

RESEARCH REVIEW 2022

**Carnegie
Mellon
University**
Software
Engineering
Institute

Maturing Assurance Contracts in Model-Based Engineering

NOVEMBER 14–16, 2022

Dionisio de Niz
Technical Director, Assuring Cyber-Physical Systems

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

©2022

DoD Digital Engineering Challenges

Model-Analyze-Build

Late Discovery of Design Errors in DoD Systems is very costly.

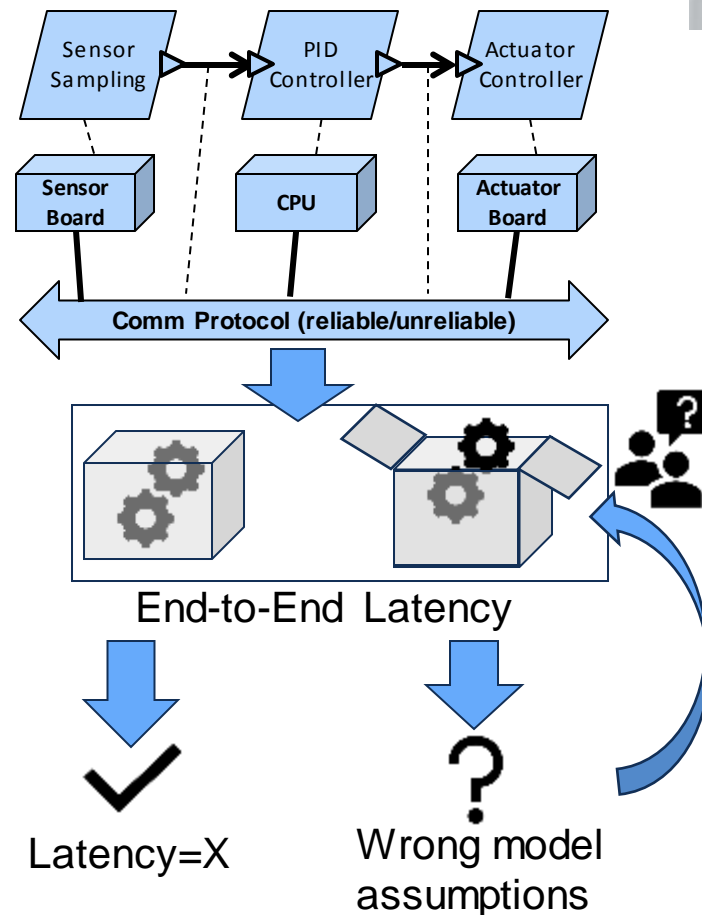
Architecture modeling and analysis can **detect** design **error early**

BUT:

Analysis assumptions are often implicit
if analysis **assumptions not met**: analyses break down for reasons not clear to users of analysis tools.

E.g., e2e Latency Assumption: periods multiple of each other (harmonic)

DoD barrier for adoption

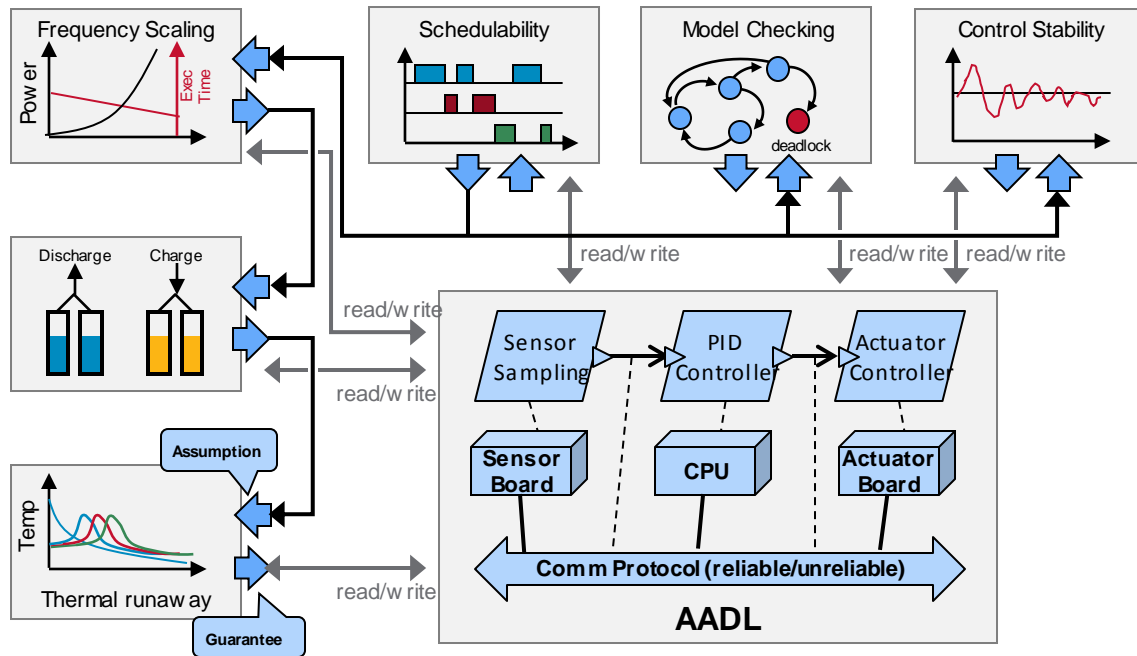


Digital Engineering: Multiple Claims - Multiple Analyses

Different Assurance Claims

- Combine multiple analysis
- Validate assumptions
- Resolve assumption conflicts

Integrate into arguments to satisfy claims



Analysis Contract: Tracking Assumptions and Guarantees

contract {

input:

E2ELatencies

assumptions:

areConnectionsDelayed()

areDeadlinesConstrained()

areTasksSchedulable()

areAllThreadsPeriodic()

analysis:

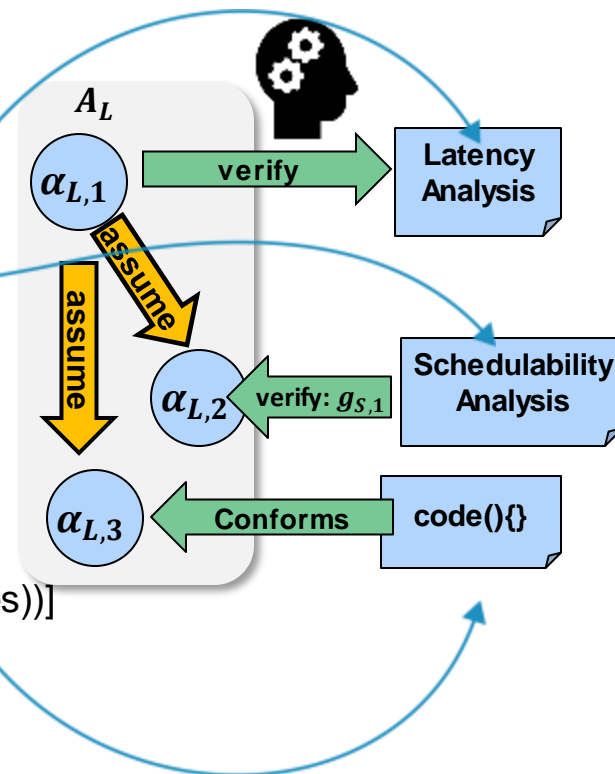
meetEndToEndLatencies()

guarantee:

[E2EResponses[i] <= E2ELatencies[i] for i in range(len(Responses))]

}

$$C_L = (A_L, G_L) \text{ with } A_L = \{\alpha_{L,1}, \alpha_{L,2}, \alpha_{L,3}\}$$



Shift Left And Down to the Implementation

Early Analysis

- Evaluate design decisions with partial information
- E.g., latency analysis before periods
 - periods of tasks must be multiples of each other

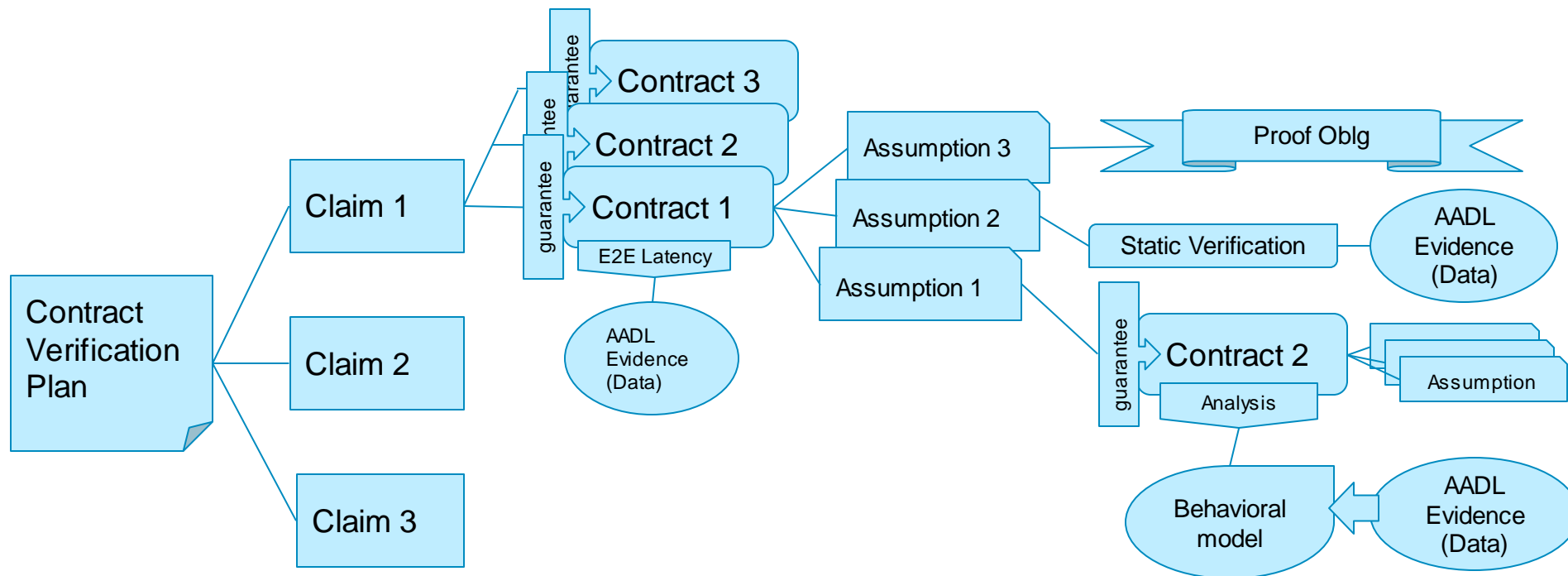
Refinement

- Track pending information
 - periods
- Track and execute pending verification
 - Schedulability

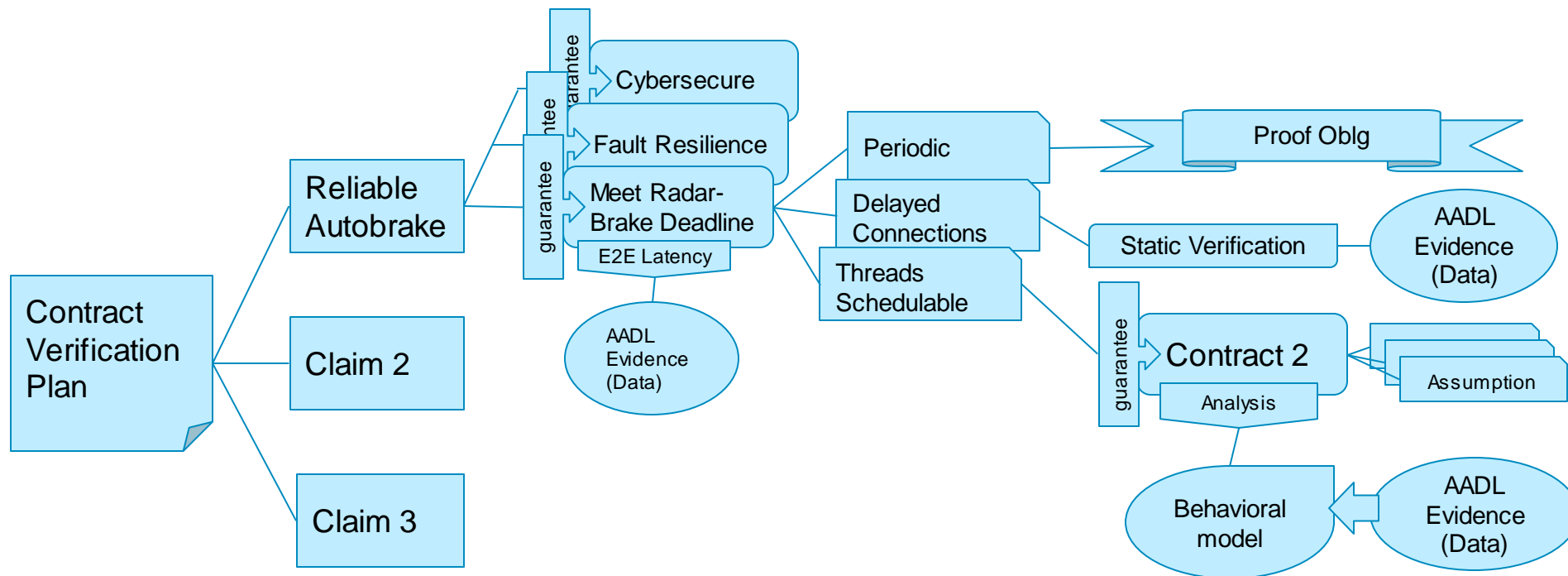
Conformance

- Track implementation assumption
- Verify implementation conformance
 - Task executed strictly periodic $\alpha_{L,3}$

Assurance Contract Argumentation



Assurance Contract Argumentation



Contract Argumentation in Development Lifecycle

Integrity of Analysis

- Verify assumptions
 - Detect violation
 - Suggest repairs
- Offer alternative analysis that satisfy assumptions

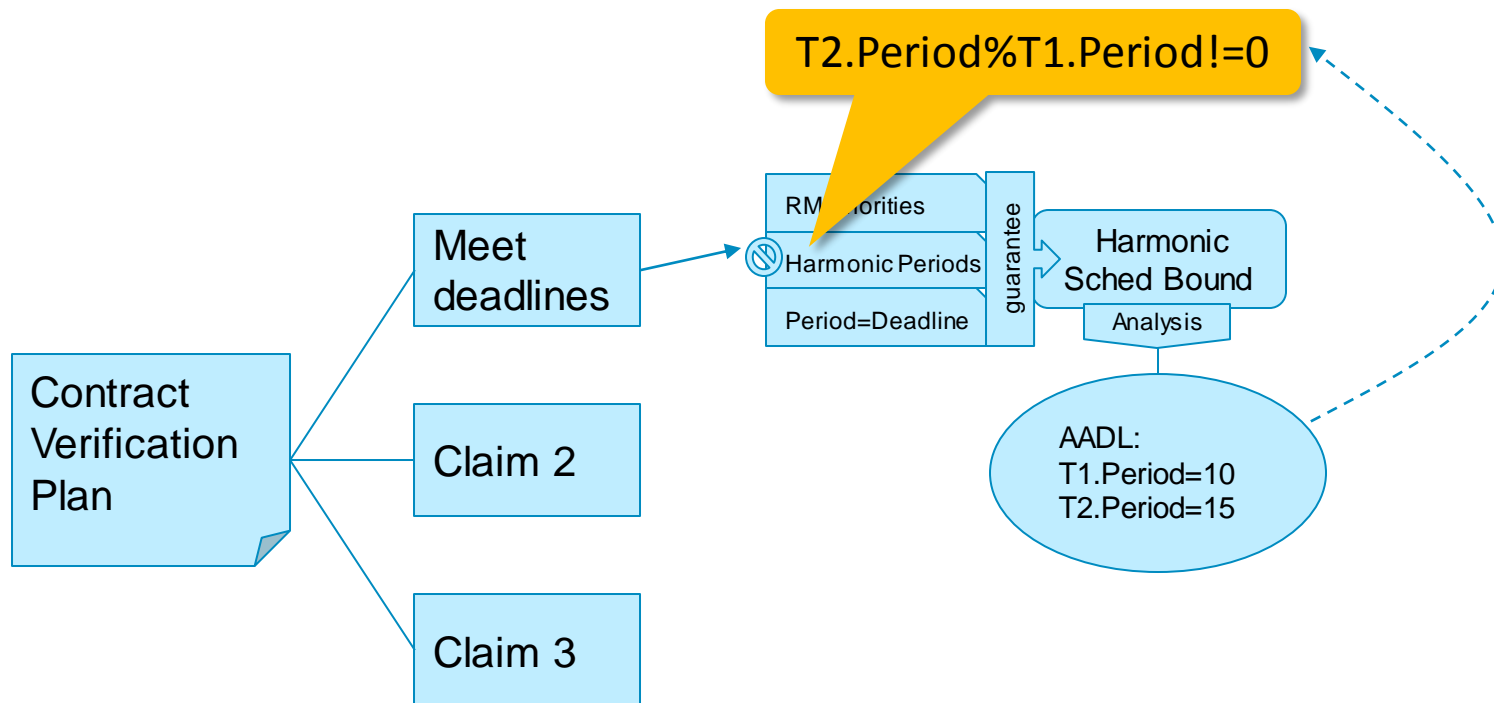
Refine Design

- When enough new data for new analysis
- When new data affects proof obligations

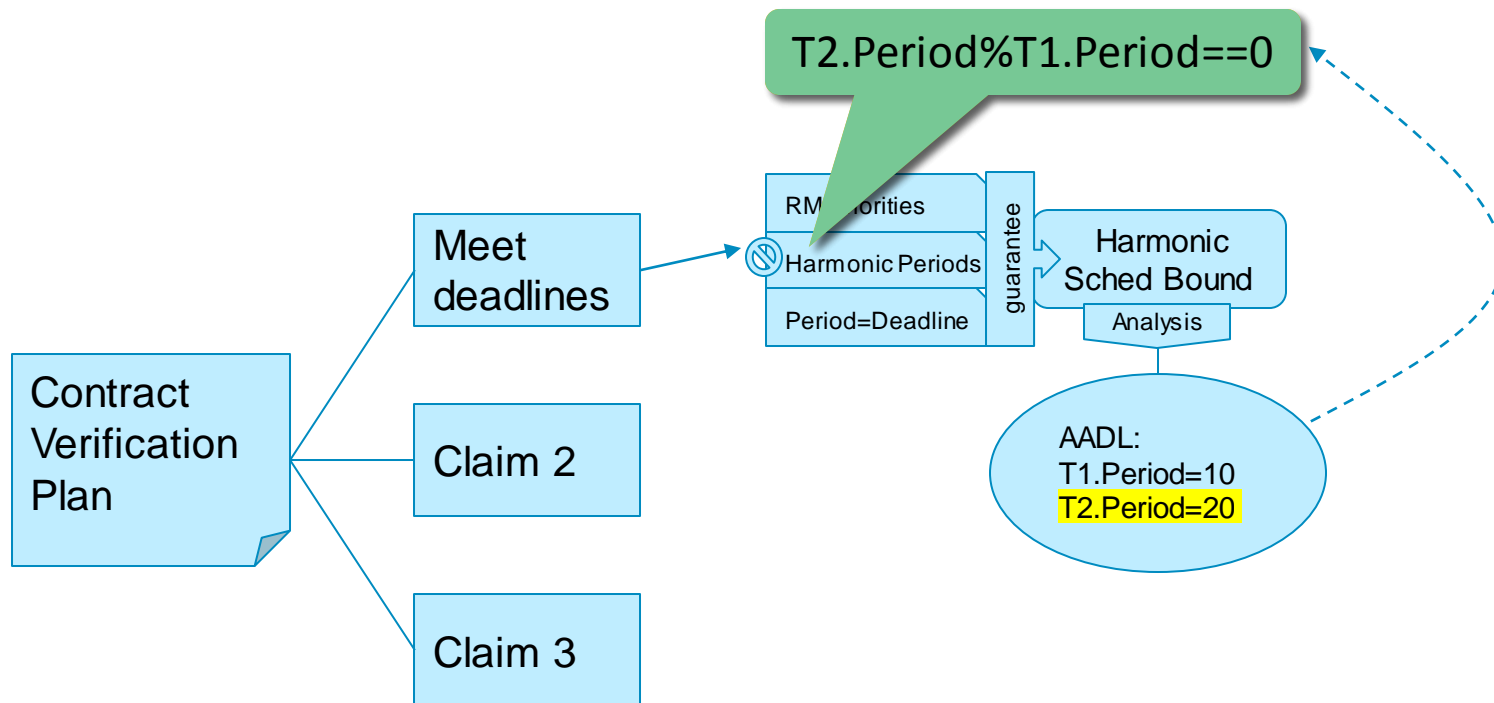
Argument reusability

- Self-contained modular analysis contracts

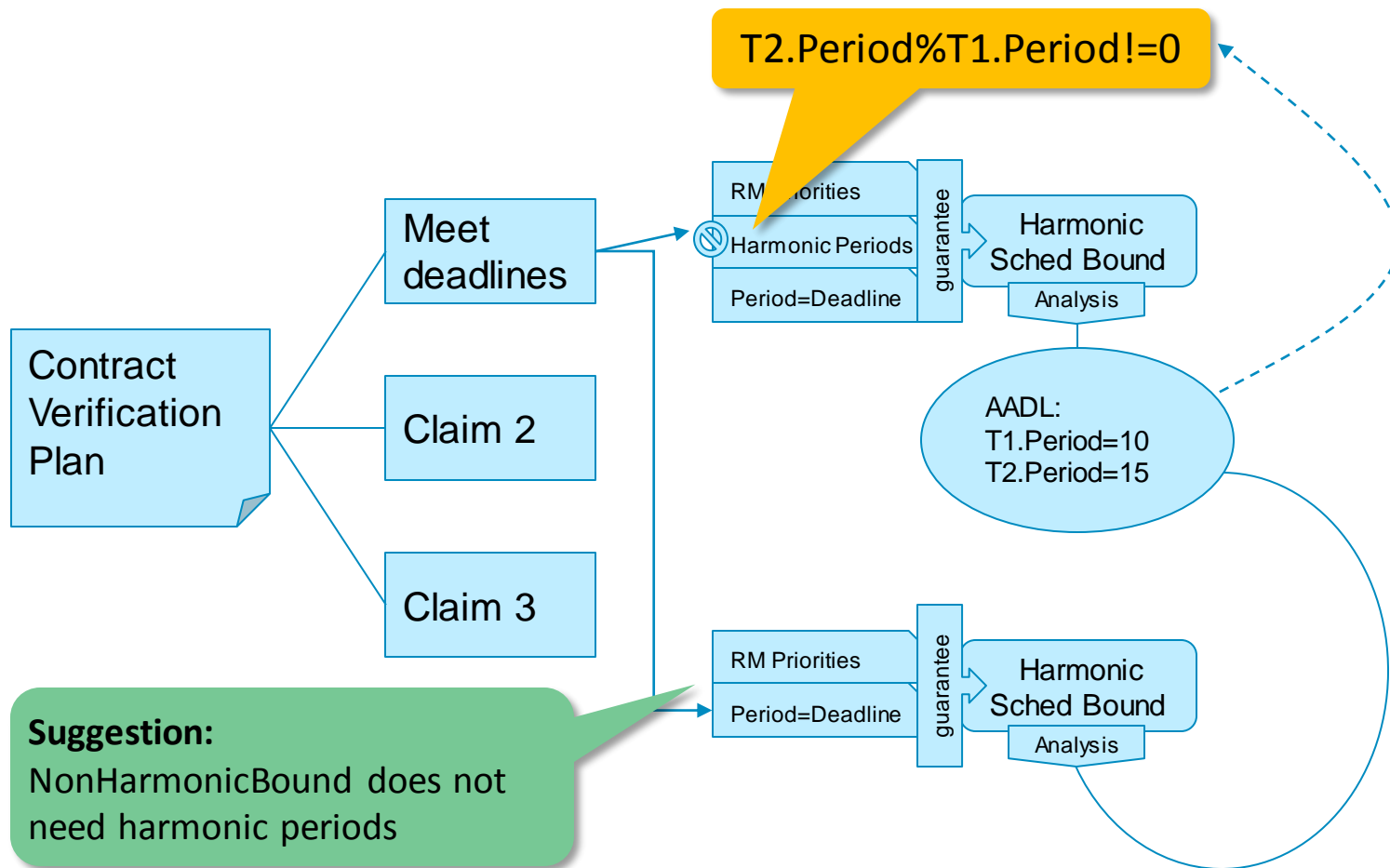
Integrity of Analysis: Repairing Assumptions



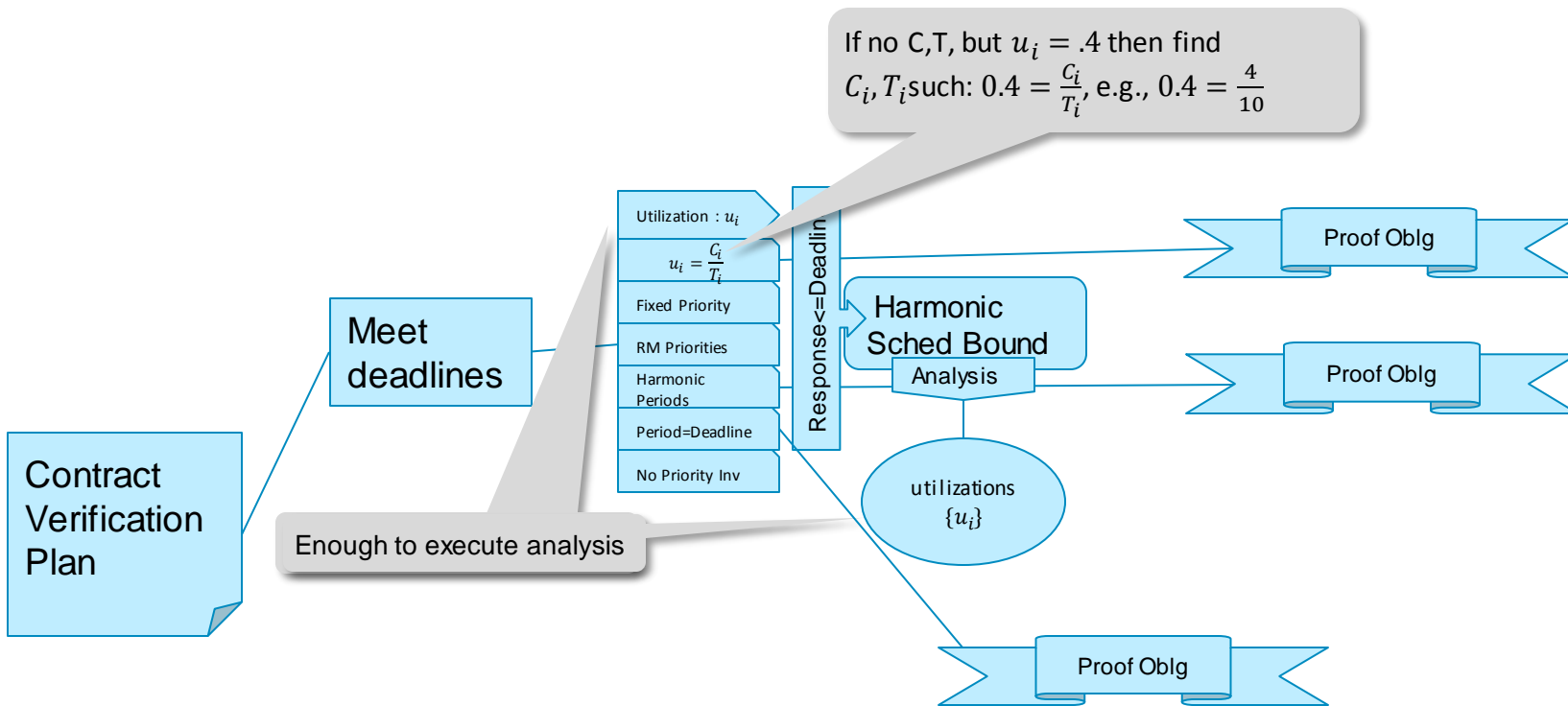
Integrity of Analysis: Repairing Assumptions



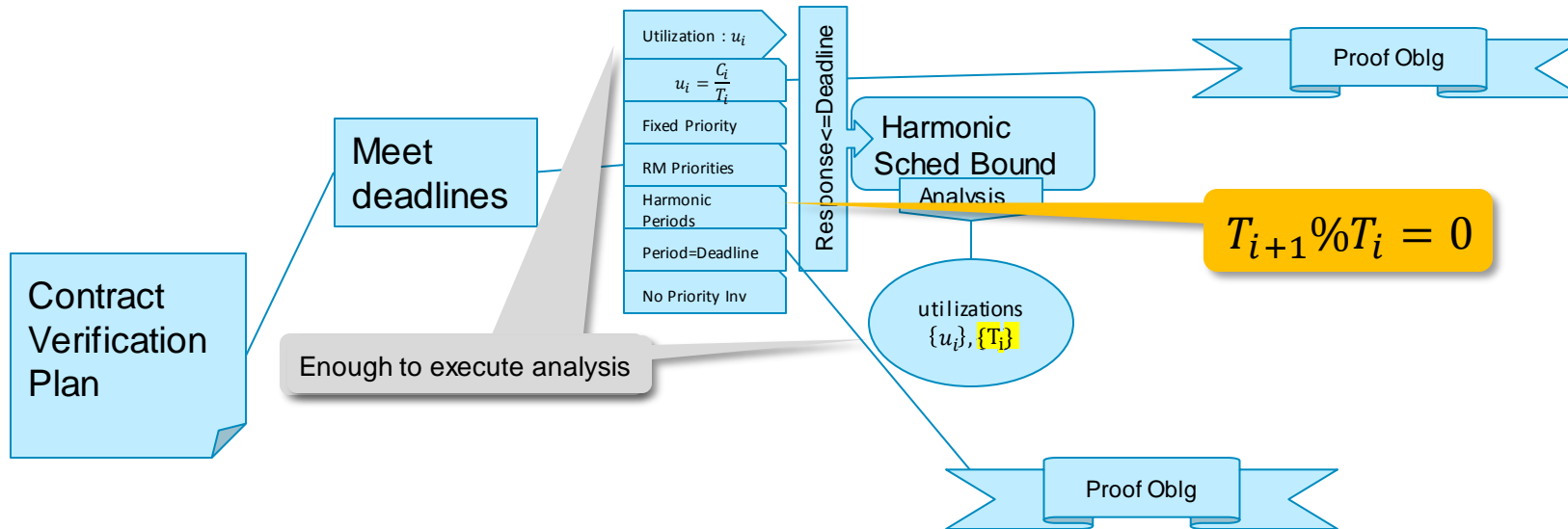
Integrity of Analysis: Repairing Assumptions



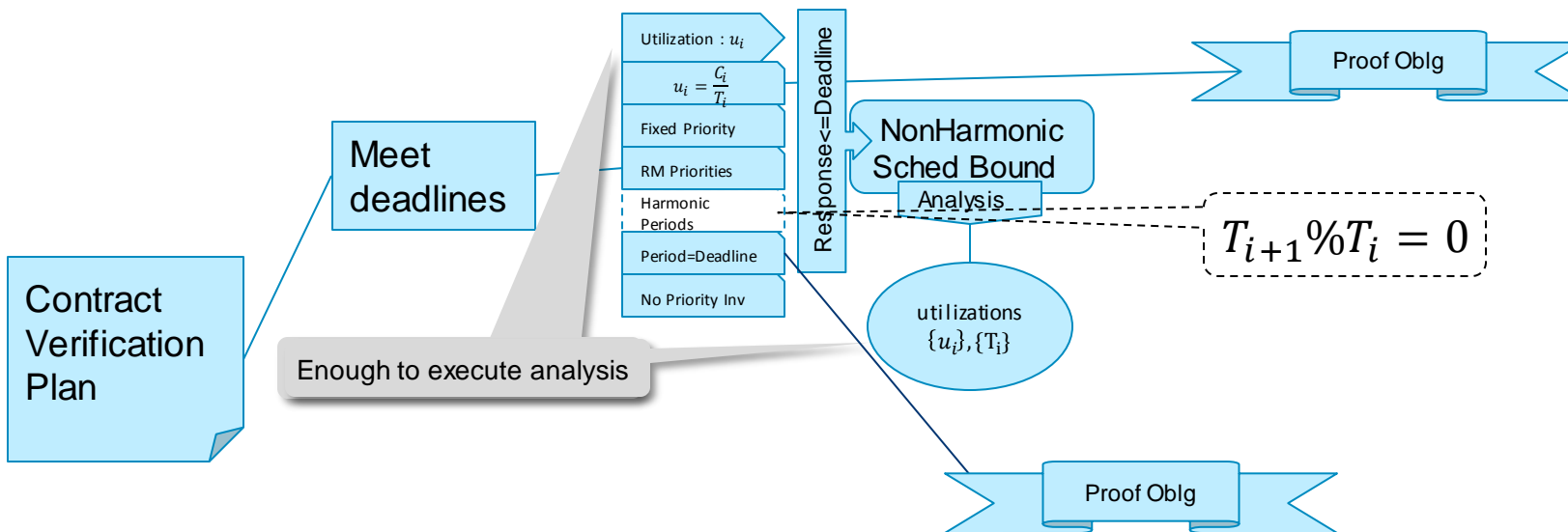
Refinement throughout development (1)



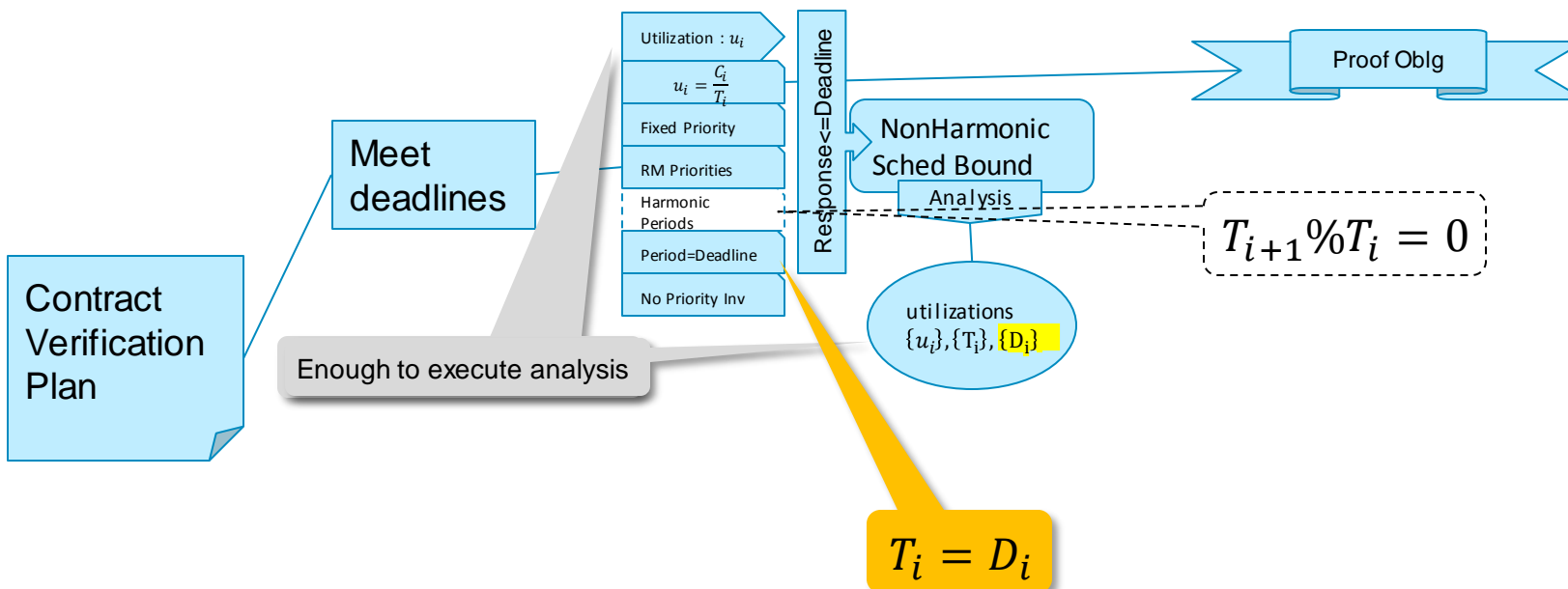
Refinement throughout development (2)



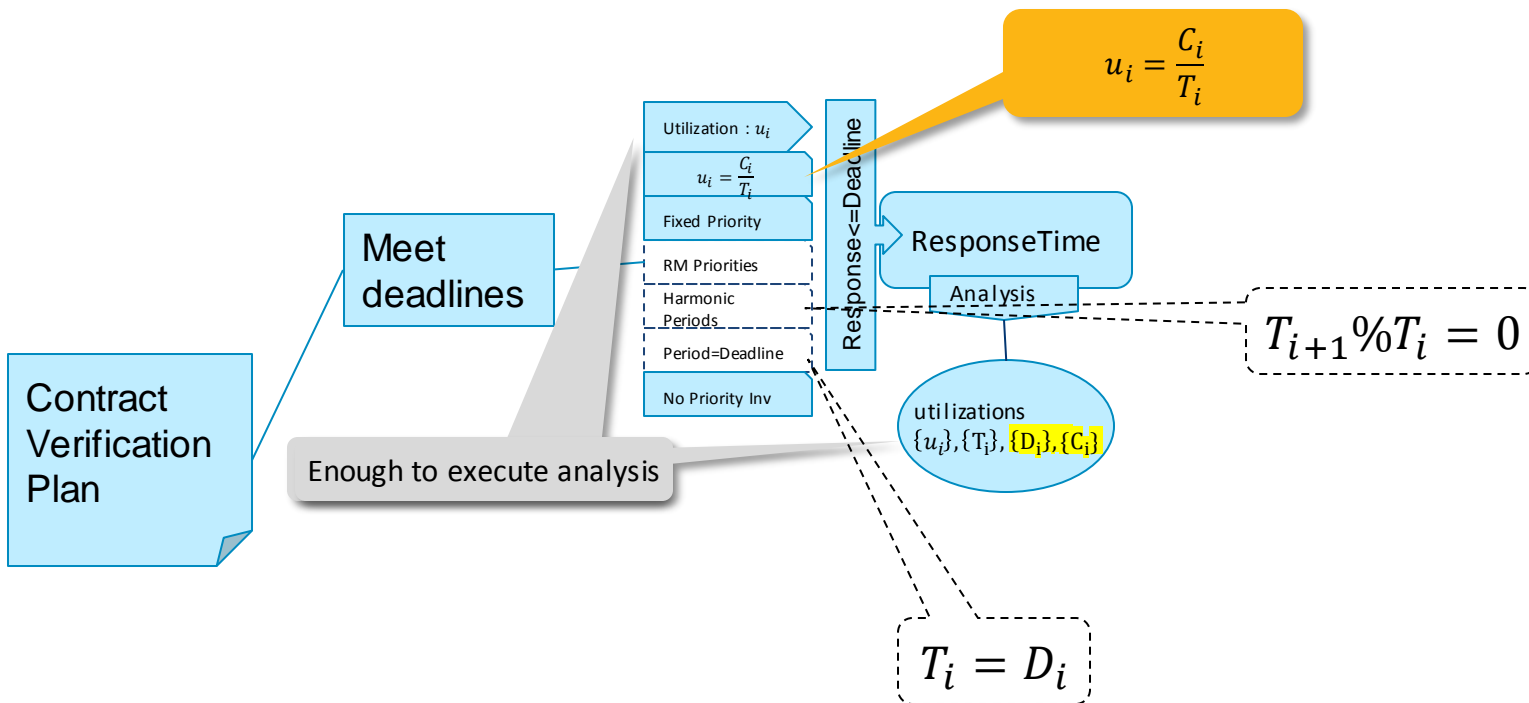
Refinement throughout development (3)



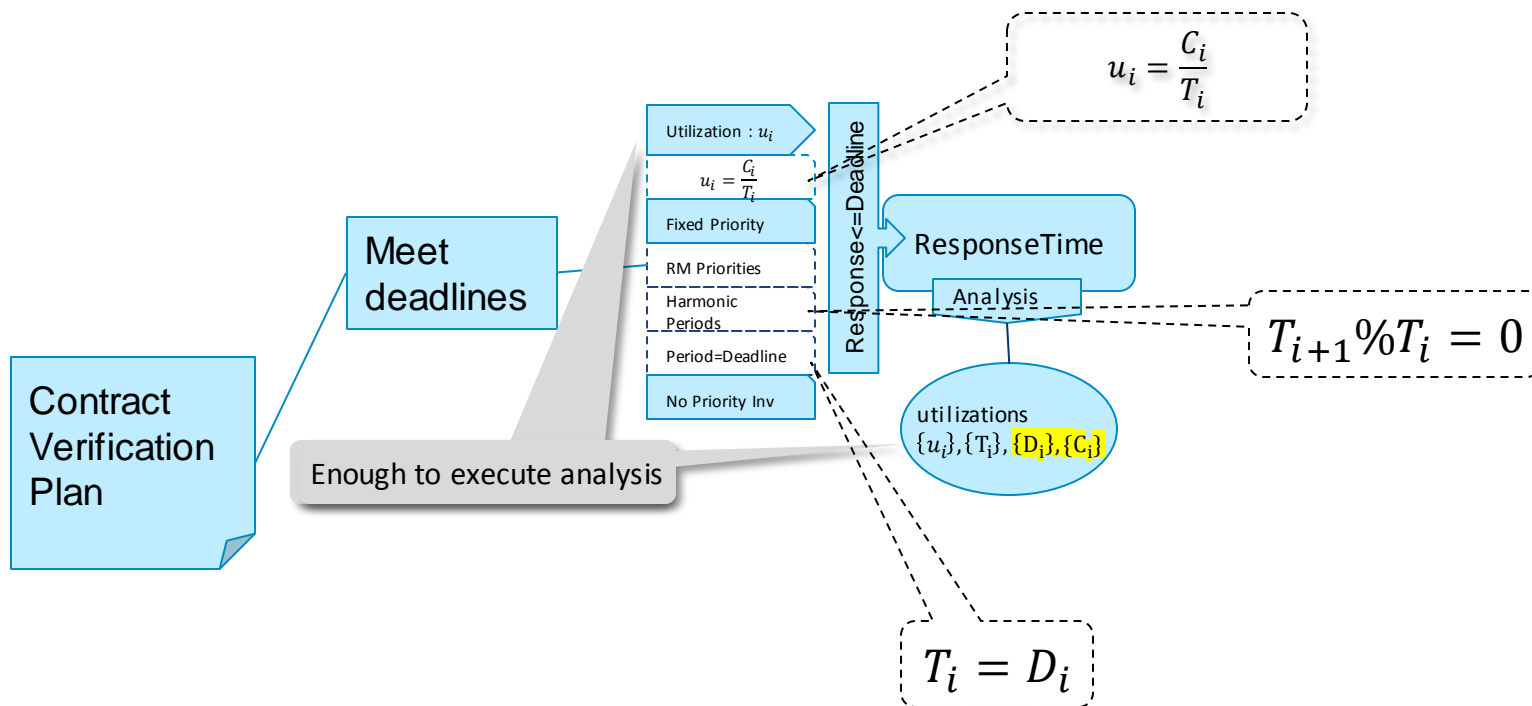
Refinement throughout development (4)



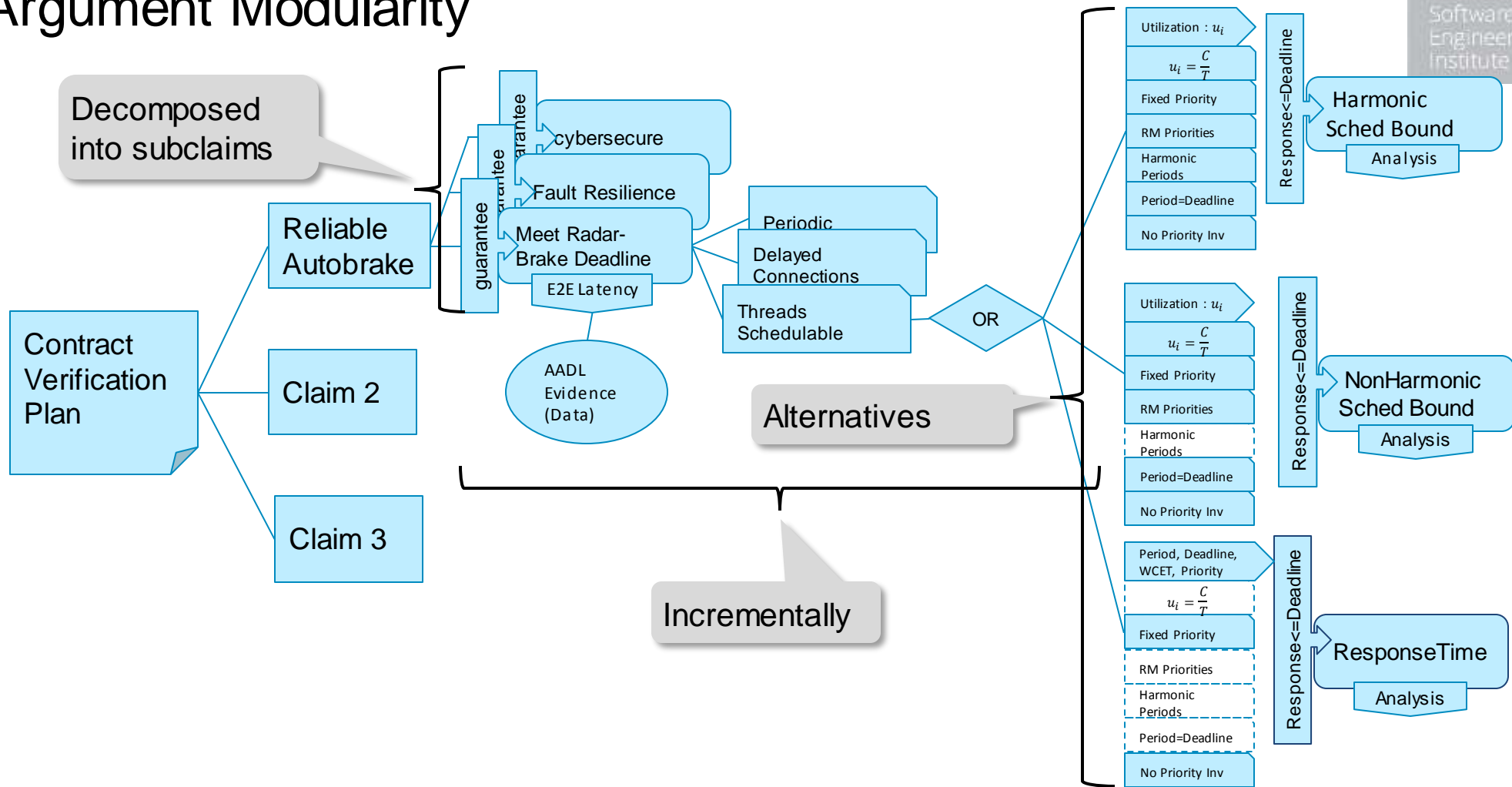
Refinement throughout development (5)



Refinement throughout development (5)



Argument Modularity



Implementation: Symbolic Contract Argumentation

Assumptions

- Constraints that must be satisfied for a valid analysis

Analysis

- Evaluate whether the guarantee can be discharged

Guarantee

- Assertion presented as a true fact on model

Implementation

- Constraint Satisfaction Solver (Satisfiability Modulo Theories – Z3)
- Implements contract argumentation
 - Evaluate whether constraints can be satisfied with facts from analysis guarantees
- Validate assumptions
 - Proof obligations: lack of constraints allow any value that satisfy assumption (e.g., RM priorities)

Artifacts

- Annex language hosted in AADL/OSATE
 - Model Query Language adaptable for multiple modeling language (e.g., SysML V2)
- Automatic Execution of Contract Verification Plan
 - Assumption repair & analyses alternatives

Contract Argumentation Scalability

Exploit Knowledge from Scientific Domain

- Efficient algorithms from specialized domains
 - E.g., greedy worst-case response time in real-time theory
 - Implemented in imperative languages

Assume correctness of analysis

- When validating the contract argumentation
- Enables connection with other lower-level verification results
 - E.g., PROSA: coq (theorem prover) verification of real-time theory

Correctness of implementation

- Exploit proven properties of runtime mechanisms: e.g., schedulers, hypervisors
- Exploit code generation
- Deferred code verification to conform to assumptions

Impact

Certification

- Automated and sound verification of assurance claims through models

Fielding Speed

- Automated assurance watchdog
 - Validate incremental refinement through design
- Concurrent formal assurance argument construction and system development

Digital Engineering and AADL Ecosystem

- Incremental sound analysis infrastructure to DoD modeling efforts
- Architecture-Centric Virtual Integration Practice (ACVIP) within FVL
- DARPA programs using AADL

Concluding Remarks

Certification in Digital Engineering Era

- Follow model-analyze-build
 - Automated argumentation supported by model-based analysis contracts

Shift Left

- Start verifying early design decisions
- As design is refined
 - Ensure properties of previous design decisions are preserved
 - New refinements can provide additional evidence / properties to support assurance

Down To The Implementation

- Drive properties & assumptions down to the implementation

Scalable

- Exploit efficient analysis from different domains

Sound

- Exploit advances in formal verification
 - Combination of analysis, verification of assumptions, implementation compliance, analysis correctness

Team



Bjorn Andersson
Principal Researcher



Jerome Hugues
Senior Researcher



Sam Procter
Senior Researcher.



John Hudak
Principal Engineer



Sholom Cohen
Principal Engineer



Lutz Wrage
Senior Researcher



Aaron Greenhouse
Senior Researcher



Ruben Martins
Research Professor CMU/CS



Gabriel Moreno
Principal Researcher



Joseph Seibel
Software Engineer



Dionisio de Niz
Technical Director ACPS.

Email us at: info@sei.cmu.edu

To learn how the results of our project can help improve the effectiveness of your systems' assurance.

Document Markings

Copyright 2022 Carnegie Mellon University.

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.

DM22-0869