National Aeronautics and Space Administration
Jet Propulsion Laboratory
California Institute of Technology

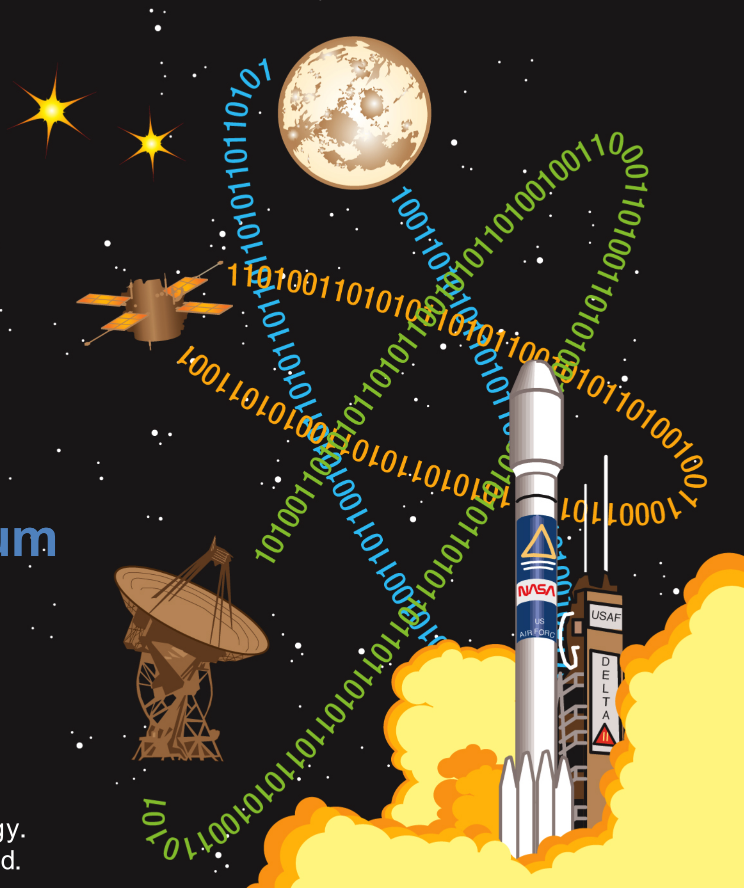
Talking Points on Reducing Software Vulnerabilities Formal Methods

Dr. Richard Doyle
JPL Space Asset Protection Team

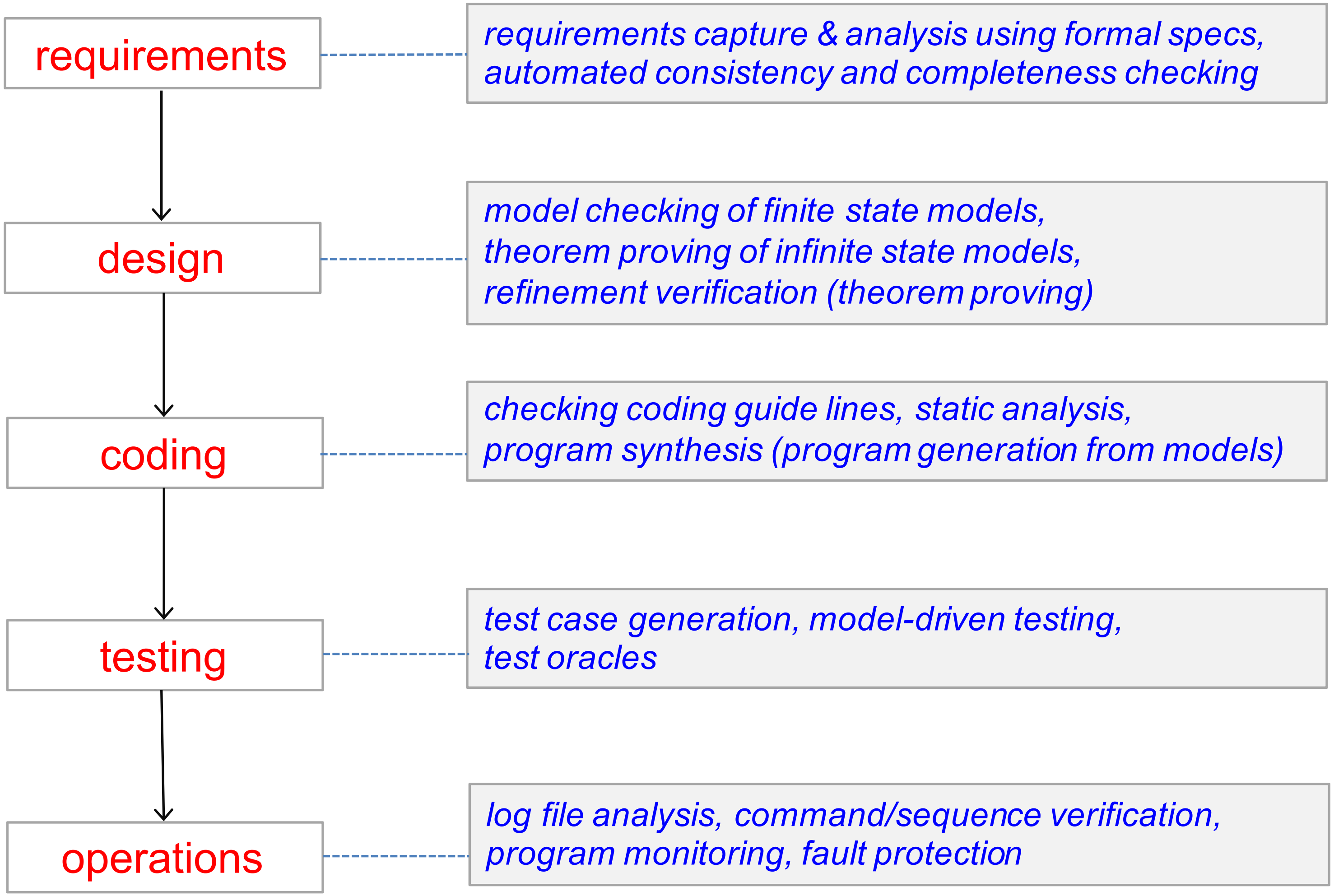
with contributions from
Dr. Rajeev Joshi, Dr. Klaus Havelund
JPL Laboratory for Reliable Software

Software and Supply Chain Assurance Forum
The MITRE Corporation
McLean, VA

July 14, 2016



Formal Methods
Lifecycle Model



requirements

requirements capture & analysis using formal specs, automated consistency and completeness checking

design

model checking of finite state models,
theorem proving of infinite state models,
refinement verification (theorem proving)

coding

checking coding guide lines, static analysis,
program synthesis (program generation from models)

testing

test case generation, model-driven testing,
test oracles

operations

log file analysis, command/sequence verification,
program monitoring, fault protection

Formal Methods
Approaches and Assessment

Metric/ Method	Properties	Coverage	Scalability	Effort	Application and Trend
Dynamic analysis (DA)	A+	D	A	B	Up and coming field. Monitoring, security, machine learning.
Static analysis (SA)	C	A+	A+	A+	Commercialized and used in practice. Millions of lines of code.
Model checking (MC)	B	A	C	C	Trend towards MC of code. Competitions. Use of parallelism/cloud
Theorem proving (TP)	A+	A+	D	D	Trend towards TP of code. To become part of dev. environments (IDEs)
Program synthesis (PS)	B	A+	D	B	Trend towards program sketching. AI: planning and scheduling.

Formal Methods
Experience internal to JPL

Dynamic Analysis

Log file analysis (LADEE command checking, MSL telemetry analysis),
Randomized differential testing (MSL/SMAP flash file system)

Static Analysis

Integrated with peer code reviews (using the *scrub* tool),
Custom checkers for checking JPL coding standards for C & Java,
Required for all JPL flight code

Model Checking

Used for critical modules (MER arbiter, MSL/SMAP data management, Cassini DRS),
Model-Driven Verification technique developed for checking C code using SPIN

Theorem Proving

Analysis of req'ts expressed in the K language for the planned Europa mission

Program Synthesis

State-Machine auto coder (MSL)

Formal Methods
Experience external to JPL

Dynamic Analysis
Deadlock and data race analysis,
Model-based testing

Static Analysis
Custom checkers for coding standards for many languages,
Analysis of runtime errors,
Commercial industry: Coverity, Code Sonar, Semmle, …

Model Checking
Flood control, ATT switch, Deep Space 1,
B&O audio video protocol

Theorem Proving
SEL 4 kernel, Microsoft hypervisor, Pentium floating point,
Formulation and proofs of aerospace theories (NASA Langley)

Program Synthesis
State-Machine auto coders,
Spreadsheet formulas (Microsoft)

Formal Methods *Open Problems, Recommendations*

- Main problems:
  - DA: monitoring with low impact, increase expressive power of spec. languages.
  - SA: reduce false positives, increase expressive power of checks performed.
  - MC: model checking using many CPUs. MC of code directly.
  - TP: guessing loop invariants in theorem provers. Automated SMT.
  - PS: finding the right abstraction level from which to generate code.

- Integration of formal methods with:
  - graphical model-based engineering systems (UML, SysML, …), preferably: design new unified approach(es).
  - programming, programming languages that are designed for abstraction, modeling and verification.
  - programming IDEs. It becomes an extension of the standard type checker.

- Combine techniques into unified framework.

Focus on design phase
Addressing software complexity

Spot: A computer language specifically targeted at testability, verifiability and validation of complex software systems

***The problem with software**: uncontrolled state space and complexity*

***Why Spot is different**: Spot manages and constrains state space*

- *In Spot we discretely identify state* parameters and place them in well defined structures, noting constraints such as valid ranges and important state combinations or sequences
- *In Spot, we retain only those distinctions in state space* that are meaningful relative to mission objectives – not all state distinctions are useful
- *In Spot we tightly control* the configuration and use of memory and inter-module communication that can cause state space expansion, logic errors and other programming hazards

Benefit:

- A run time monitoring system can check system state for correctness, and an external tester such as Spin can automatically generate and apply millions of test vectors, generate models and perform analysis to verify correct operation.

Credit: R. Some

Focus on test phase
Scenario-based randomized testing

Motivation

Traditional integrated testing focuses on scripted scenarios, each exercising a single system feature
Does not usually explore *interactions* among features that lead to unexpected behaviors
Manually writing individual tests to exercise multiple features is expensive

Approach

Test engineers write “scenario skeletons” in declarative form (easy to read and maintain)
Each scenario skeleton exercises a specific function or feature by specifying
- initial state assumptions, commands for exercising function, and properties to be checked
From such a description, a test engine automatically generates large numbers of test cases

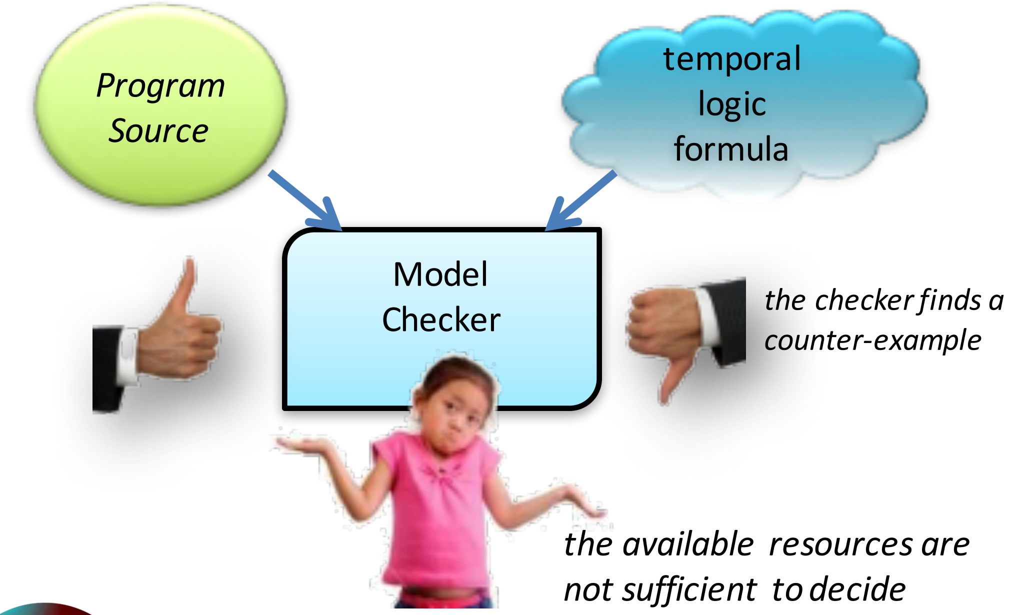
Benefits

Randomization forces system into unexpected corners (not biased by human expectations)
A declarative notation makes it easier to write new tests quickly
Easy to parallelize

Credit: R. Joshi

Focus on verification phase
Software model checking

- given a formula in linear temporal logic and a program, a model checker tries to find executions of the program that violate the formula


Program Source
temporal logic formula
Model Checker
the checker finds a counter-example
the available resources are not sufficient to decide

the Spin Model Checker, developed and maintained by JPL's Gerard Holzmann, is a popular explicit-state logic model checking tool.

It uses several strategies to deal with "state space explosion" problems.

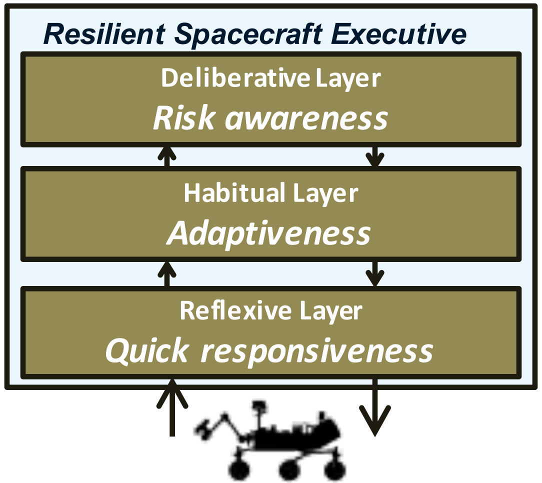
Spin checked

NASA/JPL Laboratory
for Reliable Software

Credit: G. Holzmann

Resilient Risk-Aware Autonomy for the Exploration of Uncertain and Extreme Environments
Use of Correct-by-Construction Techniques

Artist's Concepts



Objective: Develop a *Resilient Spacecraft Executive* to:

- adapt to component failures to allow graceful degradation
- accommodate environments, science observations, and spacecraft capabilities that are not fully known in advance
- make risk-aware decisions without waiting for slow ground-based reactions

Why this is important to NASA and JPL:

- Enables robotic explorations of harsh, remote, and inaccessible destinations
- Reduces operational risk and associated cost

FY15: Design and develop core algorithms of RSE; develop formal behavior models; validate algorithms through small-scale demo using simulation, rover testbed in Mars Yard, and AUV submarine. **FY16**: Integrate algorithms and behavior models; deploy RSE on simulator/hardware for Venus lander and/or Mars rover scenarios.

JPL Team

Dr. Mitch Ingham
Dr. Hiro Ono
Dr. Tara Estlin
Dr. Leslie Tamppari
(JPL)

KISS-funded collaborators

Prof. Richard Murray
(Caltech)

Prof. Brian Williams
(MIT)

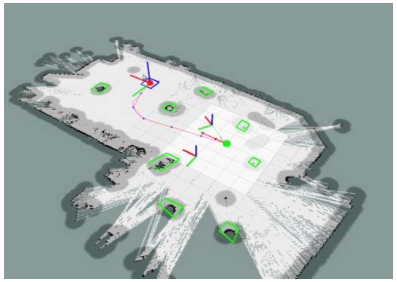
Dr. Richard Camilli
(Woods-Hole O.I.)

Overview of Approach and Early Results:

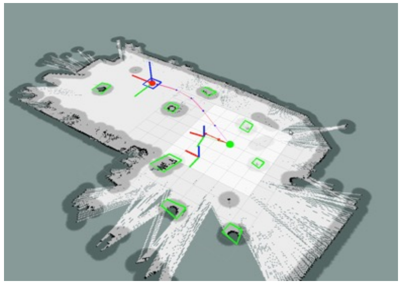
System adapts its behavior depending on acceptable level of risk

Keep the risk low.

You can take more risk.



Low Risk



High Risk

Focus on operations phase

Risk-aware autonomy