AMERICAN INSTITUTE OF
AERONAUTICS AND ASTRONAUTICS

Formal Verification in Integrated Avionics Systems

William T. Doan, Ashraful Islam, Viknesh Venkatachalam, and Mahd Malik
The University of Texas at Dallas

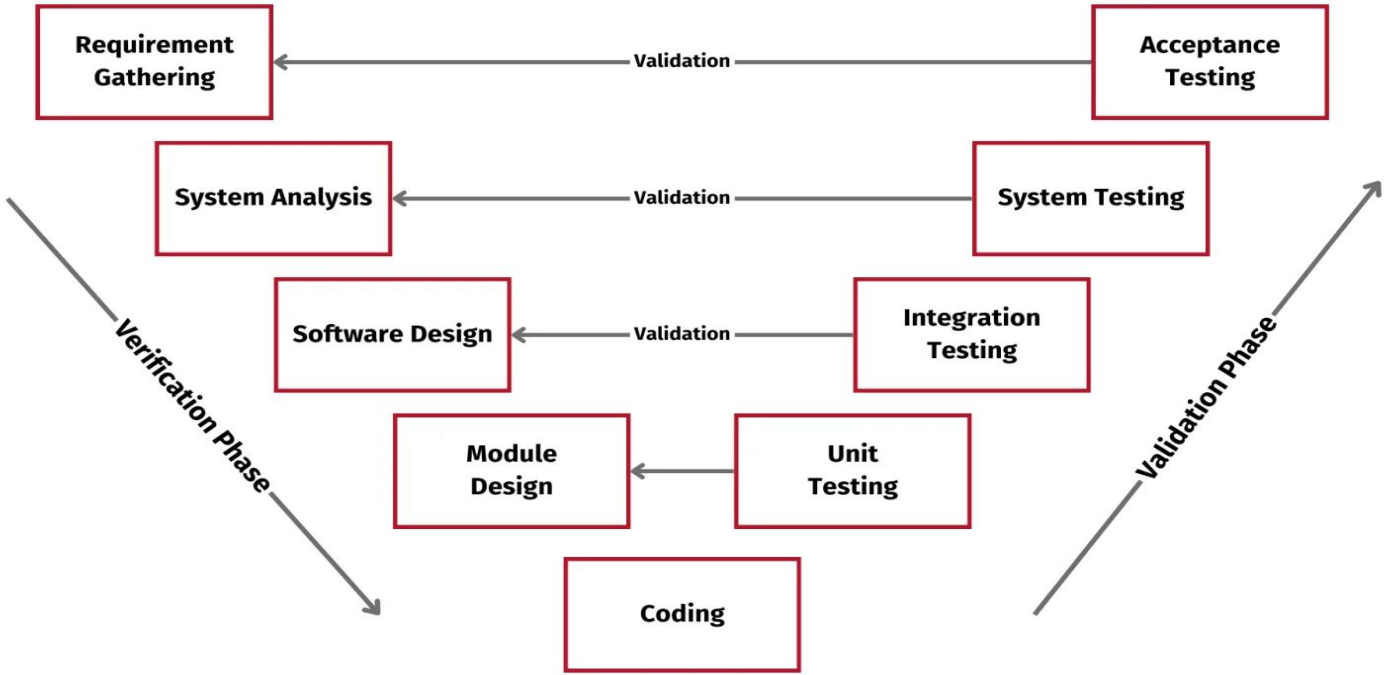
AIAA Region IV Student Conference, 27 March 2026 - 29 March 2026

The Evolution of Avionics Architecture

- Modern flight decks are highly integrated software environments, not discrete gauges.
- Advanced and automated navigation systems are a **safety-critical domain** dependent on computational correctness!
- Catastrophic failures largely originate from **errors in the early stages** of the Systems Development Lifecycle (SDLC)

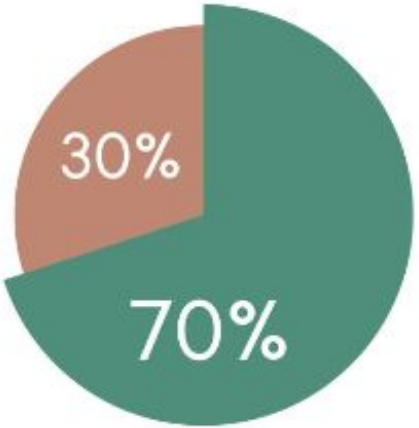
The V-Model Vulnerability

- The aerospace industry traditionally follows the **V-Model** for development.



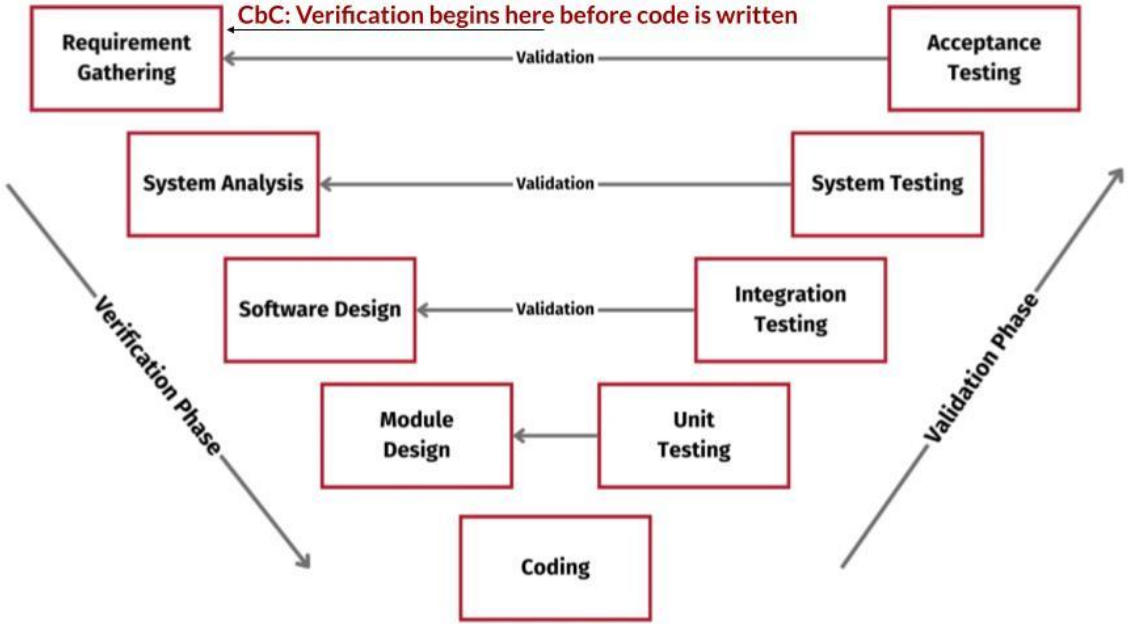
The Economic Burden of V&V

- To meet strict certification standards like **RTCA DO-178C**, developers rely on exhaustive post-implementation testing.
- Dynamic testing only shows some errors, but **never** their absolute absence.
- This reliance on testing to catch upstream errors results in **V&V** activities taking up to **70% of total development.**



Proposed Solution: Correctness by Construction

- We propose a "**Correctness by Construction**" (**CbC**) framework, shifting verification to the left side of the V-Model using **s(CASP)**'s formal specification.



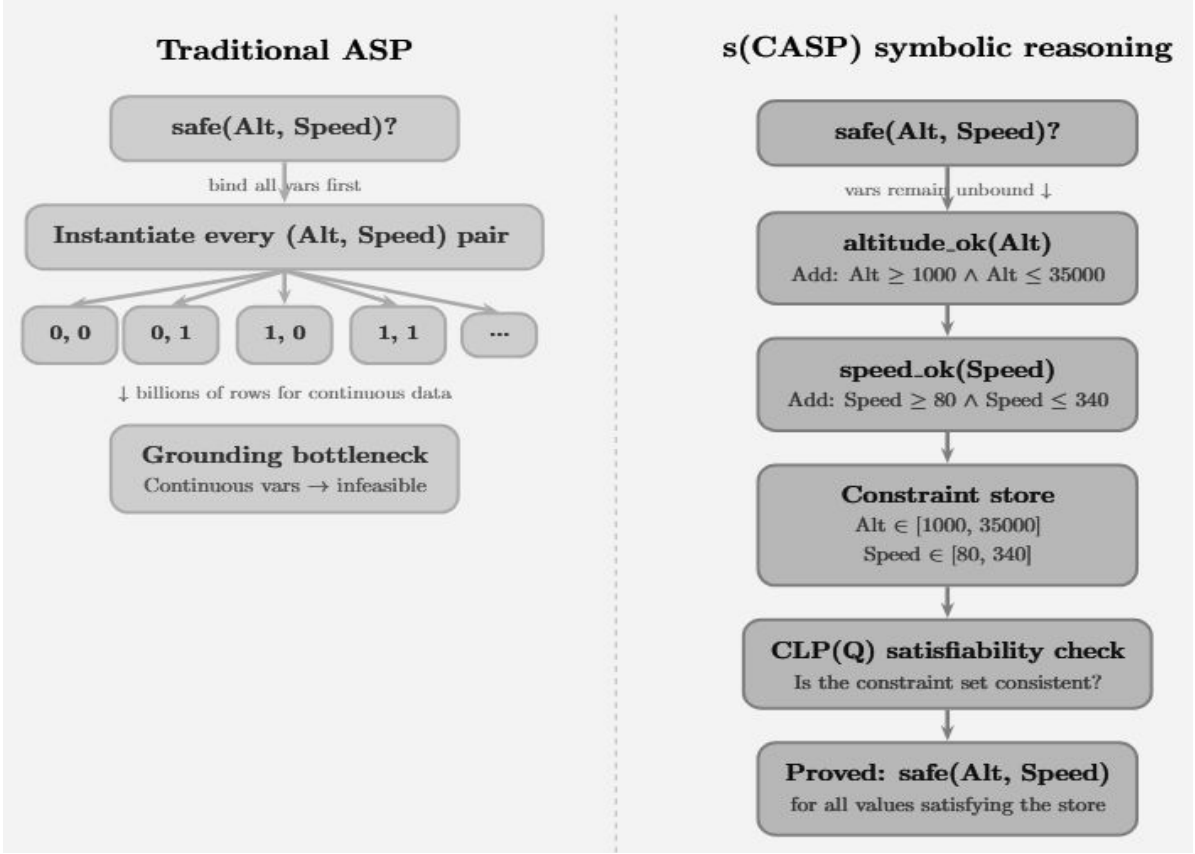
Overcoming the Grounding Bottleneck with s(CASP)

- **The grounding bottleneck**:
  - Traditional formal methods struggle with avionics scale due to the "**grounding bottleneck**", where all variables must be **instantiated before solving**.
  - This is impossible for continuous data like altitude or airspeed.

Overcoming the Grounding Bottleneck with s(CASP)

- Our framework utilizes **s(CASP)**, a goal-directed **Answer Set Programming** system that **avoids grounding** entirely.
- Using **top-down, query-driven execution** it keeps variables symbolic and accumulates constraints, mathematically bounding conditions where safety requirements might fail across the entire continuous flight data.

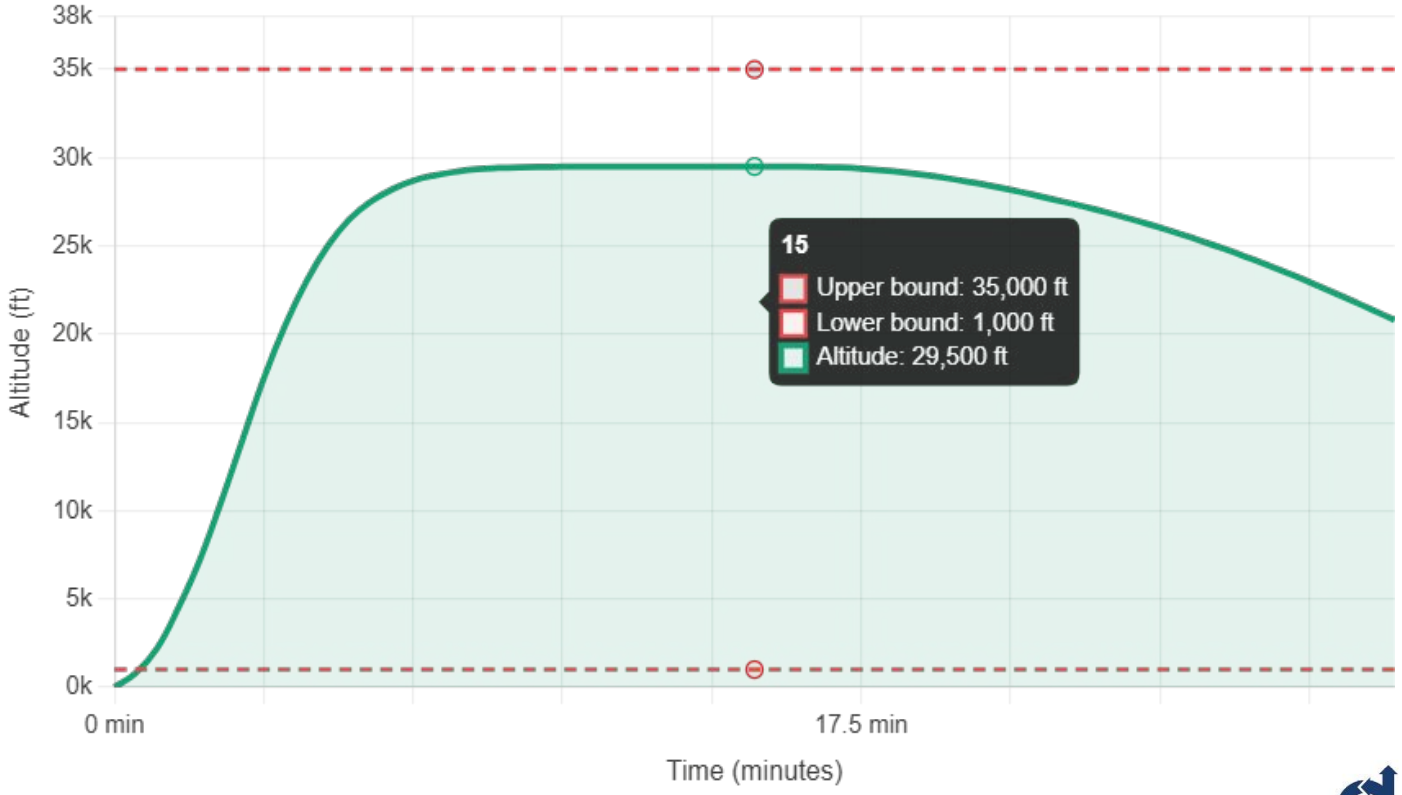
ASP vs s(CASP) Diagram



Continuous Verification via CLP(Q)

- Avionics systems interact with the physical world over **continuous** time and space.
- Our framework integrates Constraint Logic Programming over rational numbers (**CLP(Q)**).
  - **CLP(Q)** represents rational numbers exactly, not floating-point approximations.
  - Formally verifies requirements over complex flight equations and inequalities.
  - Ensures operational integrity across an infinite state space.

Sample Flight Altitude Profile



Automating the Traceability Matrix (RTM)

- Maintaining a Requirements Traceability Matrix is traditionally a manual, error-prone process.
- Our framework automates this by utilizing **s(CASP)** justification trees.
  - When a property is verified, the system generates a **structural logical path** for the conclusion.
  - By mapping symbolic logic to human-readable text via the **#pred** directive, it produces a mathematically proven, human-readable **RTM** that satisfies certification authorities.

Automated RTM via s(CASP) Justification Trees

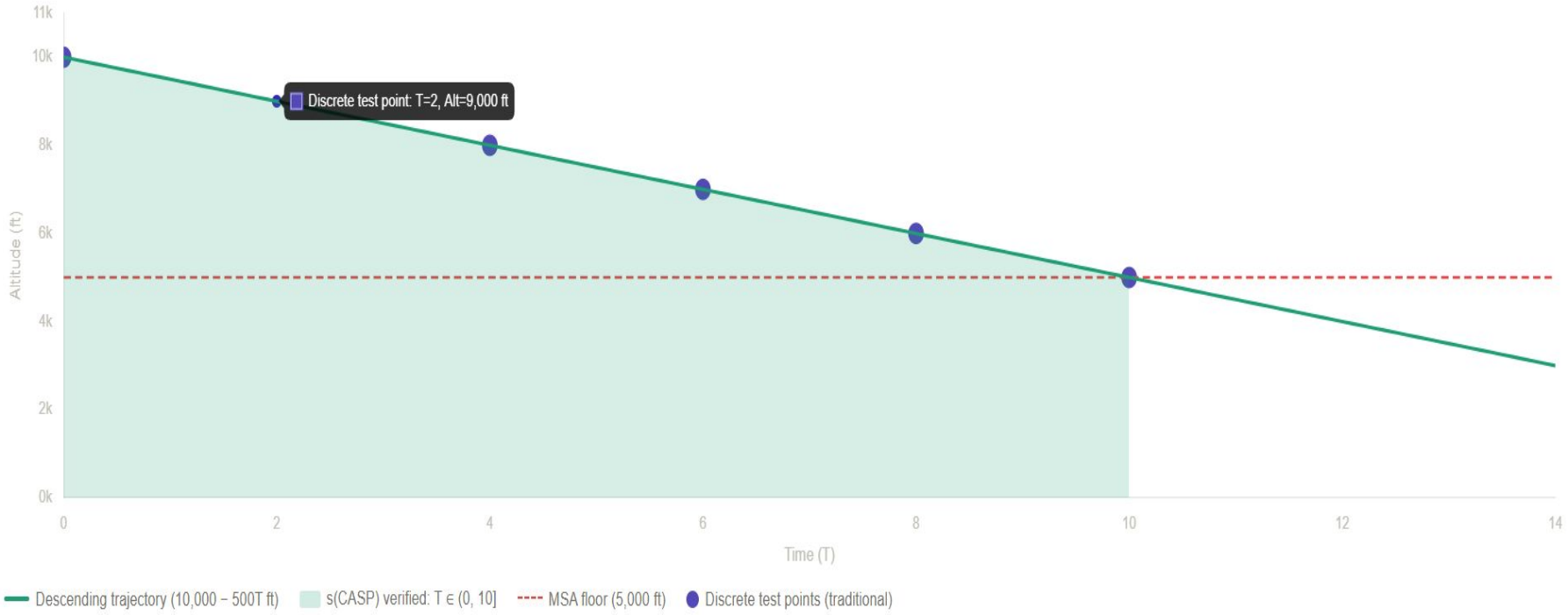
Requirement Human-readable	s(CASP) query Symbolic logic	Justification tree Logical path	#pred output RTM entry
Altitude bounds Alt ∈ [1000, 35000 ft]	`safe(Alt,_)` `altitude_ok(Alt)`	safe ← alt_ok ← constraint held	Verified Altitude is safe
Airspeed limits Speed ∈ [80, 340 kn]	`safe(_,Speed)` `speed_ok(Speed)`	safe ← spd_ok ← constraint held	Verified Speed is safe
Trajectory safety Continuous path valid	`safe_path(T)` `clp_check(T)`	path ← clp_ok ← bounds satisfied	Verified Path is safe
IMA partitioning Resource isolation	`isolated(A,B)` `no_conflict(A,B)`	isolated ← no_conf ← partition held	Verified Apps isolated

RTM auto-generated via #pred directive — symbolic logic mapped to human-readable certification evidence

Case Study: Trajectory Safety Verification

- In a traditional V-Model, engineers test discrete points (altitudes at specific times), leaving space between points unverified.
- We applied the framework to the Minimum Safe Altitude (**MSA**) monitoring function.
  - Using **s(CASP)** and **CLP(Q)**, we queried the system to find the exact mathematical bounds of the safe operation.
  - The result proved the trajectory safe for any continuous time within a specific range.
  - Eliminates the risk of latent errors between discrete test cases.

MSA Flight Trajectory Graph



Automated RTM with MSA Trajectory

DO-178C Requirement	s(CASP) predicate #pred directive	Justification tree Logical path to conclusion	Auto-generated Human-readable RTM entry

Trajectory safety
MSA monitoring function must be safe for all T in operating range
	`safe_trajectory (T1, T2)`
	Query issued:
	`?- safe_trajectory (0, T).`
		safe_trajectory(0,T)
		← altitude_at(T, A)
		← T ≥ 0
		← A = 10000−500·T
		← terrain_clearance(A)
			The flight trajectory is safe between time 0 and T, with T > 0 and T ≤ 10
			Model: T ∈ (0, 10]

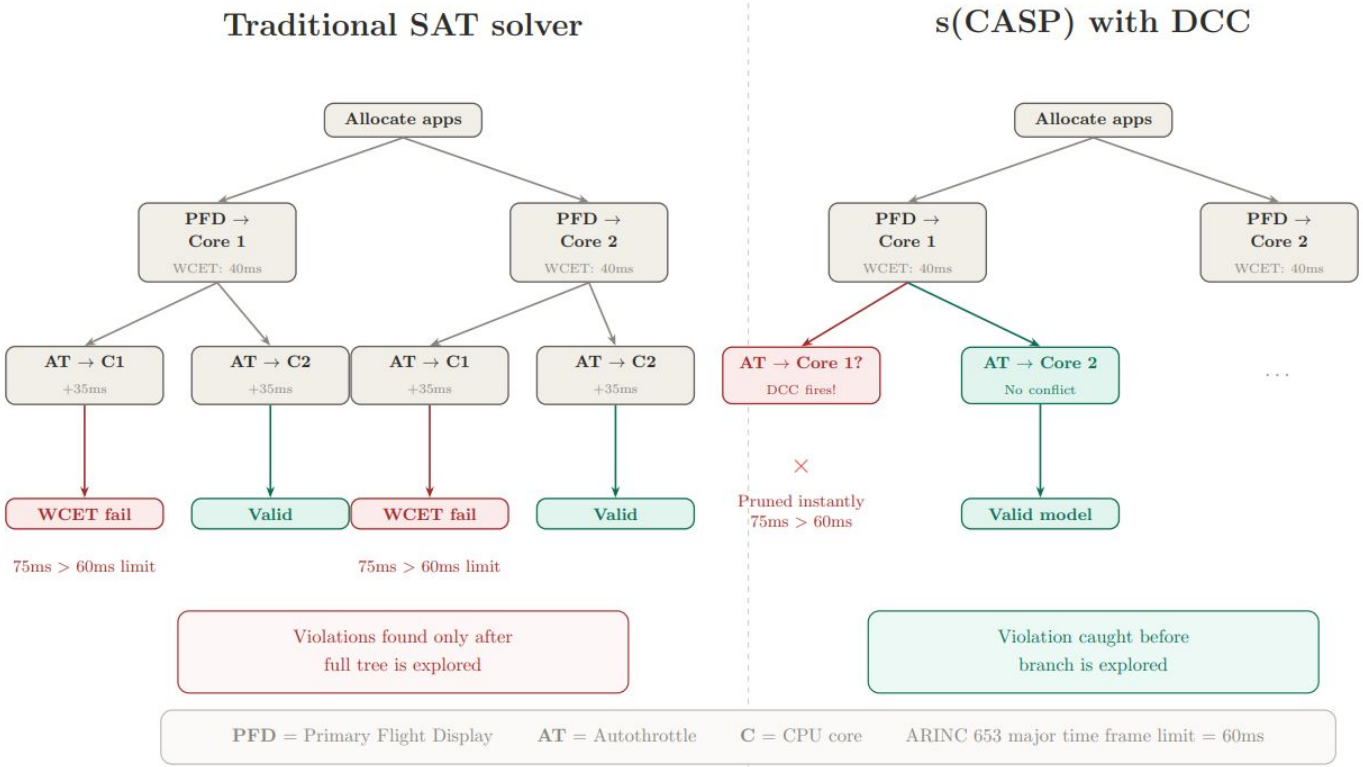
Altitude model Aircraft descends at 500 ft/unit from 10,000 ft initial alt.	`altitude_at(T, A)` T #>= 0 Initial_A #= 10000 A #= 10000−500·T	altitude_at(T, A) ← T ≥ 0 [CLP(Q)] ← A = 10000−500·T ← constraint satisfied	The aircraft altitude at time T is A, where T ≥ 0 and A = 10000 − 500·T
Terrain clearance Altitude must stay at or above MSA of 5,000 ft	`terrain_clearance(A)` MinSafeAlt #= 5000 A #>= MinSafeAlt	terrain_clearance(A) ← A ≥ 5000 [CLP(Q)] ← global constraint holds	The altitude A maintains safe terrain clearance, because A ≥ 5000

Generated by: s(CASP) --tree --human flags | #pred directives map symbolic logic to DO-178C certification evidence
Result: safe_trajectory(0, T) proved for all T ∈ (0, 10]

Scalability in Integrated Modular Avionics (IMA)

- In modern **IMA** environments, multiple applications share resources, causing combinatorial complexity and "*state-space explosion*" during verification.
- Our framework addresses this through Dynamic Consistency Checking (**DCC**).
  - **DCC** interleaves model generation with consistency checking, instantly **pruning** execution branches that violate global safety constraints (such as CPU overruns).
  - It allows formal verification of complex, multi-partition architectures without timeouts, unlike traditional solvers.

DCC: Full Tree Exploration vs. Early Pruning



Conclusion and Future Work

- The complexity of modern avionics has pushed the traditional model to its economic and technical **limits**.
- A "*Correctness by Construction*" model is now **necessary** to ensure safety.
- Our framework automates traceability, prunes complex search spaces, and verifies continuous flight logic.
- **Future work** involves integrating this into Model-Based Systems Engineering (MBSE) toolchains and extending the methodology to autonomous UAVs and Advanced Air Mobility platforms.

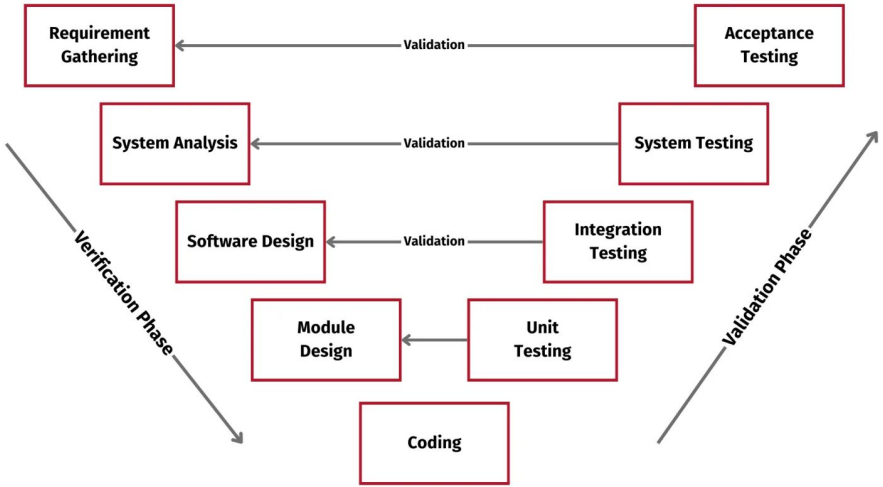
AEROSCASP

Questions?

The following slides were not included in the final presentation.

The V-Model Vulnerability

- The aerospace industry traditionally follows the **V-Model** for development.
- **A major flaw** is ambiguities and logical inconsistencies from Requirements Analysis often remain through design and implementation.
- **Defects** are typically only discovered in **late-stage** Verification and Validation (**V&V**).
- The cost of remediation increases exponentially as the project progresses, creating a "*latent error*" trap.

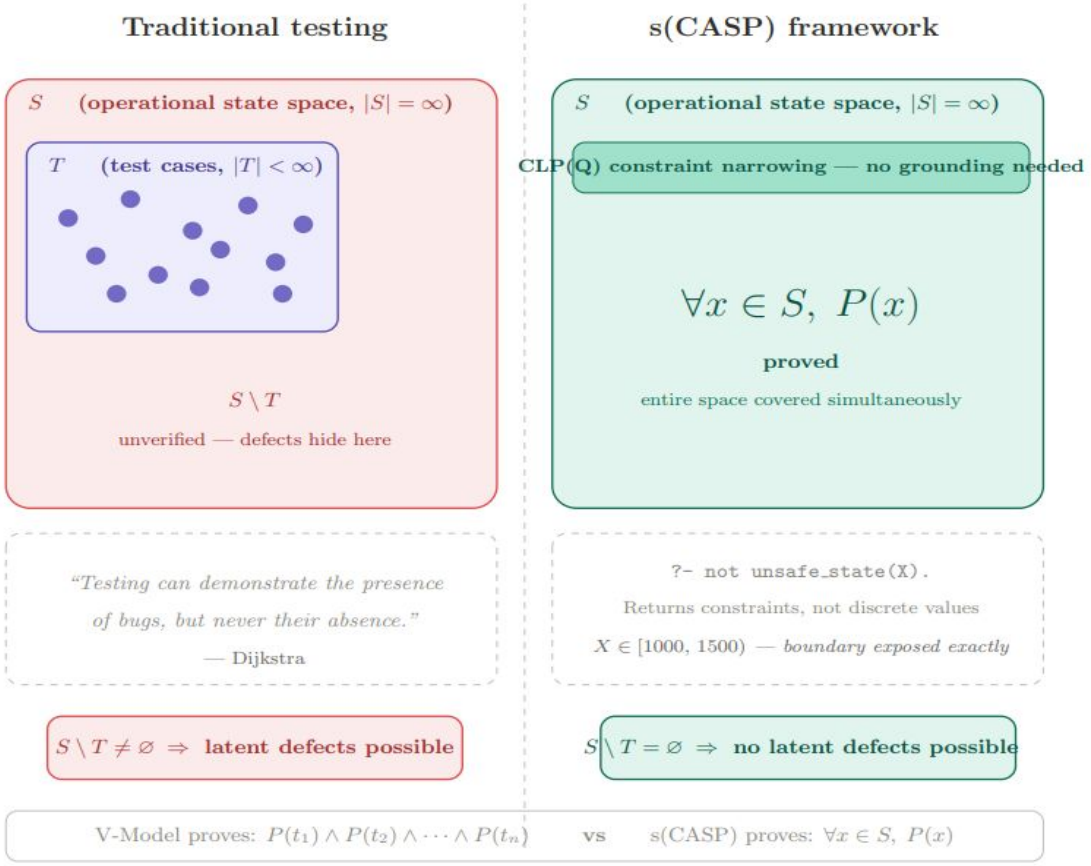


Epistemological Limits vs. Mathematical Certainty

- Traditional testing is limited because the operational state space of an aircraft is infinite, while the number of test cases is always finite.
- Testing can only verify a subset of states, leaving an infinite set of states unverified.

- Our **s(CASP)** framework uses universal quantification to evaluate continuous intervals.
  - Shifting from sampling points to proving properties over the entire state space provides a structural guarantee of safety that traditional testing cannot match.

Testing Methodologies



Proposed Solution: Correctness by Construction

- We propose a "*Correctness by Construction*" (**CbC**) framework.
  - Shift verification process to "*left side*" of the V-Model.
  - **s(CASP)**'s Formal specification establishes a rigorous mathematical groundwork for system properties before code is written.
  - Move burden of proof from post-facto testing to a priori mathematical verification.
  - Provable guarantee that software is semantically consistent with safety requirements.

Limitations and Current Scope

s(CASP) predicates are currently written manually by a verification engineer with formal logic expertise. The translation from natural language requirements is not yet automated." This framework is a proof-of-concept. The case studies use synthetic data, not live flight software from a certified avionics program. Formalizing continuous flight dynamics in CLP(Q) requires domain expertise in both aerospace engineering and constraint logic programming, which limits near-term accessibility.

The Evolution of Avionics Architecture

- Modern flight decks have transitioned from federated, discrete gauges to highly integrated software environments.
- Advanced and automated navigation systems create a **safety-critical domain** where operational integrity depends on computational correctness.
- Catastrophic failures largely originate from **errors in the early stages** of the Systems Development Lifecycle (SDLC), rarely by low-level hardware faults.

The V-Model Vulnerability

- The aerospace industry traditionally follows the V-Model for development.
- **A major flaw** is ambiguities and logical inconsistencies from the Requirements Analysis phase (**Phase Two**) often remain latent through design and implementation.
- These defects are typically only discovered during the **late-stage** Verification and Validation (**V&V**) phases.
- The cost of remediation increases exponentially as the project progresses, creating a "*latent error*" trap.

The Economic Burden of V&V

- To meet strict certification standards like **RTCA DO-178C**, developers rely on exhaustive post-implementation testing.
- Dynamic testing can only demonstrate the presence of errors, but **never** their absolute absence.
- This reliance on testing to catch upstream errors results in **V&V** activities taking up to **70% of total development.**
- This prohibitive cost stifles innovation and delays the deployment of next-generation avionics systems.

Overcoming the Grounding Bottleneck with s(CASP)

- Traditional formal methods often struggle with the scale of avionics due to the "*grounding bottleneck*," where all variables must be instantiated before solving.
  - Impossible for continuous data like altitude or airspeed.
- Our framework utilizes **s(CASP)**, a goal-directed Answer Set Programming system that avoids grounding.
  - It models complex, continuous flight dynamics by using top-down, query-driven execution to mathematically bound the conditions where safety requirements might fail.

Continuous Verification via CLP(Q)

- Avionics systems must interact with the physical world, requiring reasoning over continuous time and space.
- Our framework integrates Constraint Logic Programming over rational numbers (**CLP(Q)**).
  - Unlike floating-point approximations, **CLP(Q)** represents rational numbers exactly without rounding errors.
  - The system can formally verify requirements involving complex flight equations and inequalities.
  - Ensures operational integrity across an infinite state space.

Automating the Traceability Matrix (RTM)

- Maintaining a Requirements Traceability Matrix is traditionally a manual, error-prone process.
- Our framework a
  justification tree
  - When a proper
    logical path fo
  - By mapping sy
    #pred directive
    human-readab

Requirement Human-readable	s(CASP) query Symbolic logic	Justification tree Logical path	#pred output RTM entry
Altitude bounds Alt ∈ [1000, 35000 ft]	`safe(Alt,_)` `altitude_ok(Alt)`	safe ← alt_ok ← constraint held	Verified Altitude is safe
Airspeed limits Speed ∈ [80, 340 kn]	`safe(_,Speed)` `speed_ok(Speed)`	safe ← spd_ok ← constraint held	Verified Speed is safe
Trajectory safety Continuous path valid	`safe_path(T)` `clp_check(T)`	path ← clp_ok ← bounds satisfied	Verified Path is safe
IMA partitioning Resource isolation	`isolated(A,B)` `no_conflict(A,B)`	isolated ← no_conf ← partition held	Verified Apps isolated

RTM auto-generated via #pred directive — symbolic logic mapped to human-readable certification evidence

Conclusion and Future Work

- The complexity of modern avionics has pushed the traditional V-Model to its economic and technical **limits.**
- A "*Correctness by Construction*" paradigm is now mathematically **necessary** to ensure safety.
- Our framework automates traceability, prunes complex search spaces, and verifies continuous flight logic.
- **Future work** involves integrating this into Model-Based Systems Engineering (MBSE) toolchains and extending the methodology to autonomous UAVs and Advanced Air Mobility platforms.