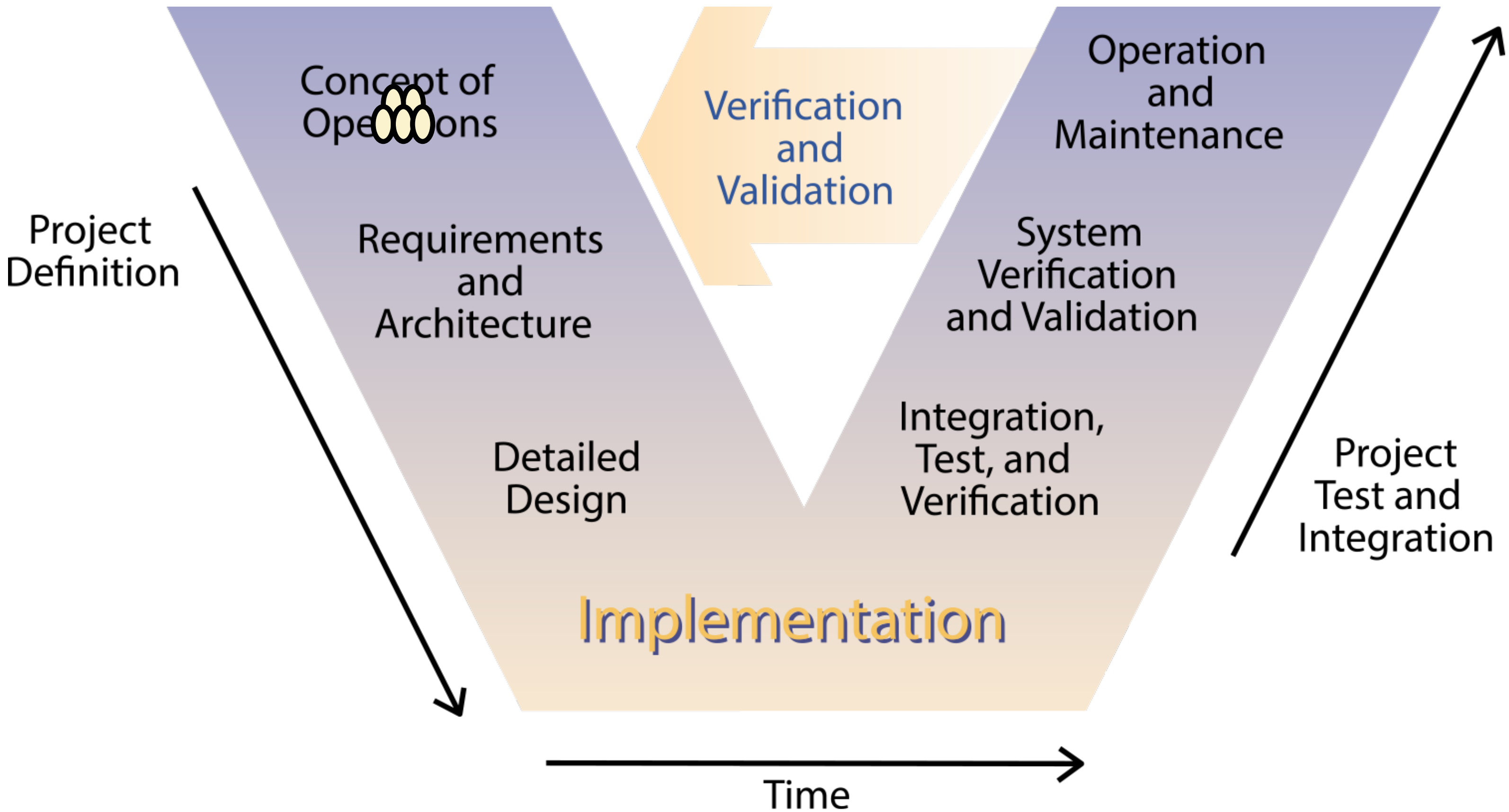
http://www.cotsjournalonline.com/articles/view/101090
Accessed: 05/27/2015

Complex systems are getting more…complex

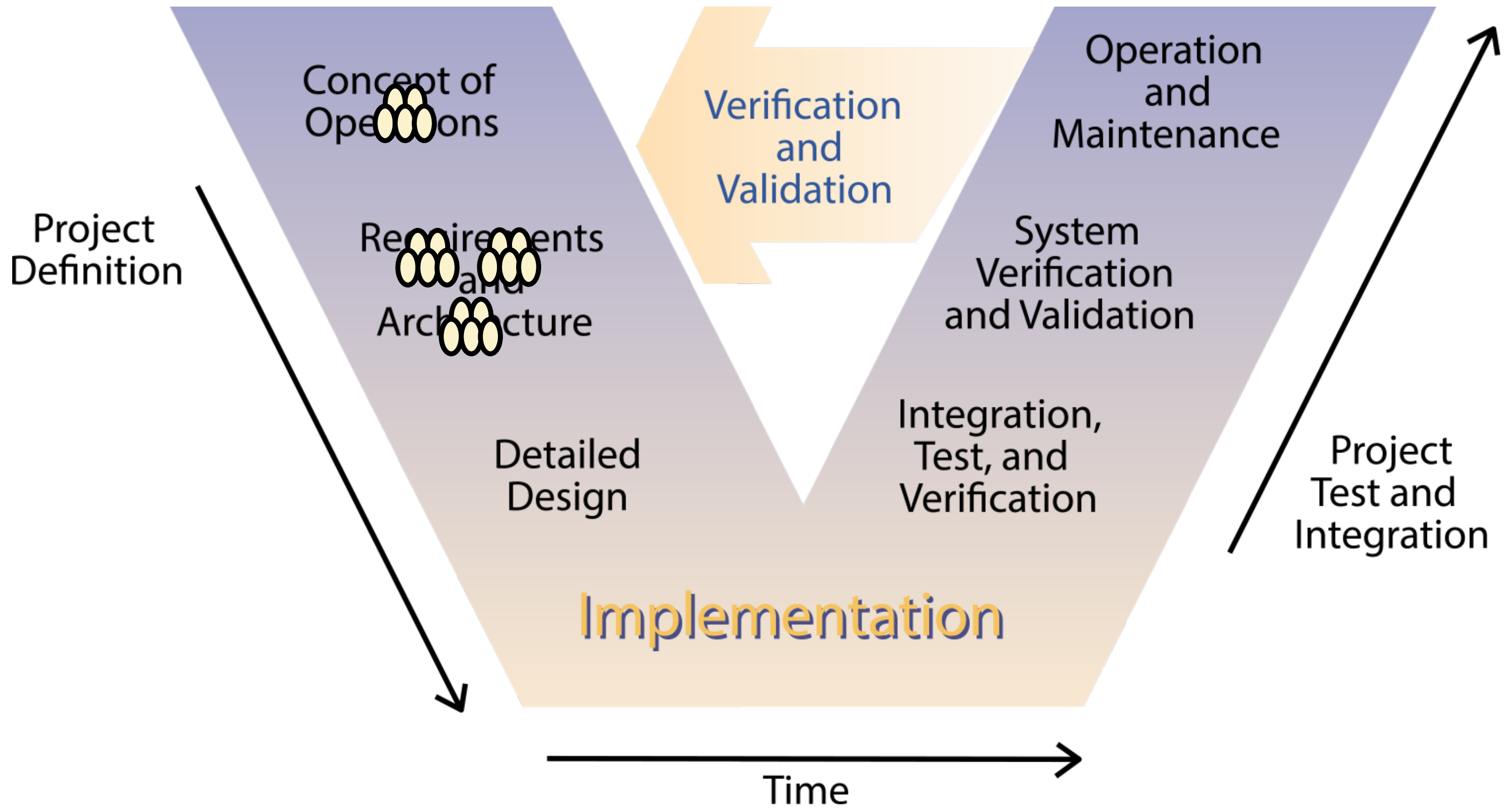


Rockwell Collins

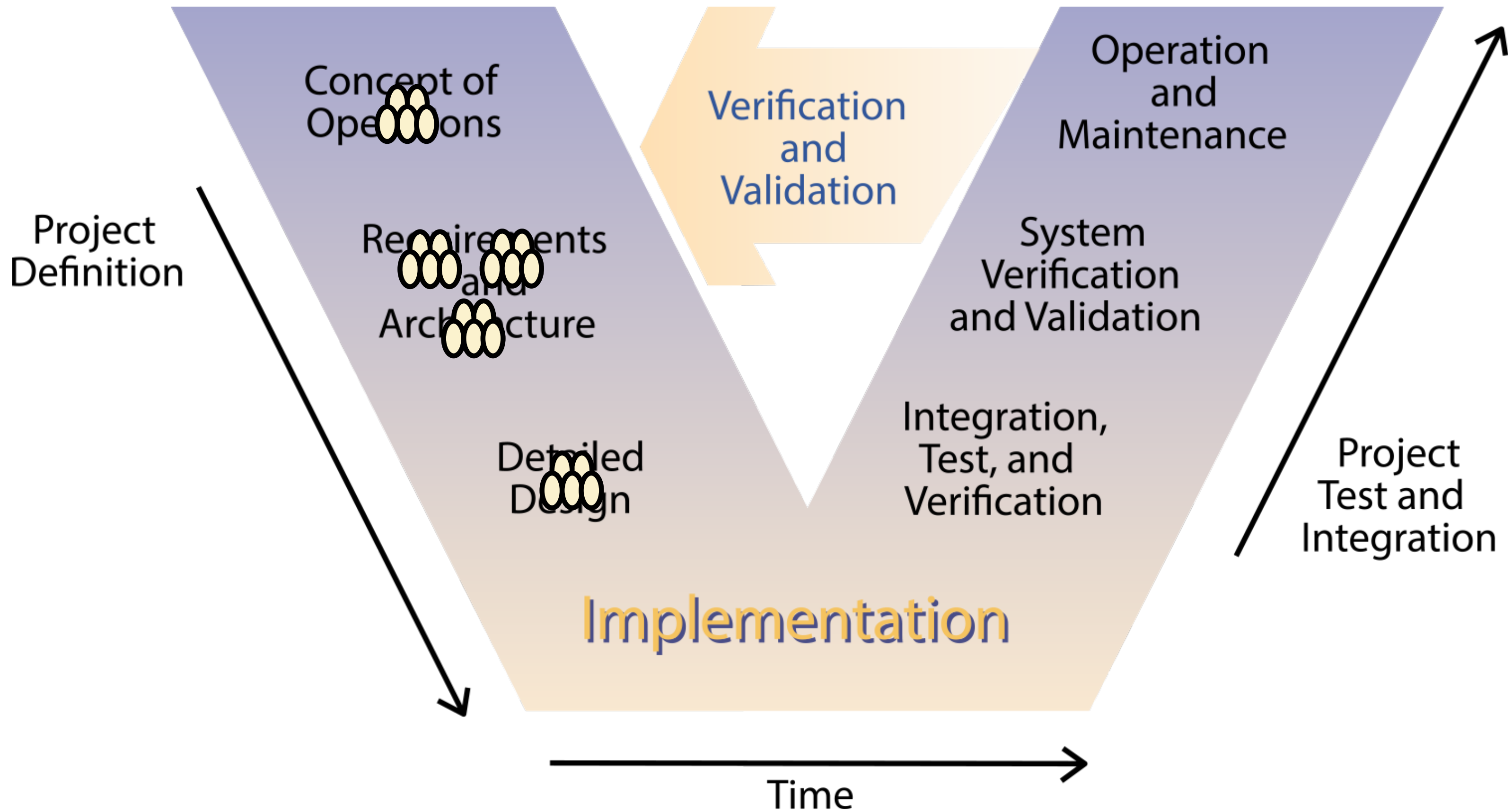
Complex systems are getting more…complex



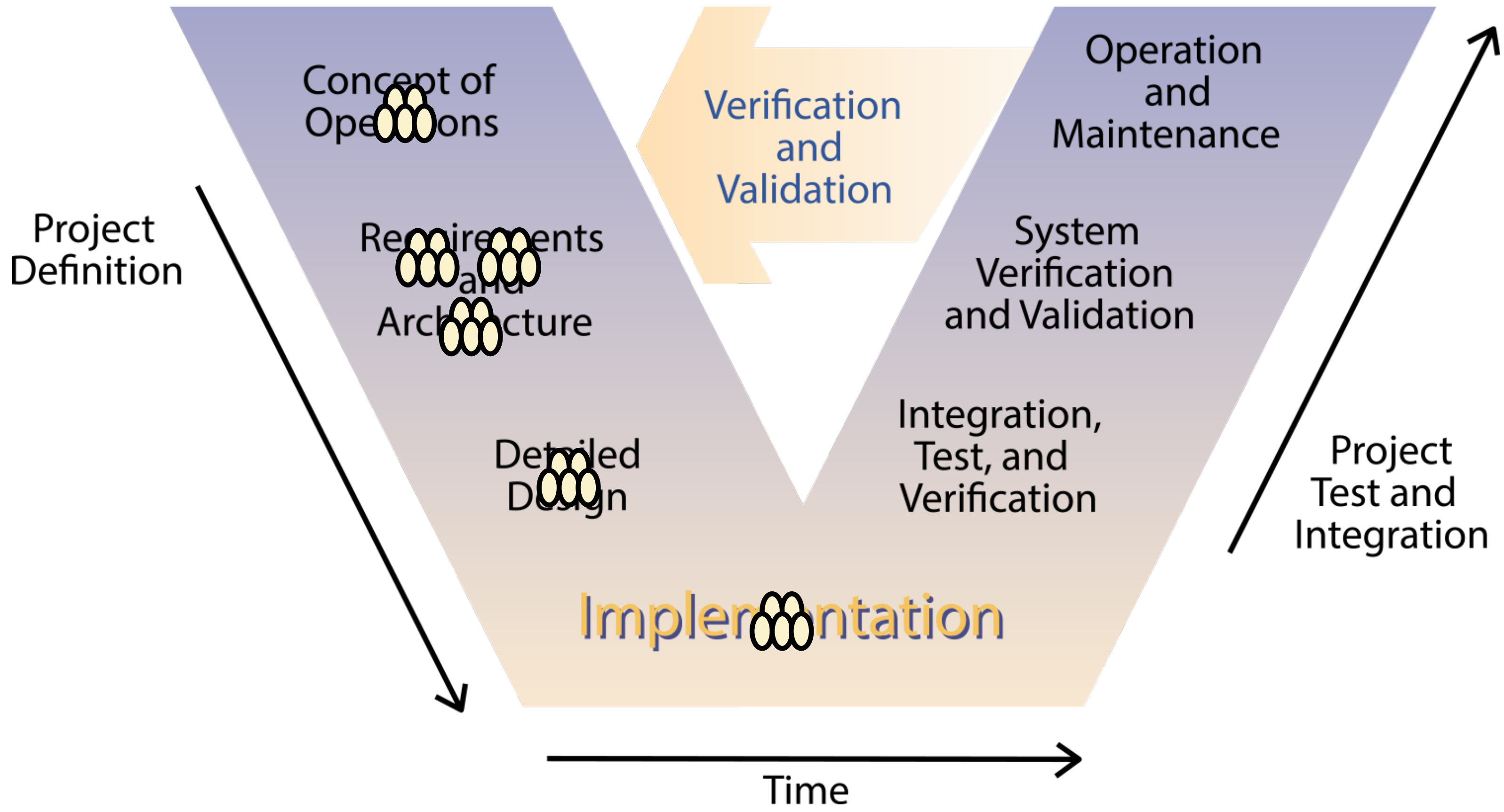
Complex systems are getting more…complex



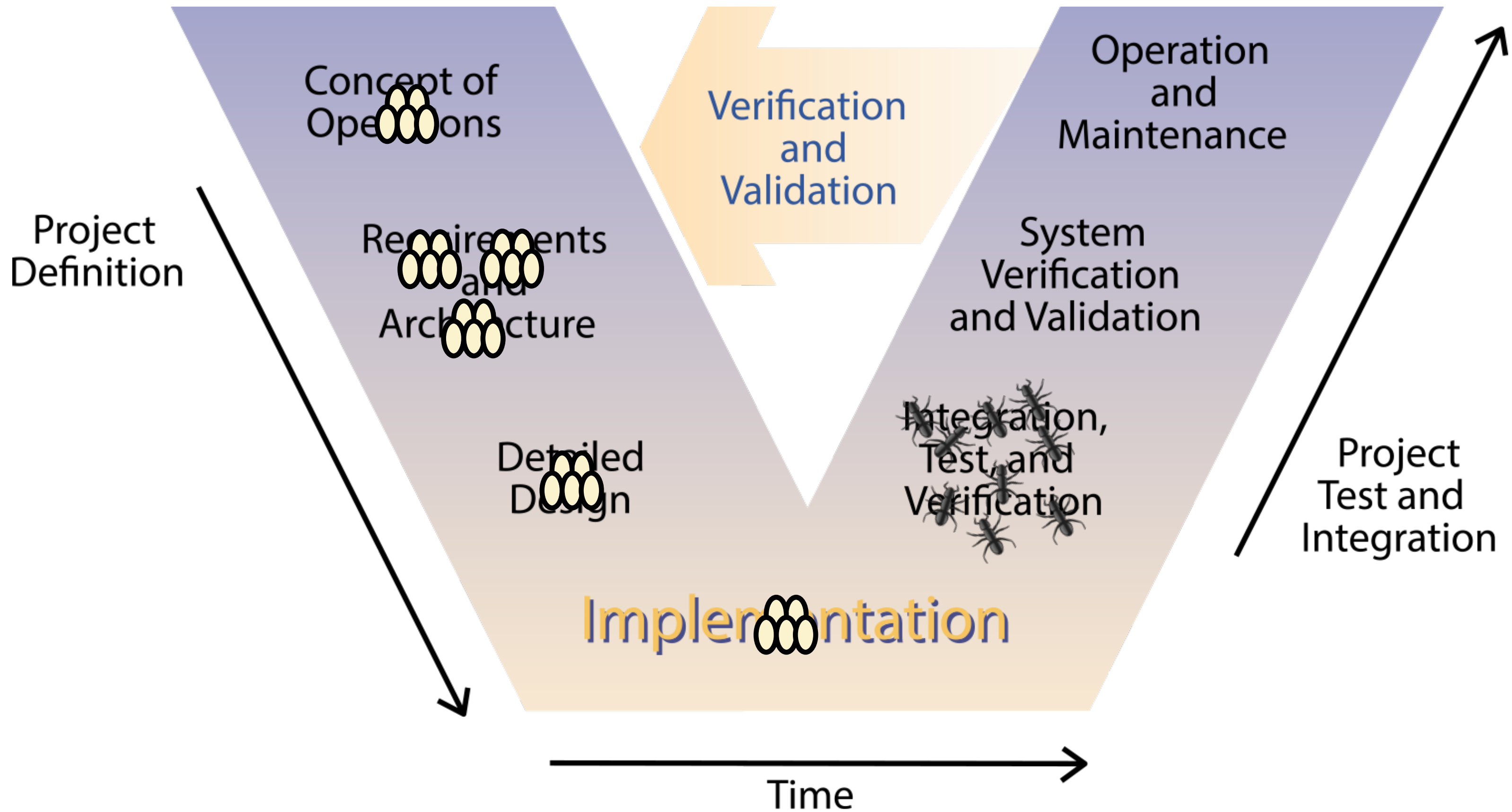
Complex systems are getting more…complex



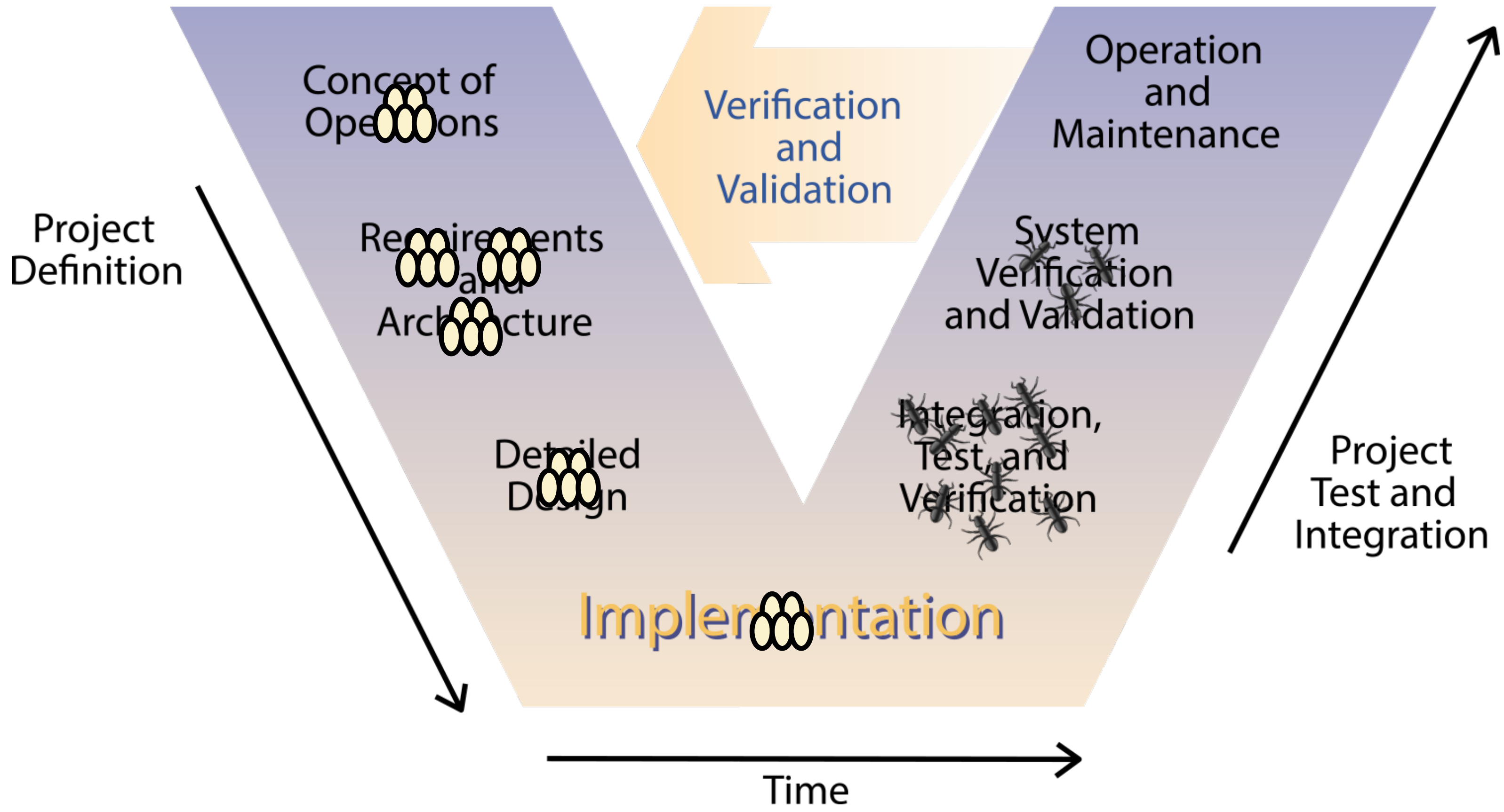
Complex systems are getting more…complex



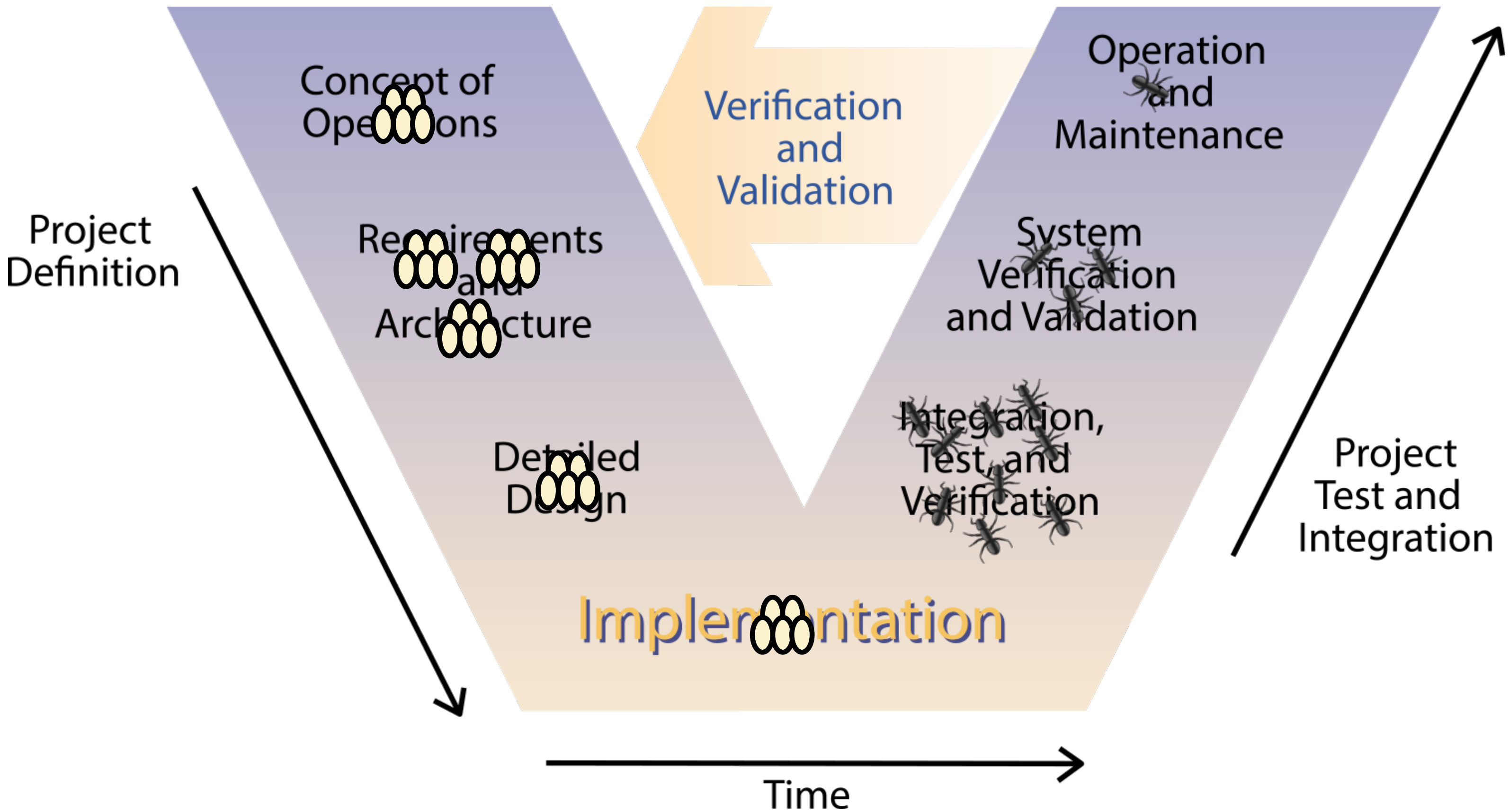
Complex systems are getting more…complex



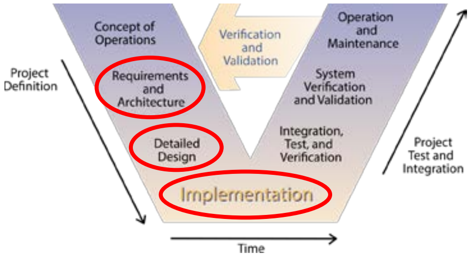
Complex systems are getting more…complex



Complex systems are getting more…complex

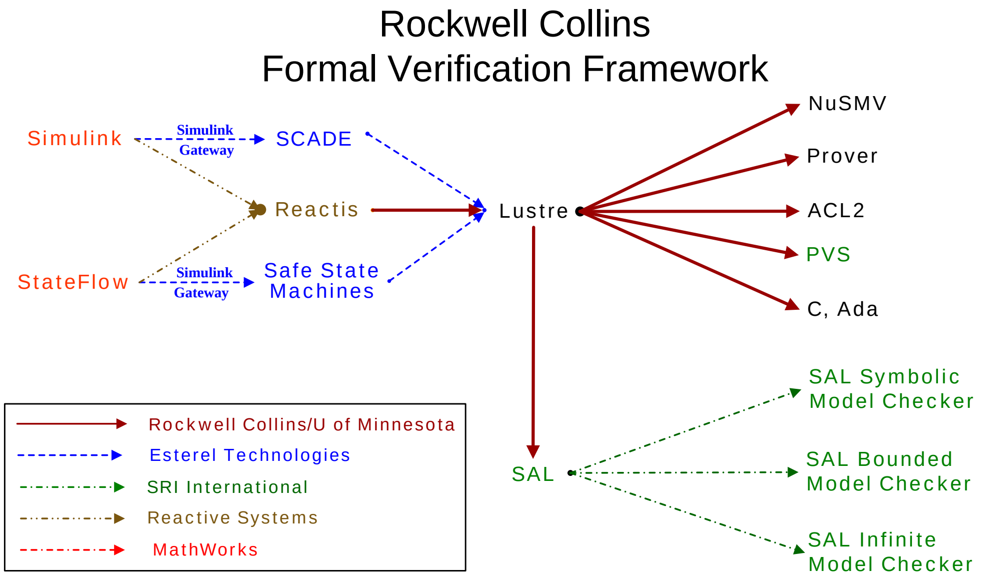


MBD and Formal Methods

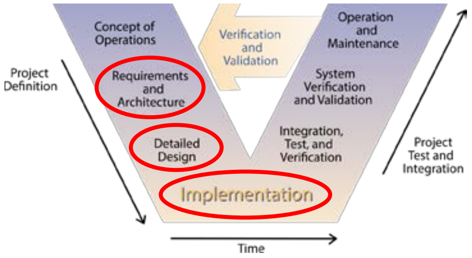


Model-based development tools have emerged to help reduce cycle-times on the traditional V-model for software by providing:

- Simulation
- Static analyses of components
  - Model checking
  - Abstract interpretation
- Test case generation
- Generate code



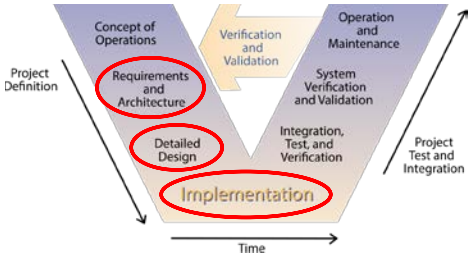
Formal Analysis of Components



Many tools exist now to analyze various software artifacts including:

- Graphical modeling
  - Simulink® Design Verifier™
  - SCADE Suite Design Verifier
- Source code
  - Polyspace®
  - Coverity®
  - Astree
  - CBMC
- Proprietary artifacts

Formal Analysis of Components

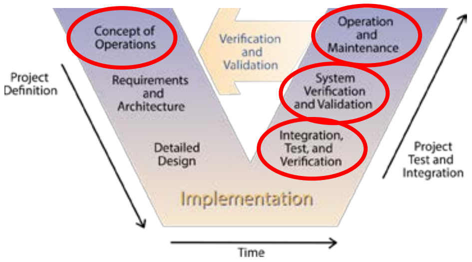


Many tools exist now to analyze various software artifacts including:

- Graphical modeling
  - Simulink® Design Verifier™
  - SCADE Suite Design Verifier
- Source code
  - Polyspace®
  - Coverity®
  - Astree
  - CBMC
- Proprietary artifacts

> …but these analyses do not scale up to the system level, and probably won't for quite some time.

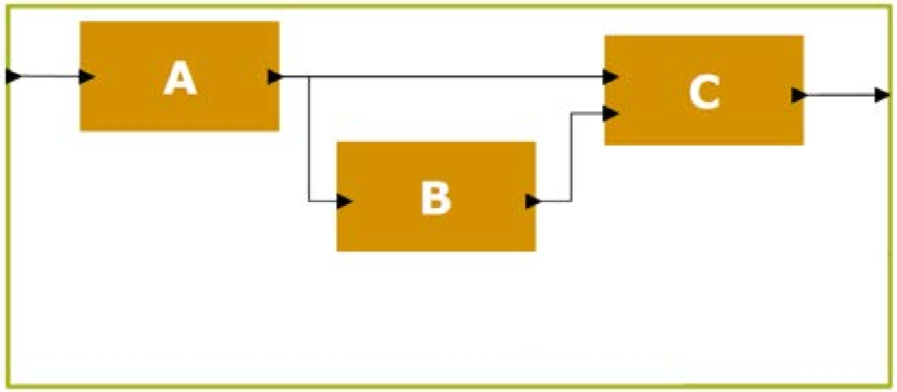
Beyond Component Reasoning



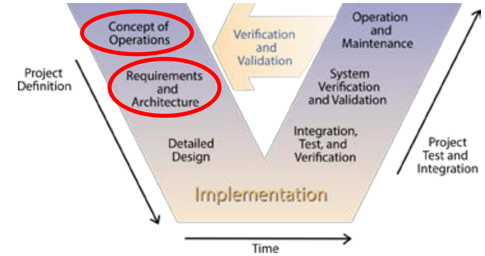
What can be done with formal analysis beyond the component level to improve the quality of our systems?

Rockwell Collins is developing methods, tools, and techniques for compositional analysis of systems.

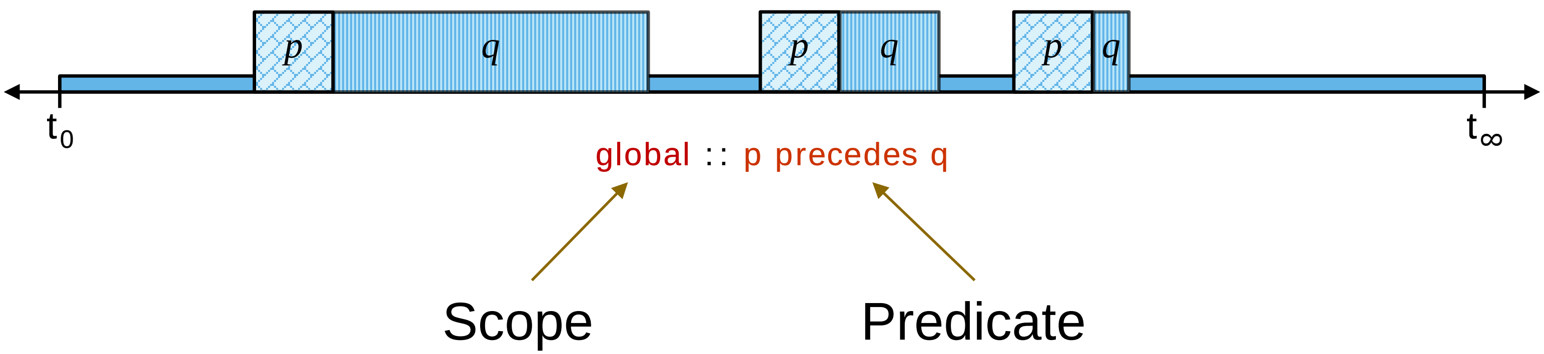
- SpeAR
- AADL/OSATE
  - AGREE
  - Resolute
- Backend analysis tools



SpeAR

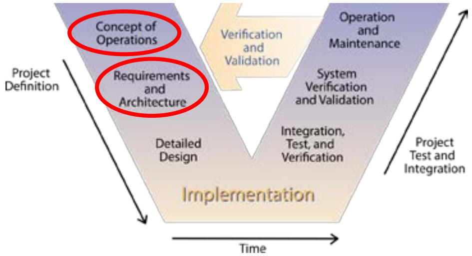


SpeAR (Specification and Analysis of Requirements) is a requirements prototyping and validation tool. It captures requirements as unambiguous formal specifications using frequently used specification patterns.



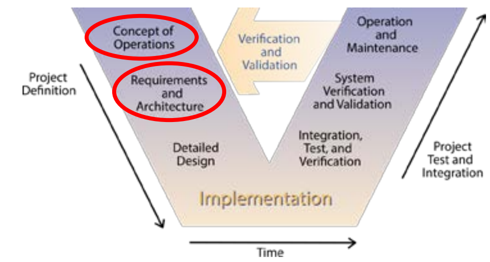
SpeAR supports the use of 6 scopes and 4 predicates that can be combined for the user to specify requirements.

SpeAR

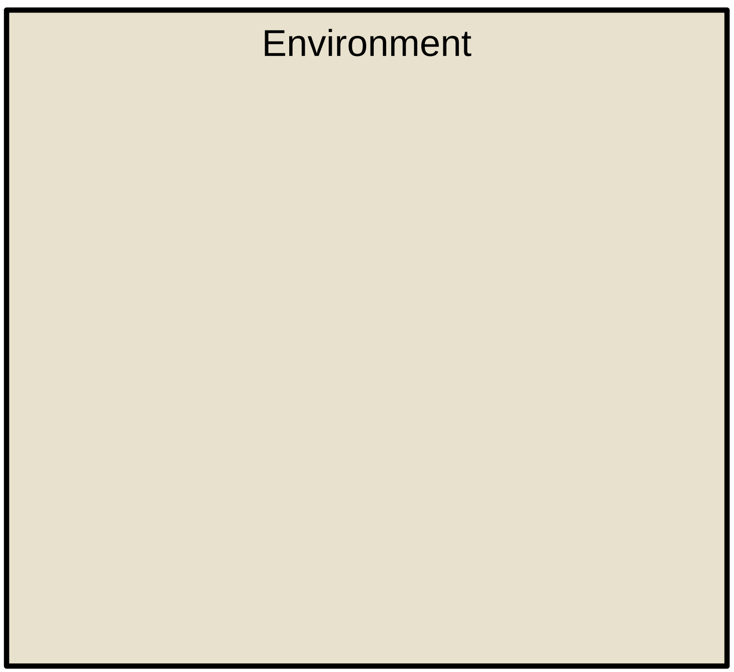


Once requirements are captured the user can check that a requirements set satisfies certain properties.

SpeAR

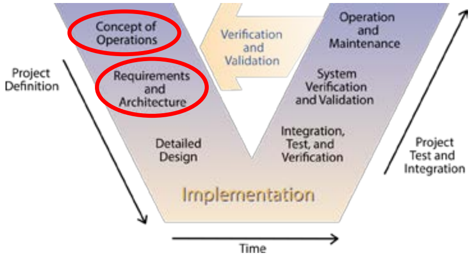


Once requirements are captured the user can check that a requirements set satisfies certain properties.

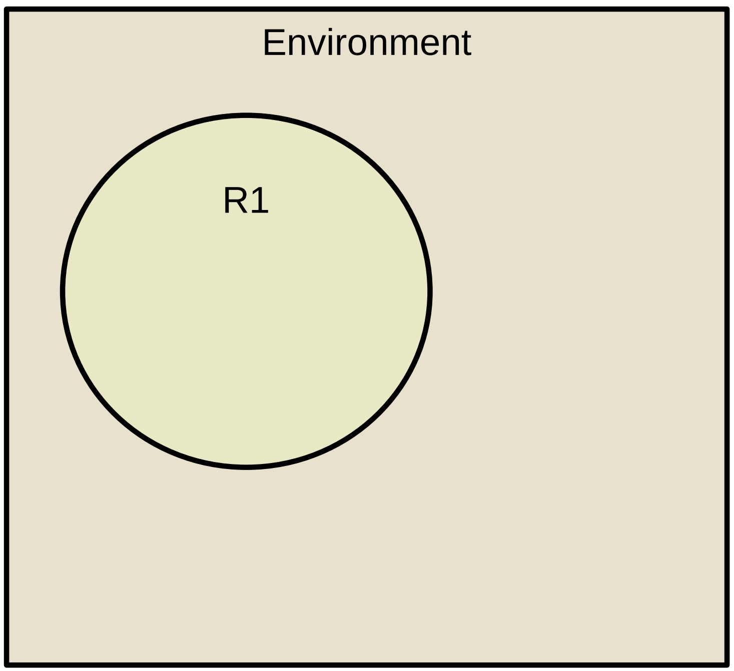


Rockwell Collins

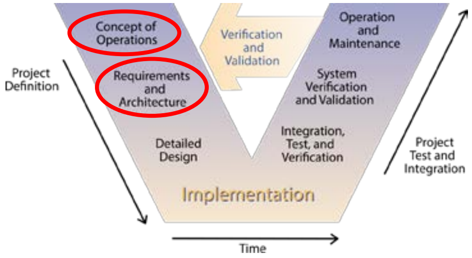
SpeAR



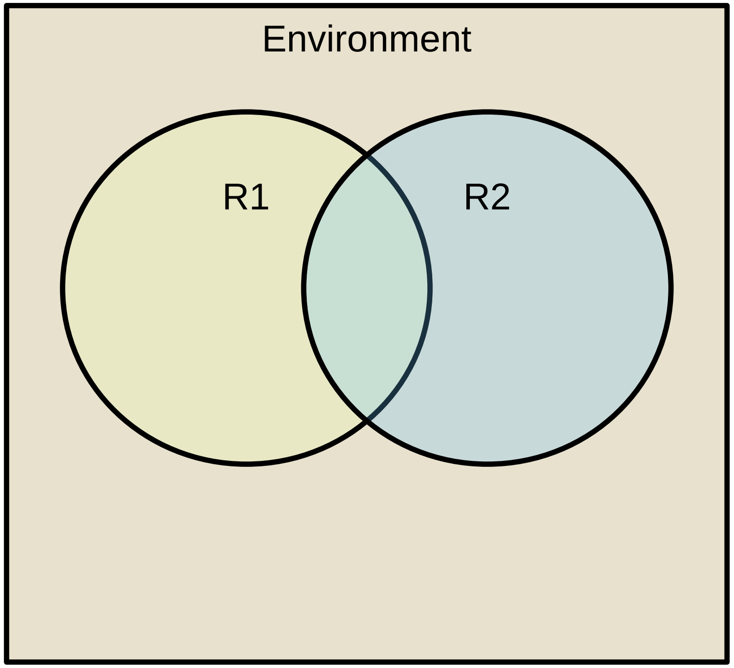
Once requirements are captured the user can check that a requirements set satisfies certain properties.



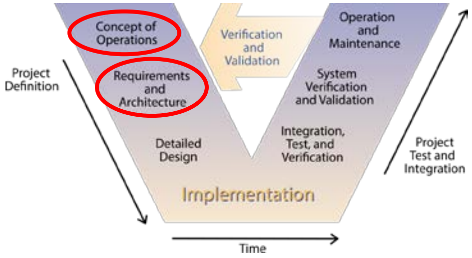
SpeAR



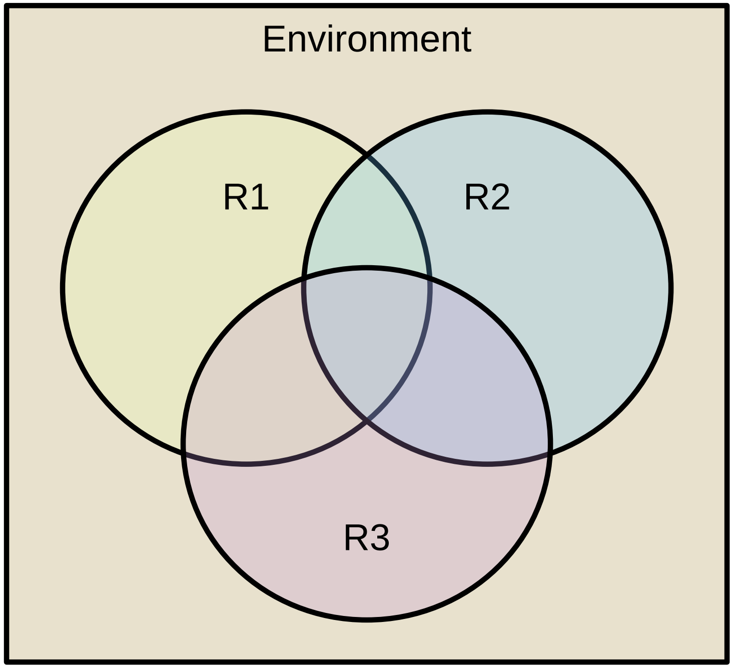
Once requirements are captured the user can check that a requirements set satisfies certain properties.



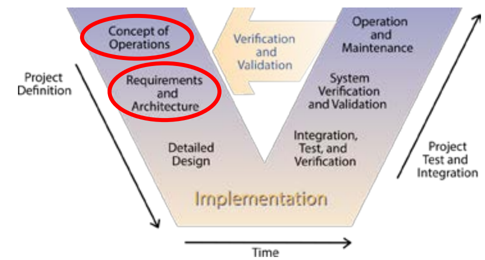
SpeAR



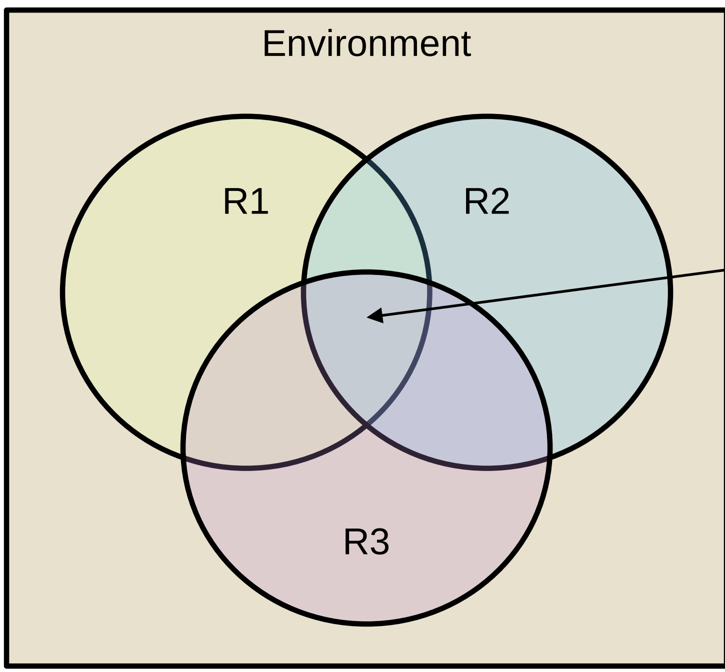
Once requirements are captured the user can check that a requirements set satisfies certain properties.



SpeAR



Once requirements are captured the user can check that a requirements set satisfies certain properties.



This represents the behaviors that our requirements allow as a set.

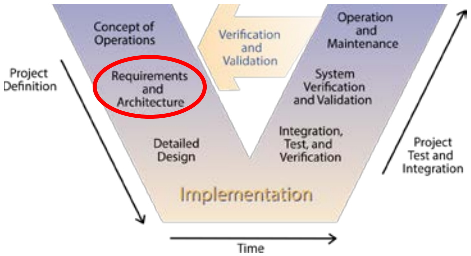
SpeAR allows the user to analyze whether these behaviors satisfy a given property.

Rockwell Collins

AADL / OSATE

AADL

Software Engineering Institute | Carnegie Mellon

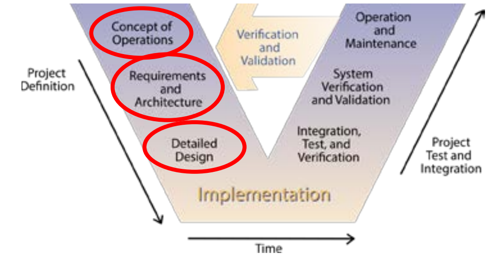


AADL is a language for specifying system architectures with features for capturing both software and hardware architectural concepts.

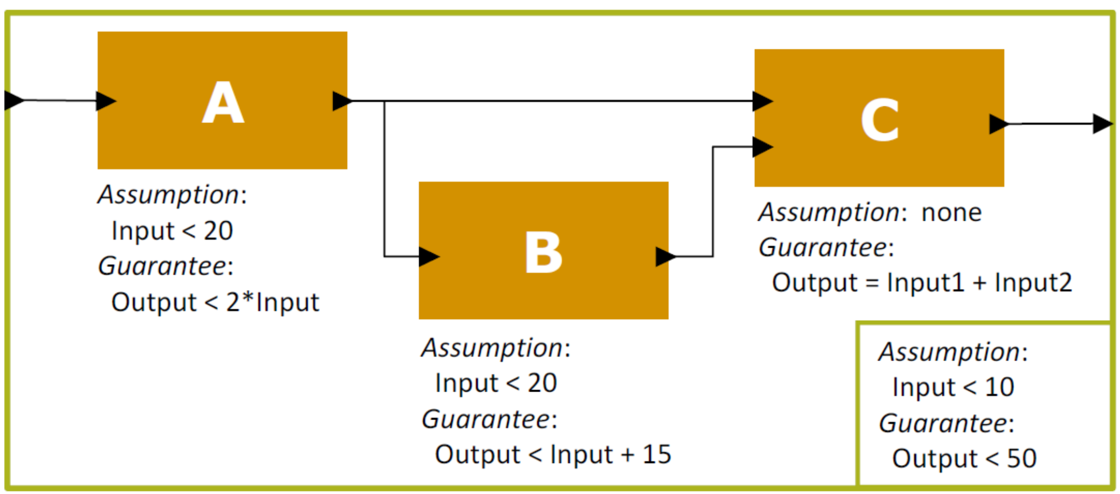
- SAE Standard AS5506
- Defined semantics
  - Improves team communication
  - Analysis tools can be built
- Extensible through the annex mechanism
- Tool support with the OSATE framework

Rockwell Collins

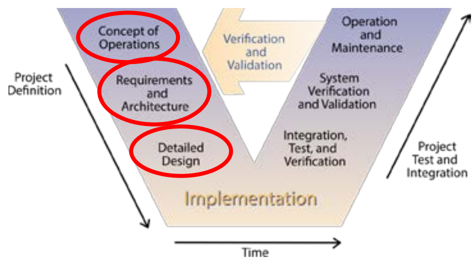
AGREE



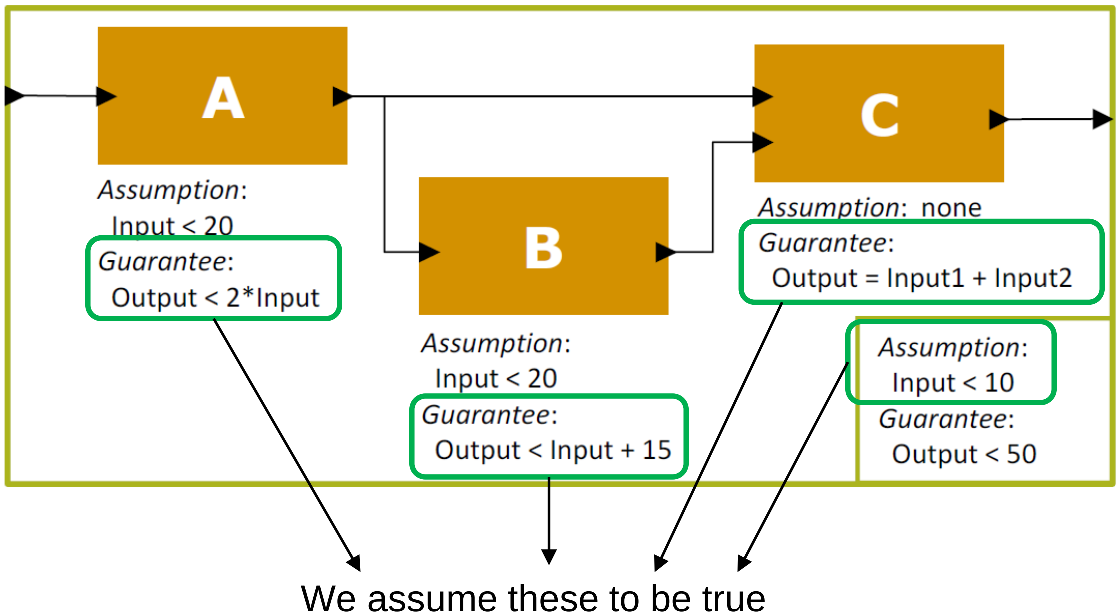
AGREE is a compositional reasoning tool for analyzing AADL models using model-checking.



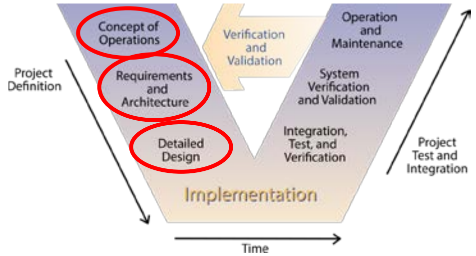
AGREE



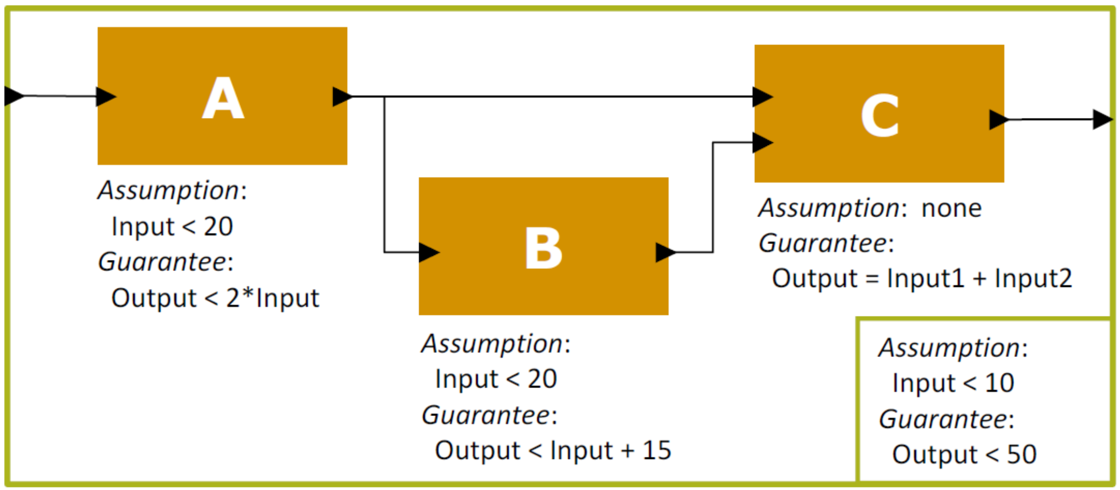
AGREE is a compositional reasoning tool for analyzing AADL models using model-checking.



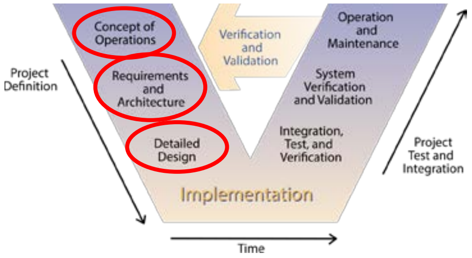
AGREE



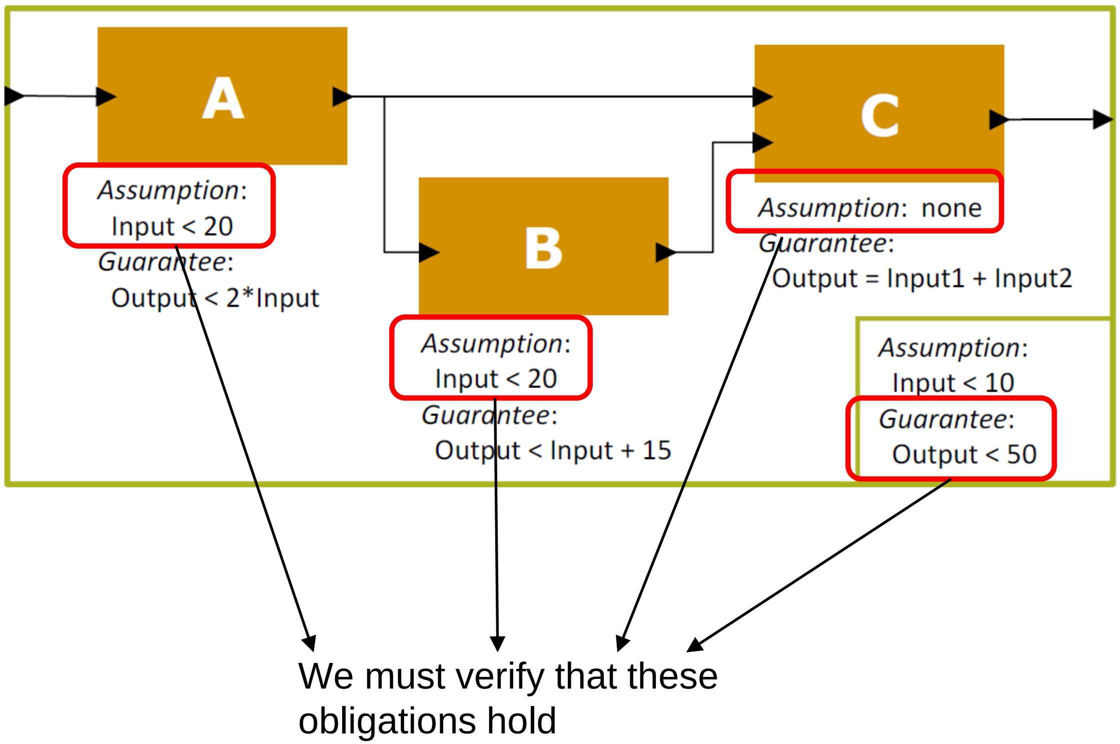
AGREE is a compositional reasoning tool for analyzing AADL models using model-checking.



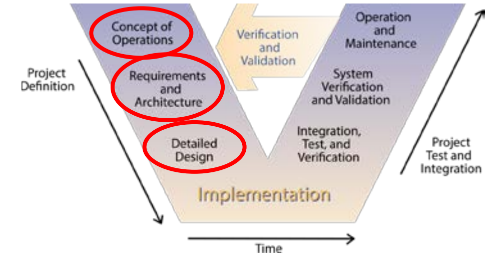
AGREE



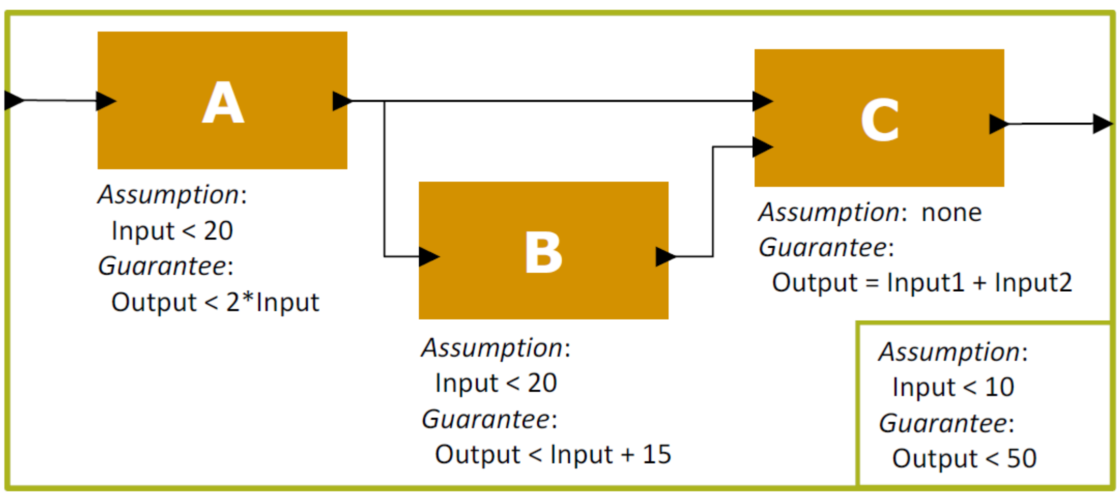
AGREE is a compositional reasoning tool for analyzing AADL models using model-checking.



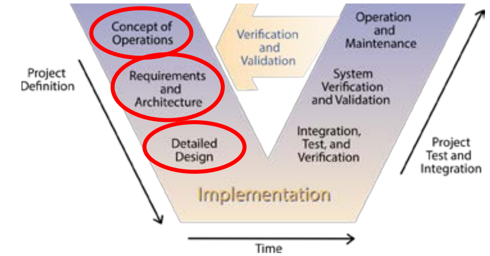
AGREE



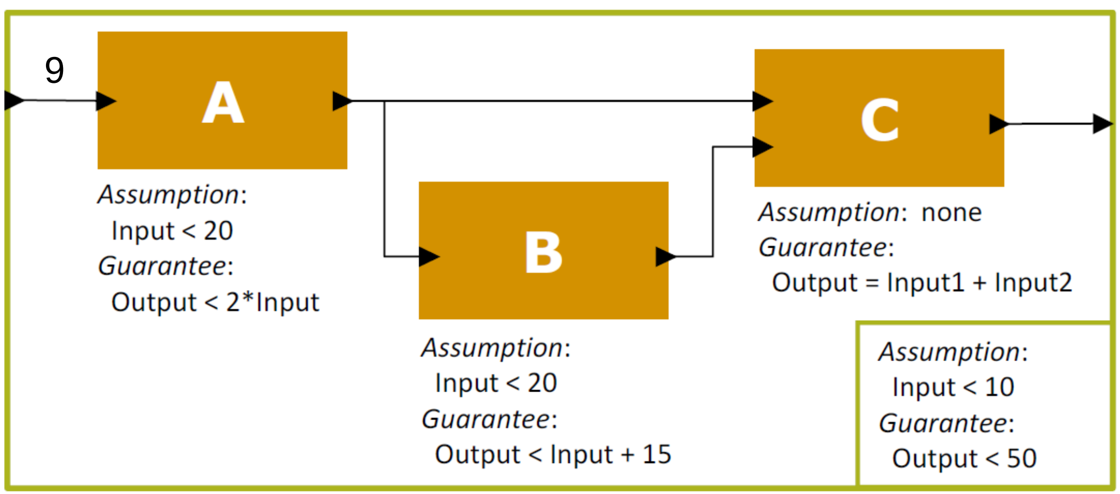
AGREE is a compositional reasoning tool for analyzing AADL models using model-checking.



AGREE

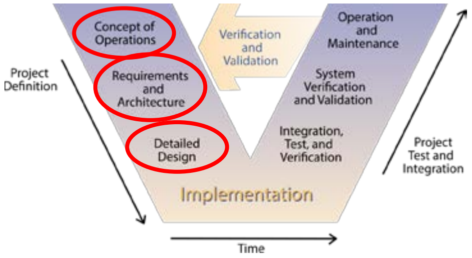


AGREE is a compositional reasoning tool for analyzing AADL models using model-checking.

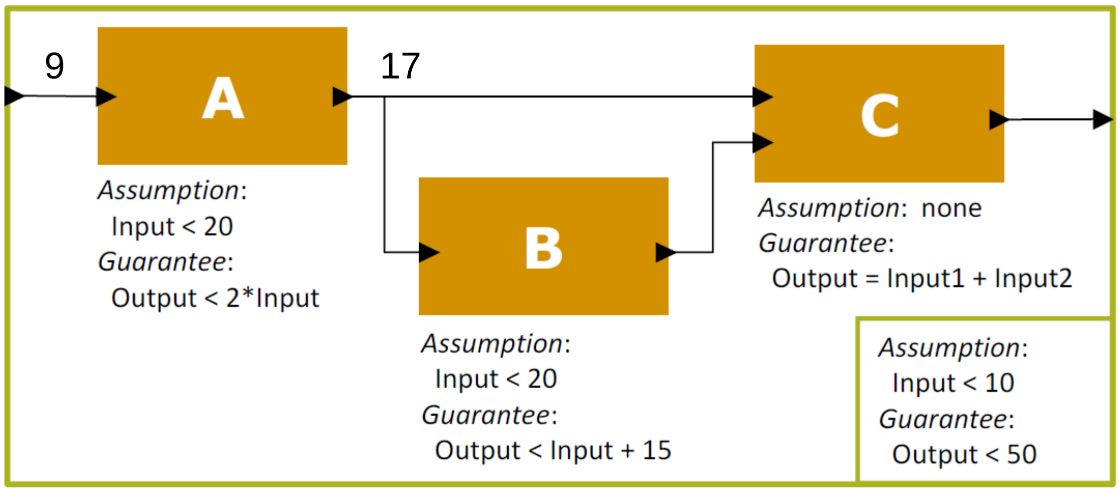


Rockwell Collins

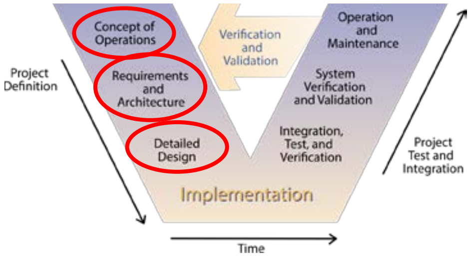
AGREE



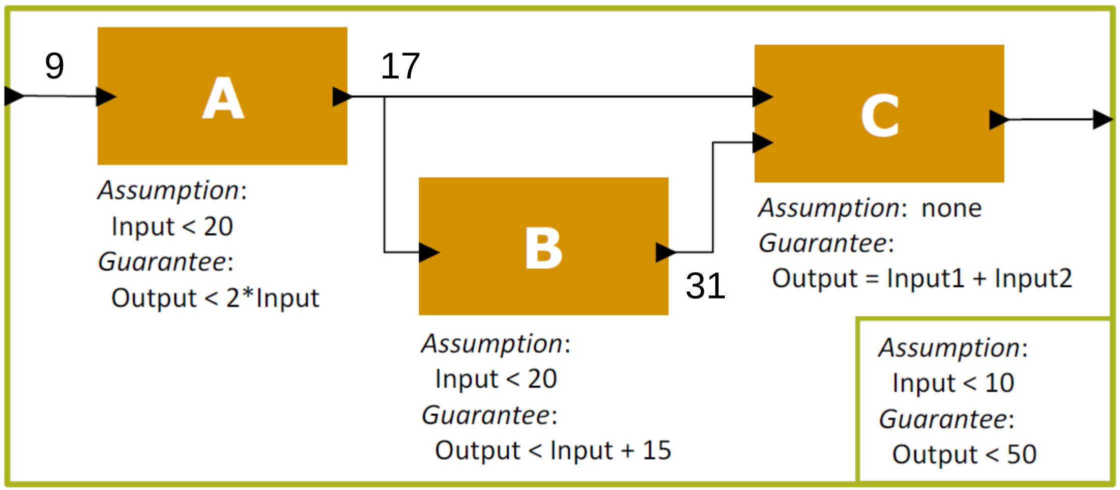
AGREE is a compositional reasoning tool for analyzing AADL models using model-checking.



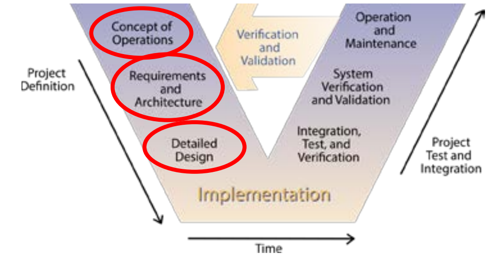
AGREE



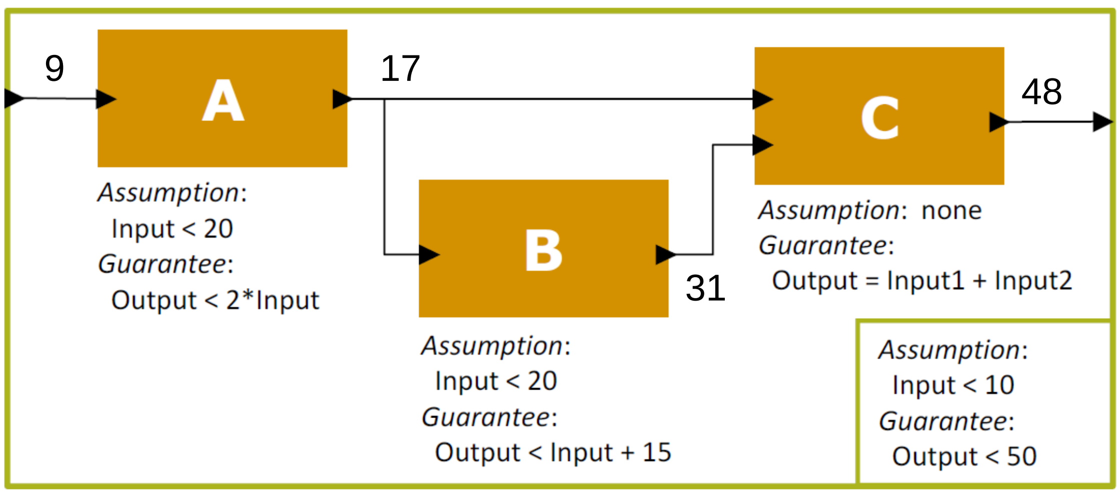
AGREE is a compositional reasoning tool for analyzing AADL models using model-checking.



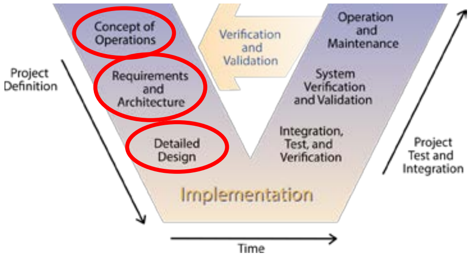
AGREE



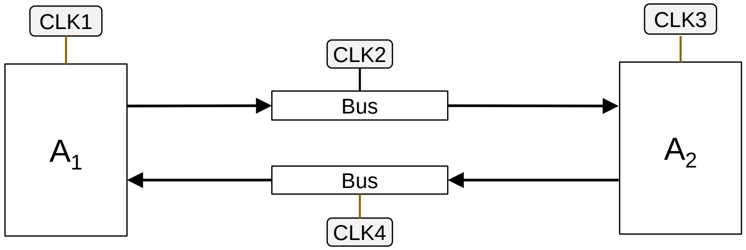
AGREE is a compositional reasoning tool for analyzing AADL models using model-checking.



AGREE supports relaxed synchrony



AGREE has been extended to reason about systems with relaxed synchrony constraints.



Systems must be implemented as redundant, fault-tolerant systems.

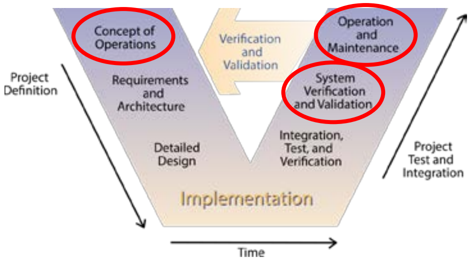
For distributed systems all nodes must agree on some portion of the global system state.

Quasi-Synchronous:

Clocks are not synchronized …

… but run at same period modulo their jitter and drift.

Assurance Cases

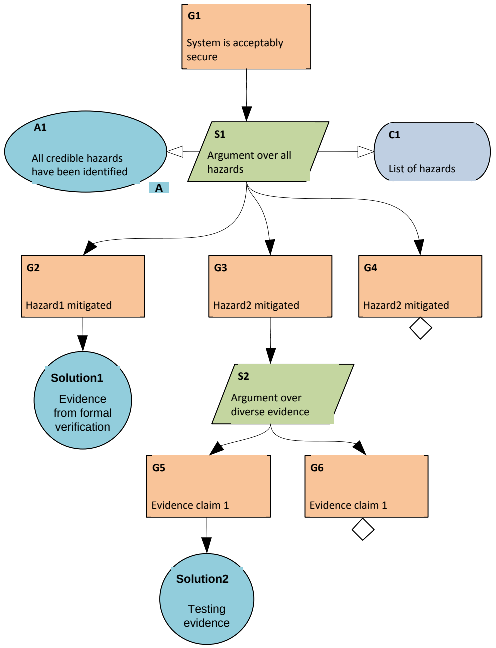


A reasoned and compelling argument, supported by a body of evidence, that a system, service or organization will operate as intended for a defined application in a defined environment

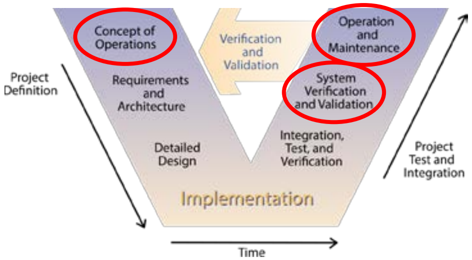
-GSN Standard V1

Goal Structuring Notation (GSN) is used in several tools

- *Goals*: claims about the system
- *Strategy*: approach to justifying the claim
- *Assumptions*
- *Solution*: leaf level evidence



Resolute



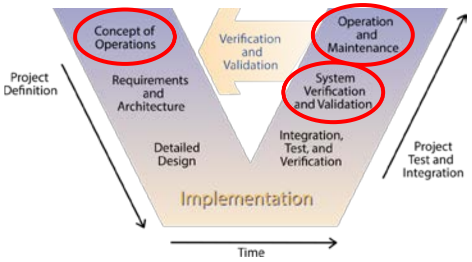
Resolute is a tool that can be used to specify claims and generate assurance cases for systems described in AADL.

Why is it better than the traditional approach?

- Assurance cases are captured in a domain specific language
- Claims are made in a logic designed for reasoning about the assurance case
- The claims are tied to the AADL model itself

```
memory_protected(p : process) <=
  ** "The memory of process " p " is protected from alterations by other processes" **
  property(p, SMACCM::OS) = "SeL4" or
  (property(p, SMACCM::OS) = "eChronos" and
   forall (mem : memory). bound(p, mem) =>
     forall (q : process). bound(q, mem) => memory_safe_process(q))
```

Resolute



The tool will generate an assurance case or give feedback on how the assurance case has been violated by the architecture.

- only_receive_gs(ML : SOFTWARE::Main_Loop.Impl)
  - 'MC : SOFTWARE::Motor_Control' only receives messages from the Ground Station
    - Only the Ground Station can send messages that pass Decrypt
    - The component 'MC : SOFTWARE::Motor_Control' only receives messages that pass Decrypt
      - The connection 'SN.motor_commands -> MC.motor_commands' only carries messages that pass Decrypt
        - The connection 'SN.motor_commands -> MC.motor_commands' delivers data without alteration
        - The component 'SN : SOFTWARE::Stability_Navigation' only receives messages that pass Decrypt
          - The connection 'CCT.mavlink_out -> SN.mavlink' only carries messages that pass Decrypt
          - The connection 'RC.commands_out -> SN.rc_commands' only carries messages that pass Decrypt
            - The connection 'RC.commands_out -> SN.rc_commands' delivers data without alteration
            - The component 'RC : SOFTWARE::Radio_Control' only receives messages that pass Decrypt

Kind 2 / JKind

Tools like SpeAR and AGREE utilize model checking to perform their analysis. Kind 2 is an open source SMT-based model checker developed at the University of Iowa:

- Bounded Model Checking
- K-induction
- Property Directed Reachability (PDR)
- Parallelized architecture

Rockwell Collins implemented a Java version (JKind) to improve:

- Deployability
- Integration into tool-chains

SMT Solvers

Satisfiability Modulo Theories (SMT) solvers are used in many tools, including the Kind tools. These tools enable reasoning about systems containing reals, integers, strings, arrays, bitvectors, …

Some solvers include:

- CVC4 (University of Iowa/NYU)
- Z3 (Microsoft)
- Yices2 (SRI)

In general these tools do not support nonlinearity which limits our ability to analyze complex algorithms such as nonlinear control systems.

Conclusions and Future Directions

SpeAR

- Add capability for user-defined patterns
- Improve the usability of the tool for novice users

Resolute

- Integration with other assurance case tools
- Support for other GSN types (only support claims)

AGREE

- Improve the contract specification language
- Real-time contracts

Questions