Advanced Verification and Validation Methods for Cyber-Physical Systems

James Kapinski,
Joint work with Hisahiro Ito, Ken Butts,
Jyotirmoy Deshmukh, and Xiaoqing Jin
October, 2017

MBD
TOYOTA TECHNICAL CENTER

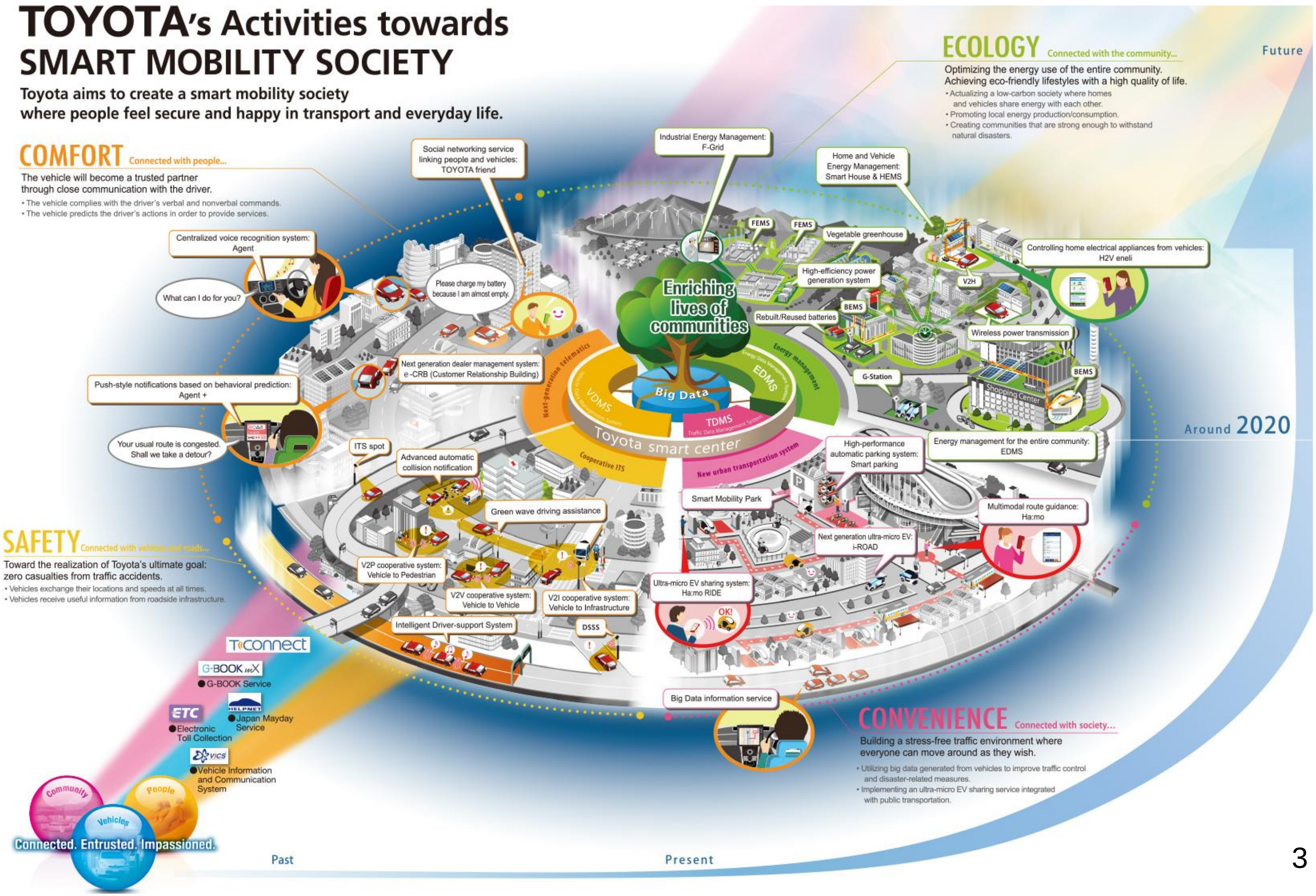
Outline

1. Cyber-physical Systems for Verification & Validation
   - Perspective
   - Analysis for CPS
     - Formal setting
     - Emerging techniques
2. Requirements Engineering
   - Ongoing challenges
   - ST-Lib: Library of formal requirements for CPS applications

MBD
TOYOTA TECHNICAL CENTER

J. Kapinski

CPS will be everywhere!



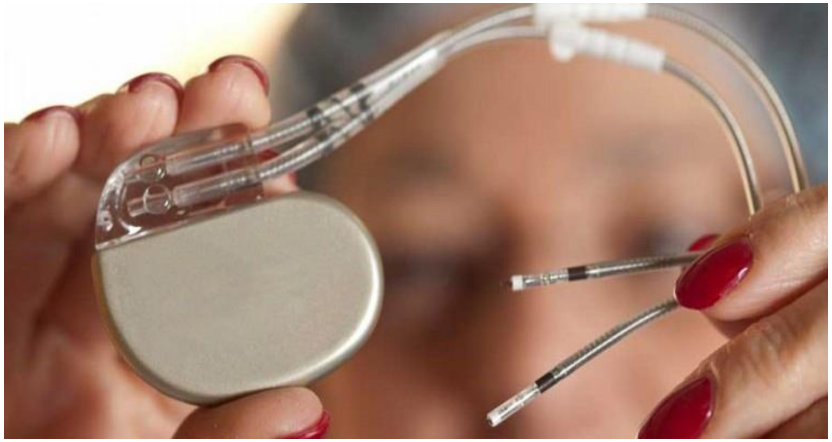
http://www.toyota-global.com/innovation/intelligent_transport_systems/mobility/images/its_mobility_02_large.jpg

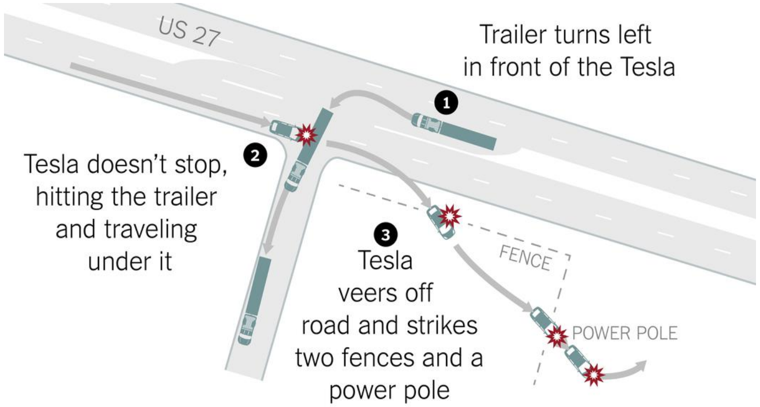
CPS is safety critical!

- CPSs used in safety critical applications
  - Automotive powertrain control
  - Smart grids
  - Aerospace control
  - Medical devices
  - …

MBD TOYOTA TECHNICAL CENTER

CPS is safety critical!





The FDA has issued 23 recalls of defective devices during the first half of 2010, all of which are categorized as "Class I," meaning there is "reasonable probability that use of these products will cause serious adverse health consequences or death."

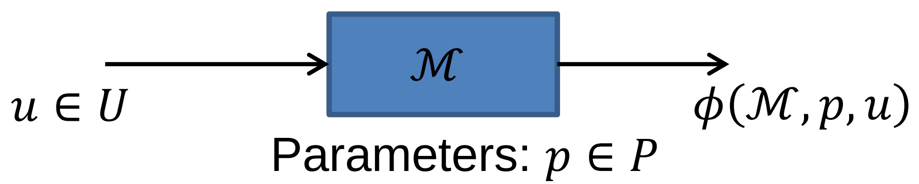
TOYOTA TECHNICAL CENTER

FORMAL ANALYSIS SETTING

Setting

<u>Definition (System):</u>
System $\mathcal{M}$ is some manifestation of a dynamical system whose behaviors $\phi(\mathcal{M}, p, u)$ are determined by parameters p and inputs u

- Generally, $\mathcal{M}$ can be a model, a test experiment (e.g., HILs, SILs), or the physical system
- For simulation and analysis, we will assume $\mathcal{M}$ is a model of the system (e.g., Simulink)
- Note that p and u can be taken from (possibly infinite) sets P and U

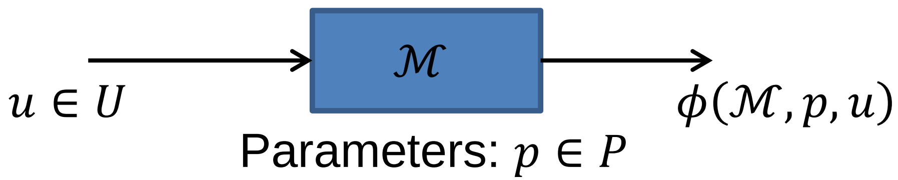


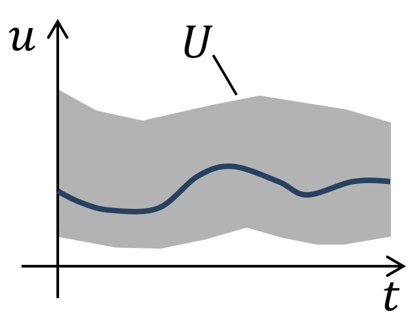
TOYOTA TECHNICAL CENTER

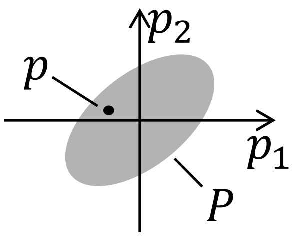
Setting

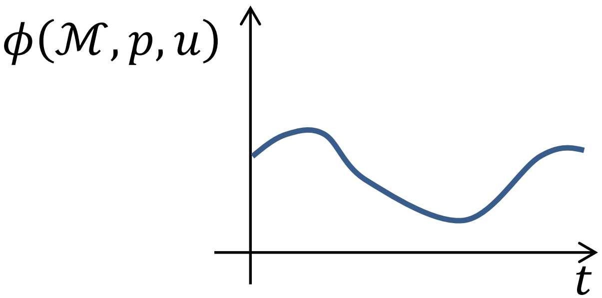
<u>Definition (Simulation):</u>
Process of generating $\phi(\mathcal{M}, p, u)$, which are the behaviors of $\mathcal{M}$ given parameters p and inputs u

- Assume simulations can be generated by numerical integration solver (e.g., Simulink)





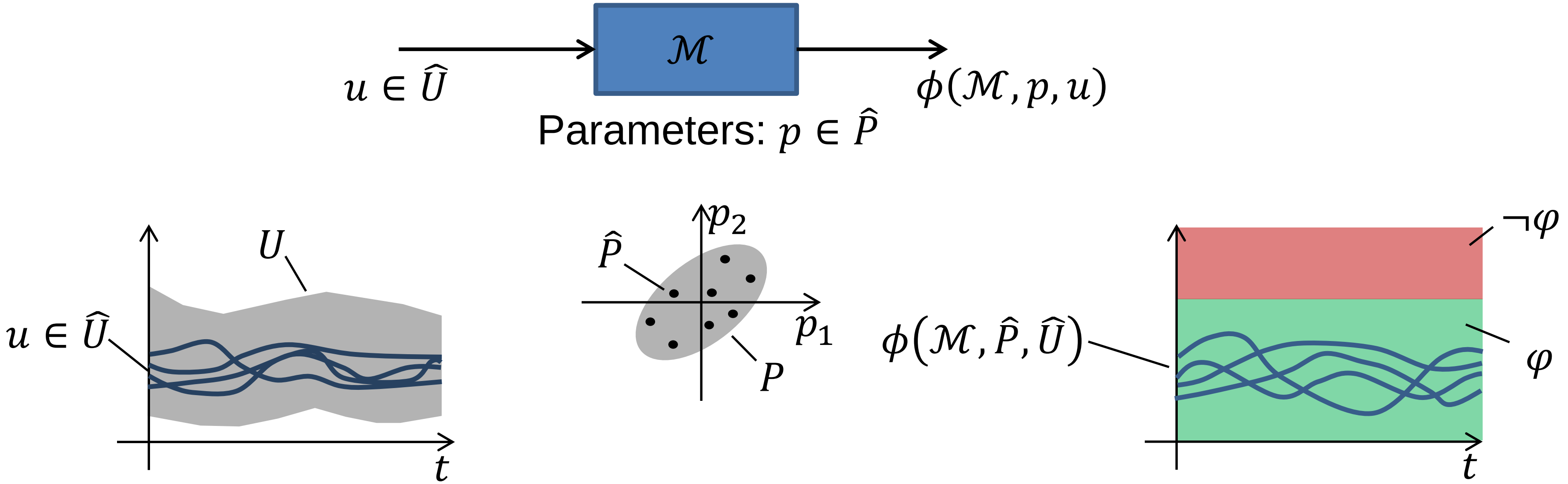




Setting

Definition (Testing):
Determine whether $\phi(\mathcal{M}, \hat{P}, \hat{U}) \models \varphi$ for given finite sets $\hat{P} \subseteq P$ and $\hat{U} \subseteq U$

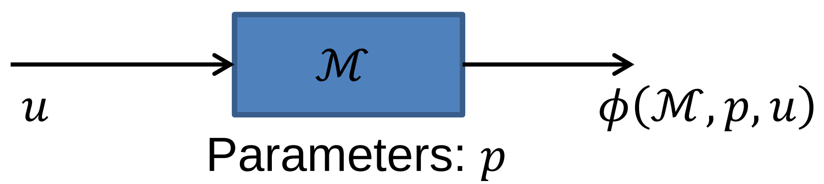
- Testing does not guarantee φ holds for all $p \in P$ and $u \in U$

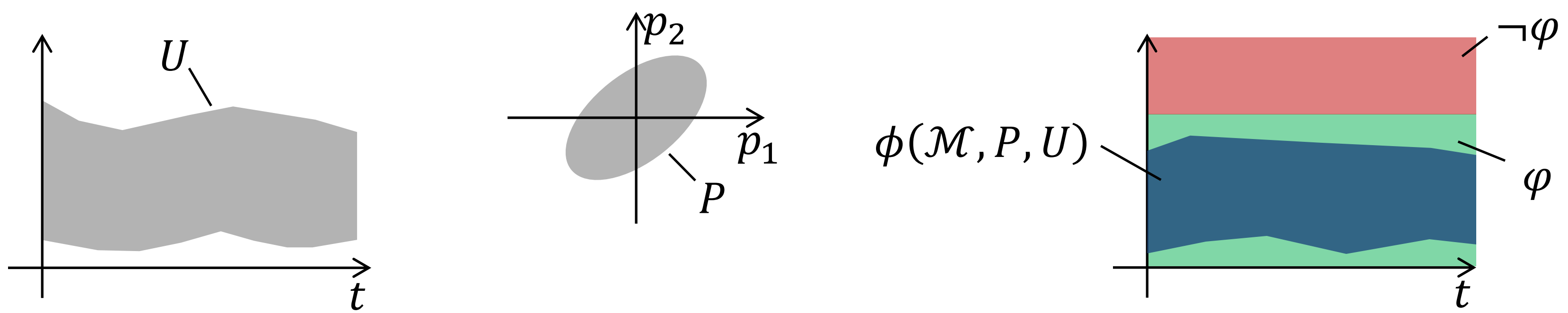


Setting

Definition (Verification):
Prove $\phi(\mathcal{M}, P, U) \vDash \varphi$ given P and U

- Proves φ holds for all $p \in P$ and $u \in U$

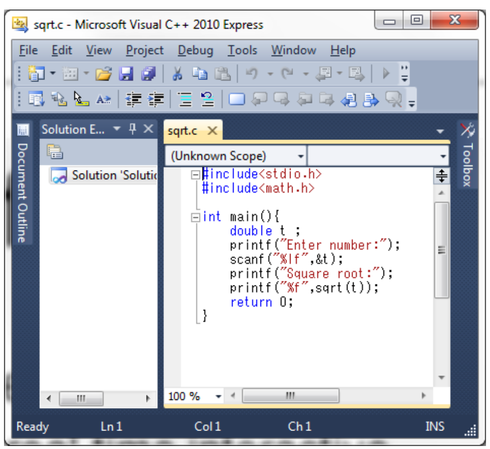


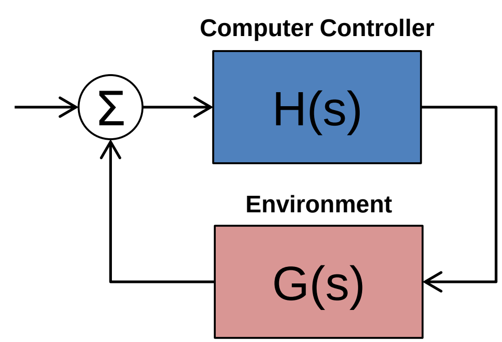


TOYOTA TECHNICAL CENTER

Software vs. Control Design

- Classical software design
  - Nontrivial verification questions for finite state models of software are hard
  - In general, proving nontrivial properties for software is **undecidable**
    - Σ_1 undecidable
- Embedded control system design
  - Nontrivial verification questions for even simple CPSs are **very undecidable**
    - Σ_2 undecidable†

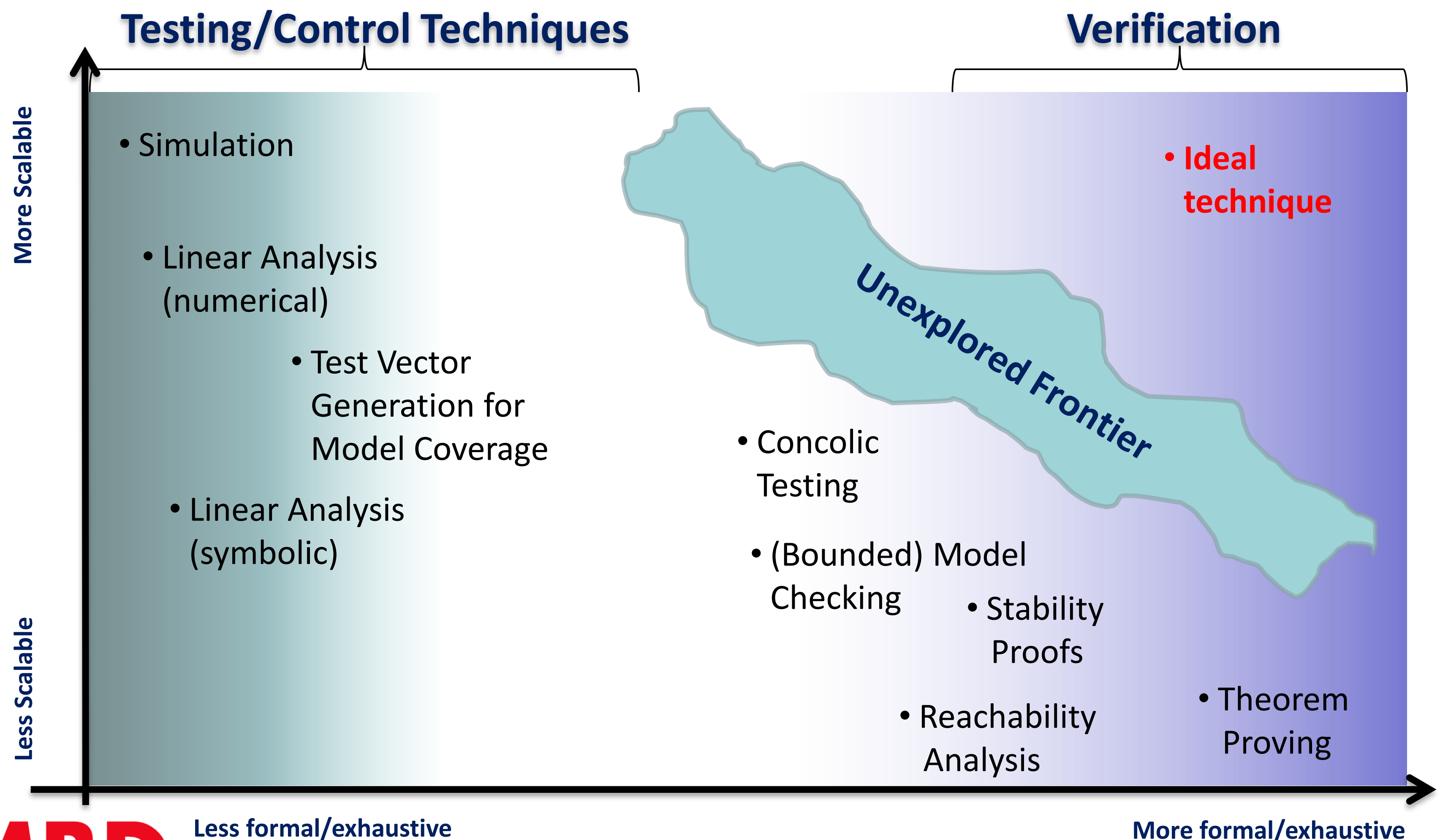




† E. Asarin and O. Maler. Achilles and the Tortoise Climbing Up the Arithmetical Hierarchy. *Journal of Computer and System Sciences*, Vol. 57, No. 3, pp. 389-398, 1998.

MBD TOYOTA TECHNICAL CENTER

Spectrum of Analysis Techniques



TOYOTA TECHNICAL CENTER

SIMULATION-BASED CHECKS FOR POWERTRAIN CONTROL

Why simulations?

- Help design validation
- Provide visual feedback
- Can use existing design artifacts
- Can uncover bugs
- Unlike formal verification, simulation does not require knowledge of:
  - Temporal Logic, SAT modulo theories, Bounded Model Checking
  - Simulations are cheap and usually fast
- Test-suites can be shared and built up across models

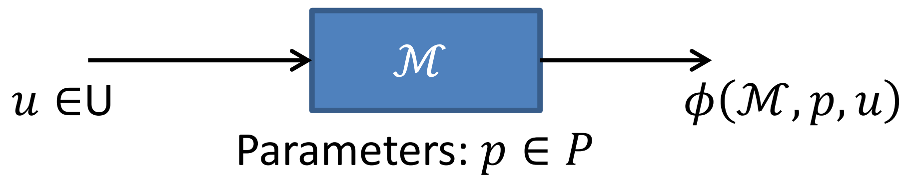
- Promising simulation-based approach: requirement falsification…

MBD
TOYOTA TECHNICAL CENTER

J. Kapinski

Requirement Falsification

<u>Definition (Falsification):</u>
Find parameters $p \in P$ and input $u \in U$ such that behaviors $\phi(\mathcal{M}, p, u)$ do NOT satisfy requirements φ (i.e., $\phi(\mathcal{M}, p, u) \nvDash \varphi$)



- Not verification, but **systematic bug-finding**
- No guarantees of completeness (except asymptotic/probabilistic)

TOYOTA TECHNICAL CENTER

Some key enablers

Robust satisfaction of φ by simulation trace $\phi(\mathcal{M}, p, u)$

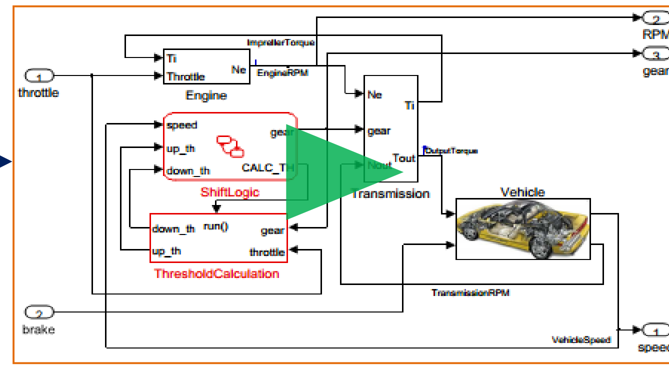
- A function maps φ and $\phi(\mathcal{M}, p, u)$ to $\mathbb{R}$
- Positive number = $\phi(\mathcal{M}, p, u)$ satisfies φ
- Negative number = $\phi(\mathcal{M}, p, u)$ does not satisfy φ
- Moving towards zero = moving towards violation

Black-box Global optimizers

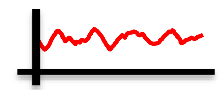
- Powerful heuristics to get close to global optimum

Falsification by optimization

u



$\phi(\mathcal{M}, p, u)$



φ

Optimizer:

Minimize robust satisfaction value

MBD TOYOTA TECHNICAL CENTER

Falsification supported by both **S-TaLiRo** and **Breach** tools

Requirement Falsification

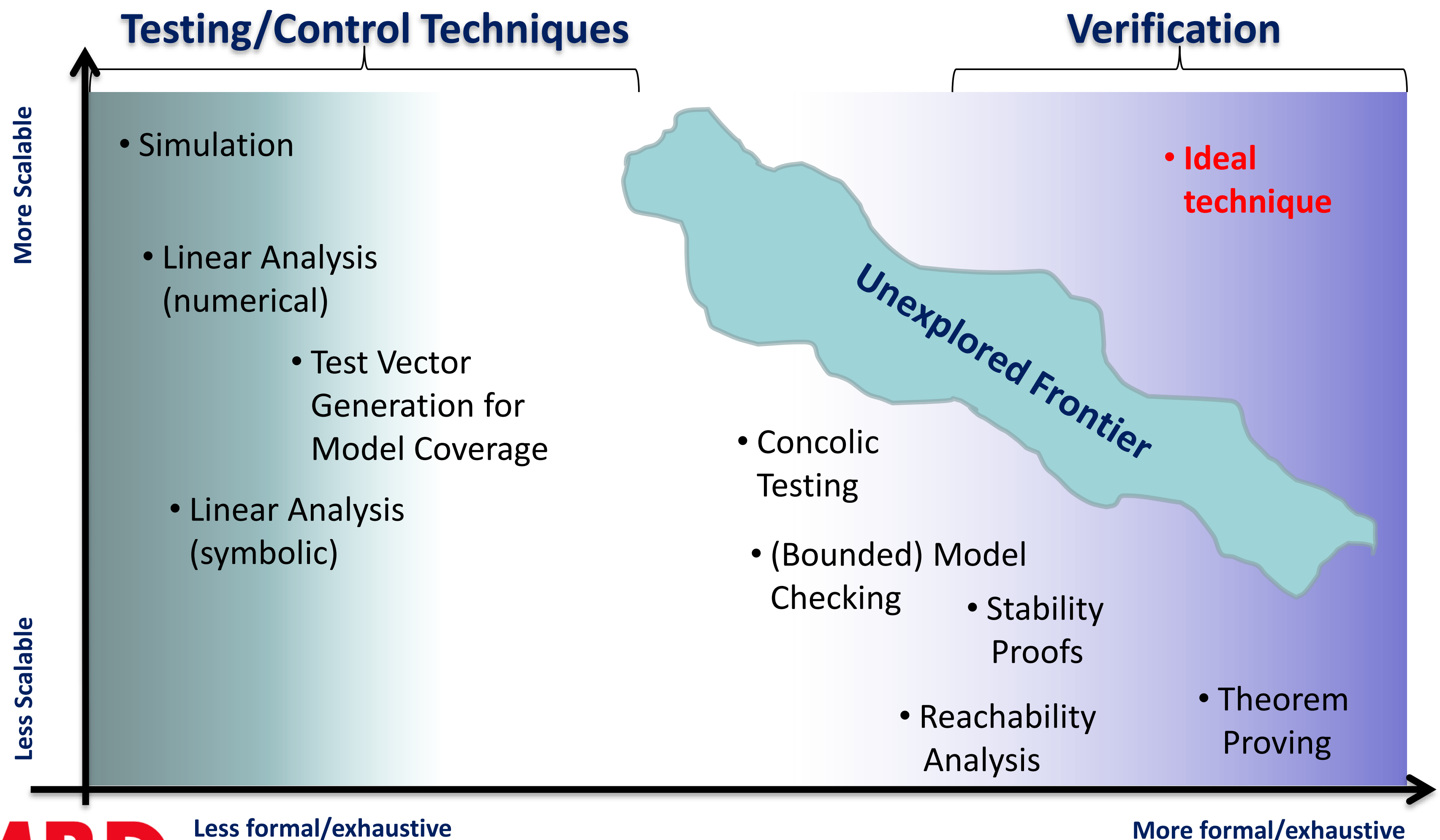
- Work by others
  - S-TaLiRo [Fainekos, Sankaranarayanan, et al.]
    - Metric Temporal Logic based requirements
    - Supports several stochastic optimizers
  - Breach [Donzé, CAV 2010, NSV 2013]
    - Signal Temporal Logic based requirements
    - Supports Nelder-Mead
    - Can exploit sensitivity info

Requirement Falsification

- Other things we've done in the past
  - Multiple Shooting [Zutshi, Sankaranarayanan, et al., EMSOFT 2014, HSCC2016]
    - Multiple short simulations segments leading from initial conditions to unsafe states; adjust initial conditions to piece segments together
  - Stochastic Local Search for Falsification [with Deshmukh, et al., ATVA 2015]
    - Discrete optimization method used as search heuristic
  - Simulation-based testing for coverage [with Dreossi, et al., NASA Formal Methods 2015, extensions with Adimoolam, et al., CAV 2017]
    - Selecting inputs to maximize coverage of infinite state-space
  - Simulation-based convergence/stability testing [with Sankaranarayanan et al., HSCC 2014, extensions with Balkan, et al., EMSOFT 2016]
    - Specifications in the form of Lyapunov-like function to test for convergence/stability

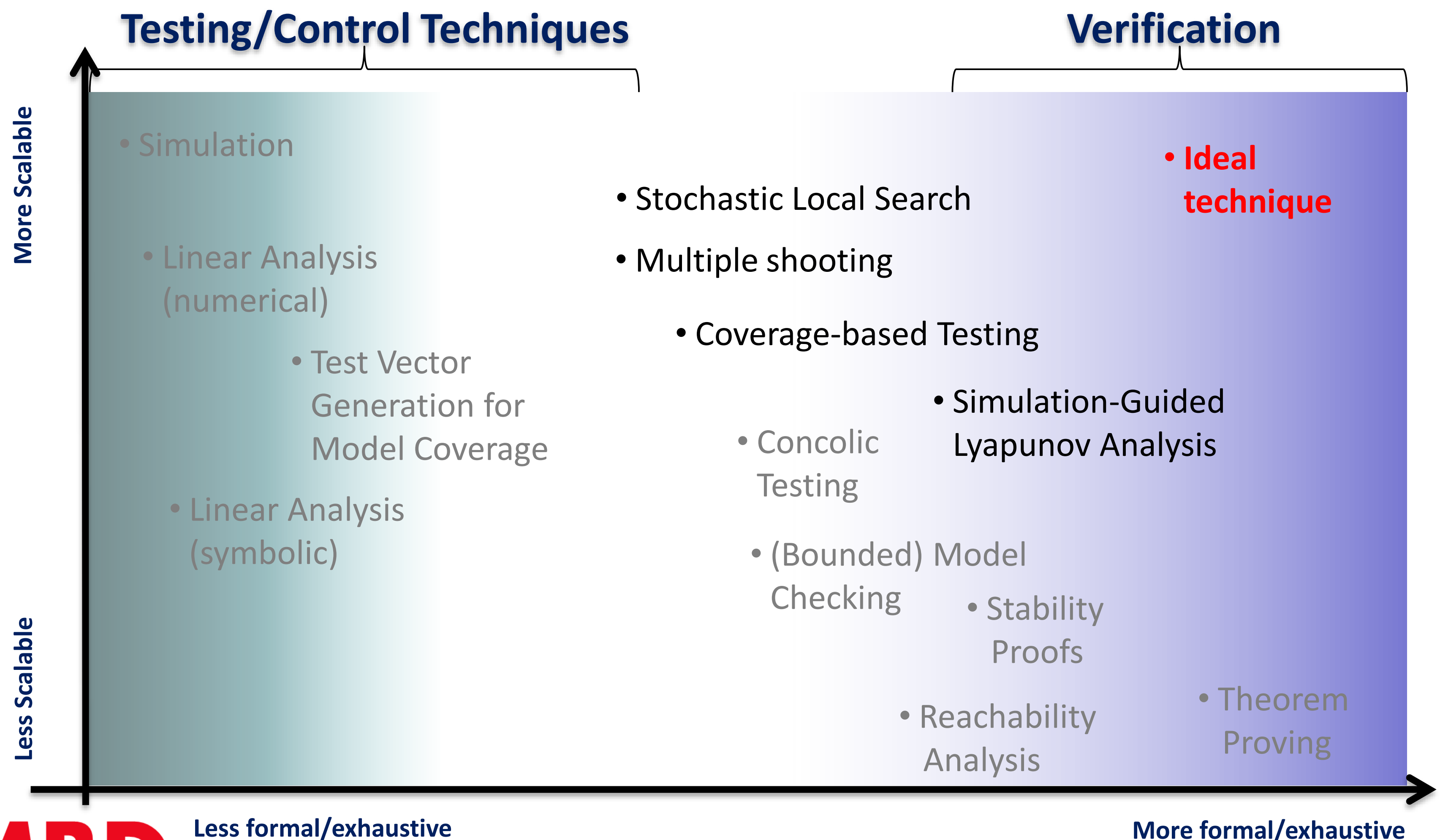
MBD TOYOTA TECHNICAL CENTER

Spectrum of Analysis Techniques



TOYOTA TECHNICAL CENTER

Spectrum of Analysis Techniques



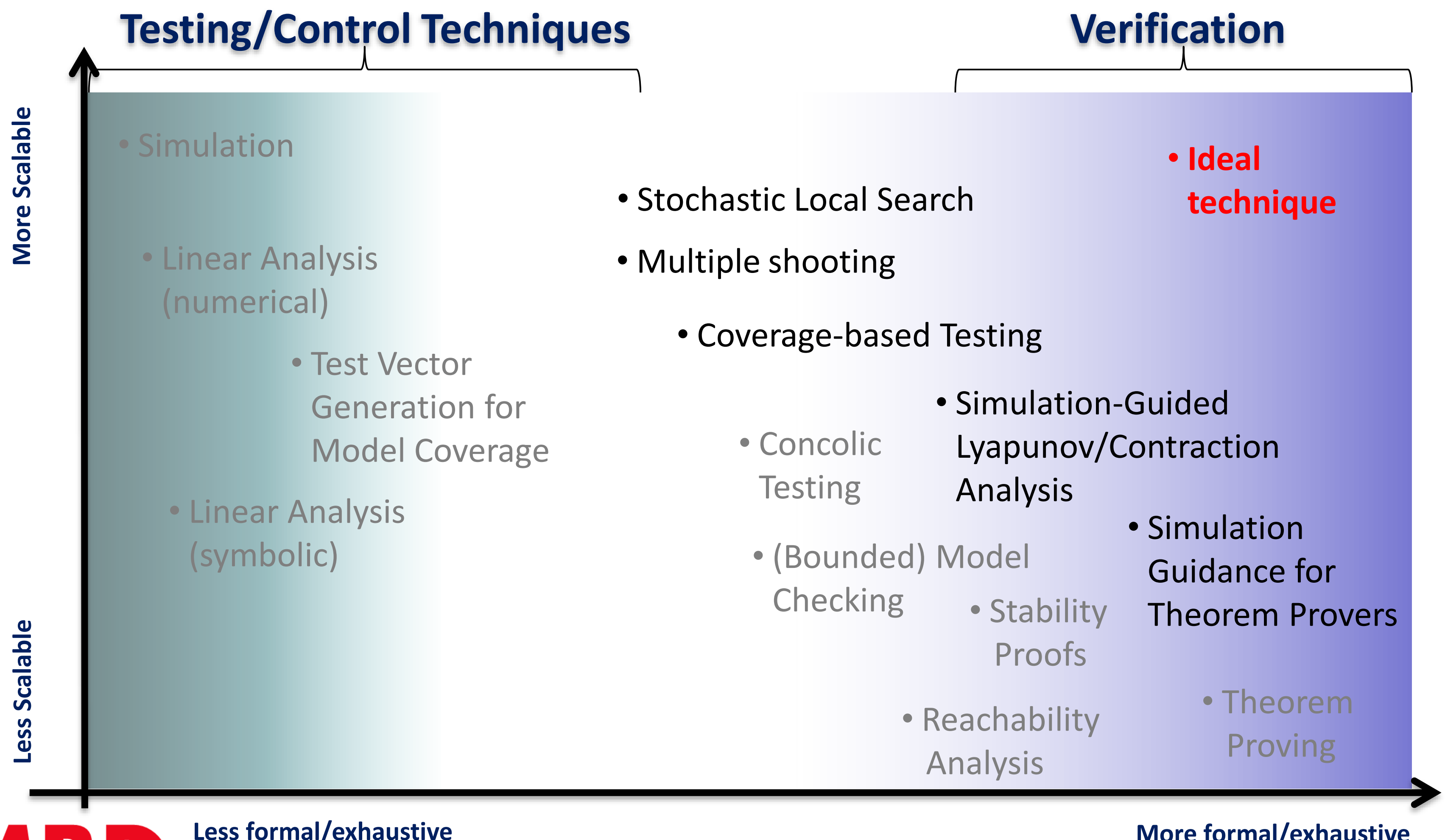
TOYOTA TECHNICAL CENTER

Other Simulation-based Methods

- Other things we've done in the past
  - Simulation traces to learn contraction metrics [with Balkan, et al., ICC, 2015]
    - Simulations to learn Lyapunov-like function showing *convergence*; used to compute *flowpipes* that contain all system behaviors; used for verification (proving safety for infinite sets of behaviors)
  - Simulation-based verification [with Fan, et al., EMSOFT 2016]
    - Simulations used to compute *flowpipes* to prove safety
  - Simulation traces to assist mechanical theorem provers [with Arechiga, et al., EMSOFT 2015]
    - Simulations used to learn invariant sets; invariant sets used in theorem prover to show safety

MBD TOYOTA TECHNICAL CENTER

Spectrum of Analysis Techniques



MBD
TOYOTA TECHNICAL CENTER

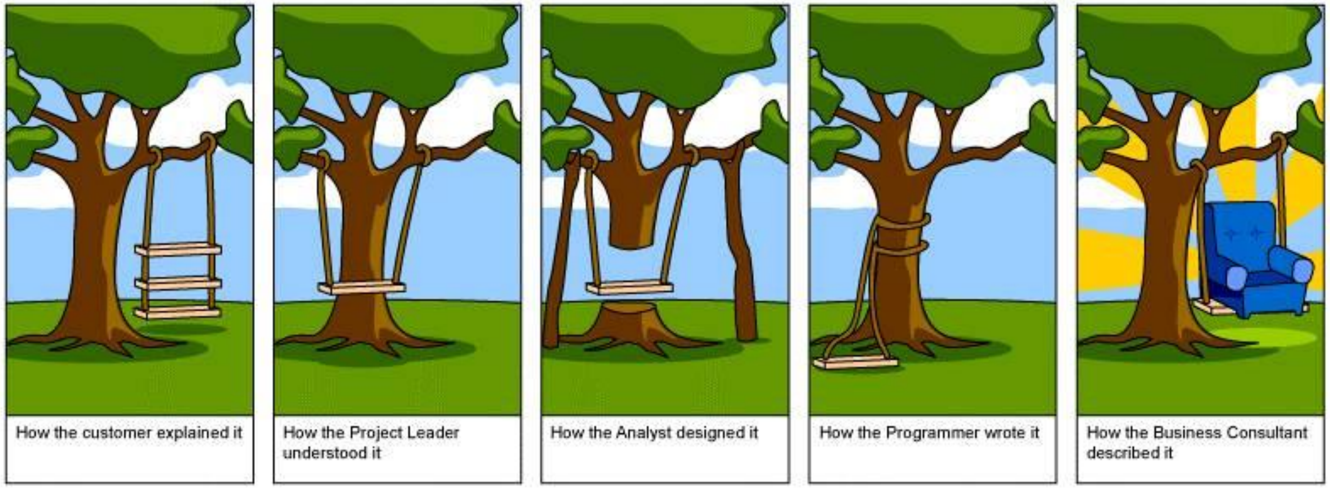


Image source: LinkedIn

REQUIREMENT ENGINEERING CHALLENGES

TOYOTA TECHNICAL CENTER

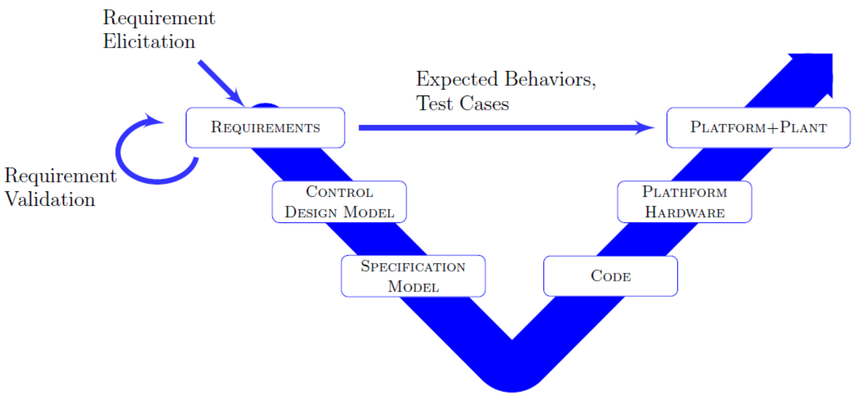
Requirement Engineering Challenges

- Outline
  - Overview of requirements engineering philosophy
    - Comparison of perspectives: software vs. CPS
    - Challenges
  - ST-Lib: collection of formal requirements for control engineering applications
    - Results and challenges applying ST-Lib

TOYOTA TECHNICAL CENTER

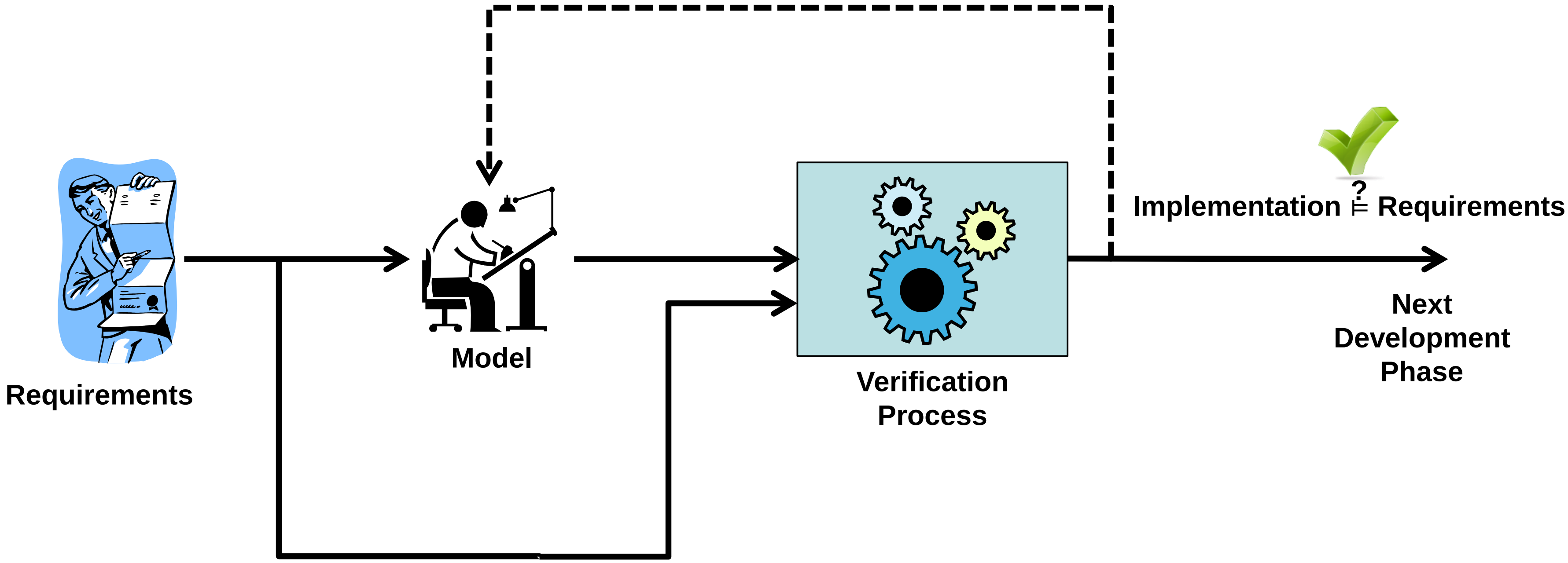
Requirements-Driven Approach

- Many of our efforts focus on providing a requirements-driven development approach
  - Requirements are developed and iterated on
  - Requirements used to develop control models and specify expected behaviors of models
  - Same requirements also used to define expected behaviors from calibration & test, as well as from the deployed system



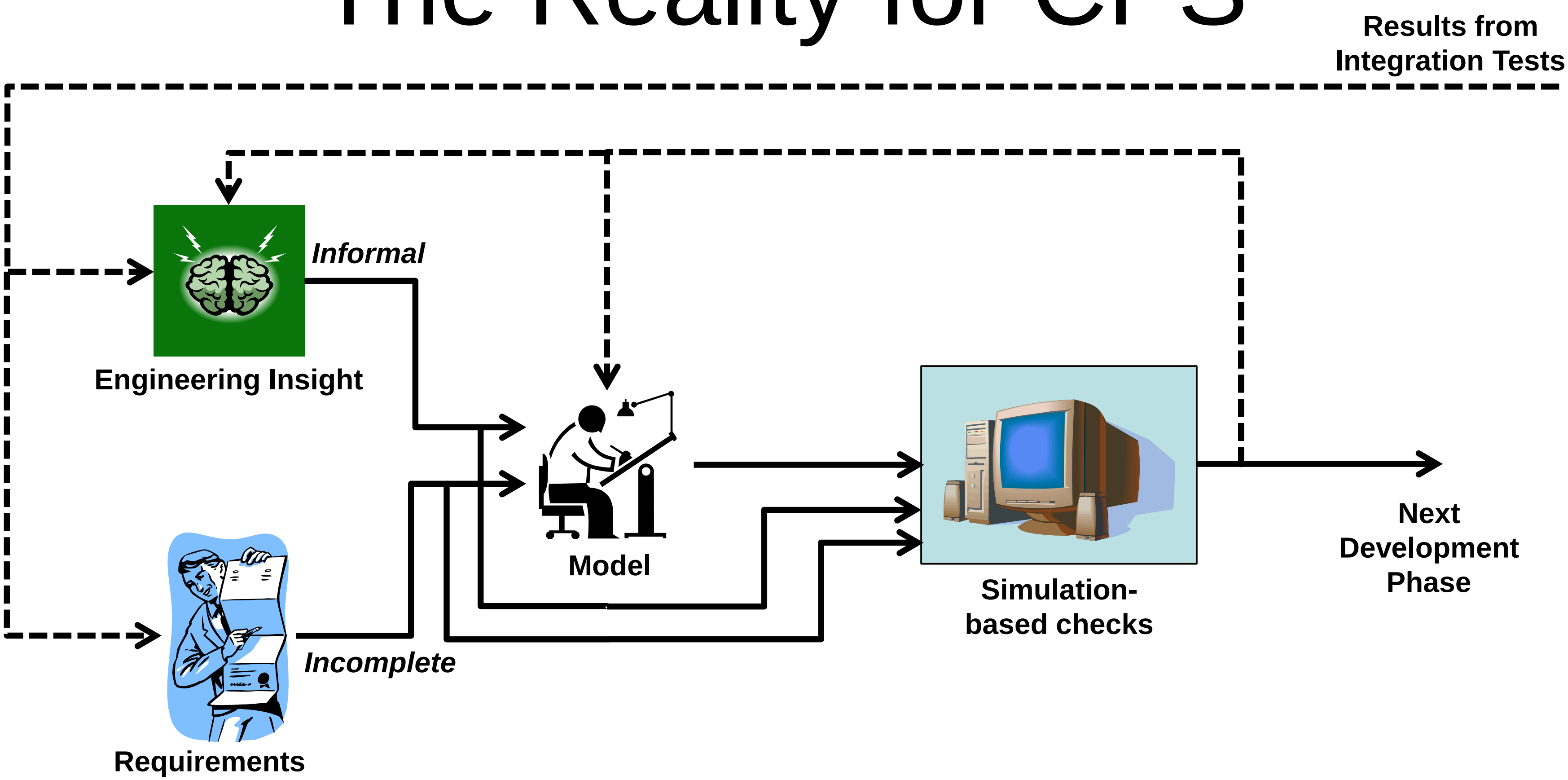
TOYOTA TECHNICAL CENTER

Classic Verification Assumption



MBD
TOYOTA TECHNICAL CENTER

The Reality for CPS

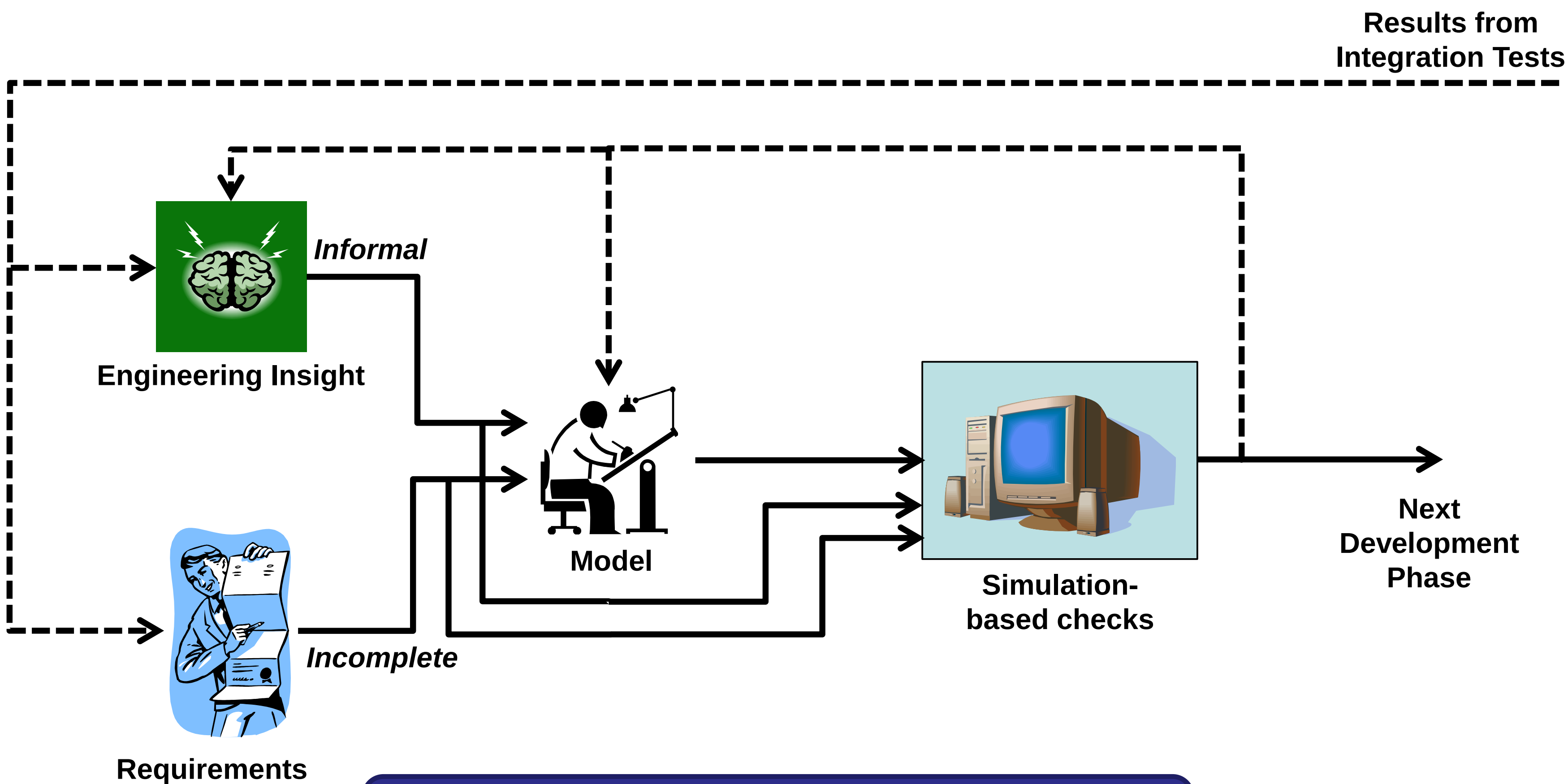


Takeaways:
1. Difficult/impossible to specify every aspect of CPS behaviors
2. Aspects of possible behaviors are discovered in simulation and testing phases

MBD
TOYOTA TECHNICAL CENTER

J. Kapinski

The Reality for CPS



Additionally: Hardware development taking place in parallel with controller development!

MBD
TOYOTA TECHNICAL CENTER

CPS Requirement Challenges

- Requirements are evolving due to CPS-related issues
  - System hardware/software designs evolve concurrently
  - Not possible to create a plant model that captures all behaviors
  - Subtle interactions between states/signals are not known before integration test
- Definition of correct behaviors exist only in engineer's brain
  - Formal requirements are hard for engineers to develop
  - Existing requirements do not capture all of the desired behaviors
    - Model may capture appropriate/expected behavior but requirements do not

MBD TOYOTA TECHNICAL CENTER

ST-LIB
SIGNAL TEMPLATE LIBRARY

MBD TOYOTA TECHNICAL CENTER

- J. Kapinski, X. Jin, J. Deshmukh, A. Donze, T. Yamaguchi, H. Ito, T. Kaga, S. Kobuna, S. Seshia. "ST-Lib: A Library for Specifying and Classifying Model Behaviors". Society of Automotive Engineers Technical Paper (SAE), 2016.

What is ST-Lib?

- **Is** a library for specifying and classifying signal patterns of system behaviors
- **Isn’t** a modeling language like Simulink for simulation

TOYOTA TECHNICAL CENTER

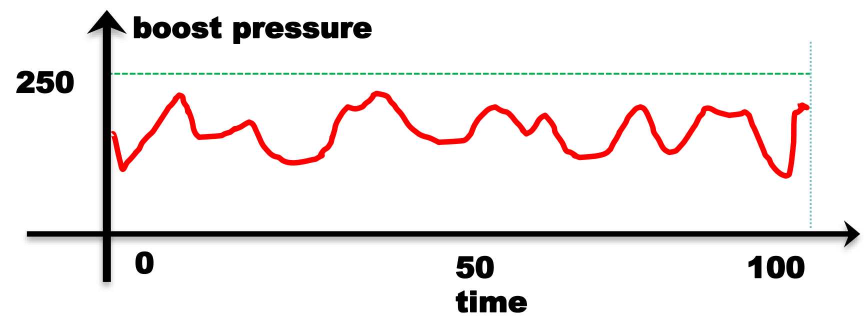
Why ST-Lib?

- **Can** formally specify intended design behaviors using a signal template
- **Can** automatically use simulation-based techniques to identify (near) worst-case behaviors of system

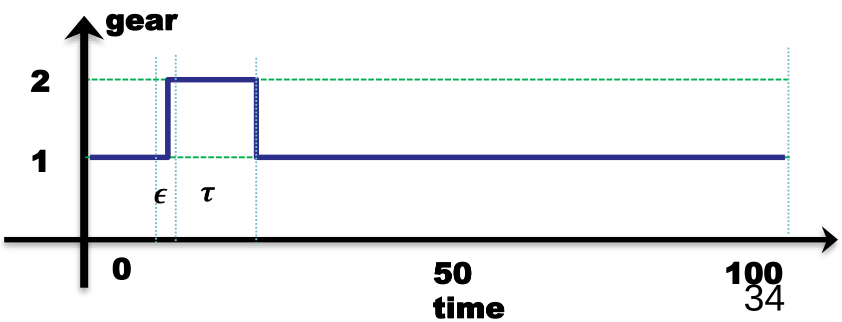
Introduction to STL

- Signal Temporal Logic (STL)
  - Specify timed behaviors of systems, containing:
    - Logic operators (∧, ¬, ∨, →)
    - Temporal operators (“always”, “eventually”, and “until”)
    - Atomic constraint formula ($f(x) \geq 0$)
  - Examples

$$always_{[0,100]}(\textbf{boost pressure} < 250)$$



$$always_{[0,100]}\big((\textbf{gear} = 1 \ \wedge eventually_{[0,\epsilon]}\textbf{gear} = 2\big) \rightarrow always_{[\epsilon,\tau+\epsilon]}(\textbf{gear} = 2))$$



TOYOTA TECHNICAL CENTER

ST-Lib

- ST-Lib uses STL to identify signal patterns of interest to design engineers, including:
  - Ringing
  - Spikes and glitches
  - Excessive overshoot or undershoot
  - Slow response time (settling, rising, or falling)
  - Undesirable timed relation behaviors
  - Steady state or tracking error

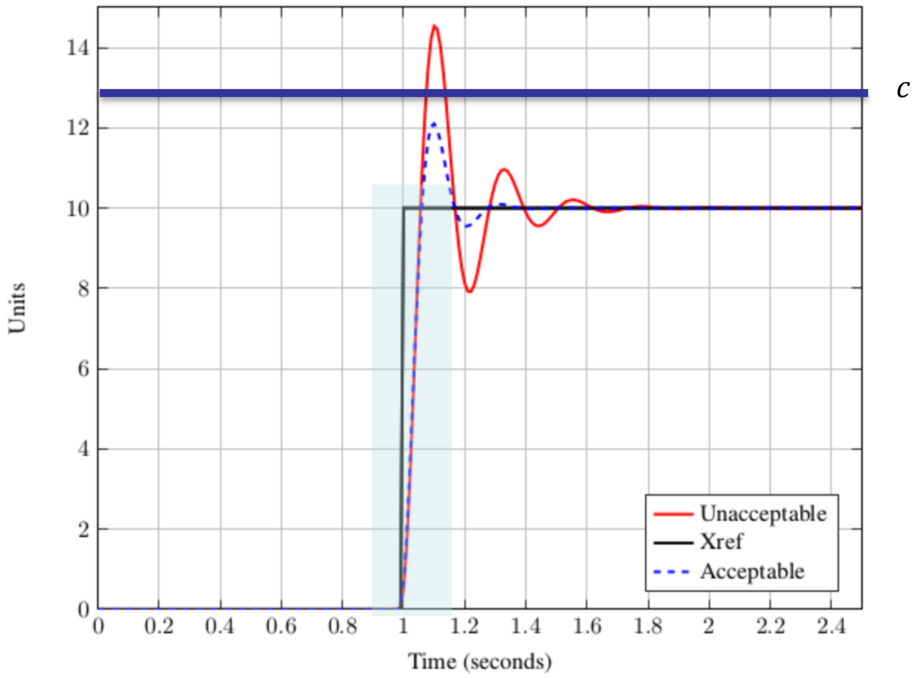
Example: Overshoot

$\varphi \coloneqq \mathit{eventually}_{[0,T]}\,(\mathit{step}(x_{ref}, r) \wedge \mathit{eventually}\,(x - x_{ref} > c))$

$\mathit{step}(x_{ref}, r) = x_{ref}(t+\epsilon) - x_{ref}(t) > r$

Note: Original ST-Lib requirements expressed *bad* behaviors:

- Bad behavior: $\varphi \coloneqq ev_{[0,T]}\,(\mathit{step}(x_{ref}, r) \wedge ev\,(x - x_{ref} > c)$
- Expected behavior: $\neg\varphi = alw_{[0,T]}\left(\mathit{step}(x_{ref}, r) \Rightarrow alw(x - x_{ref} < c)\right)$



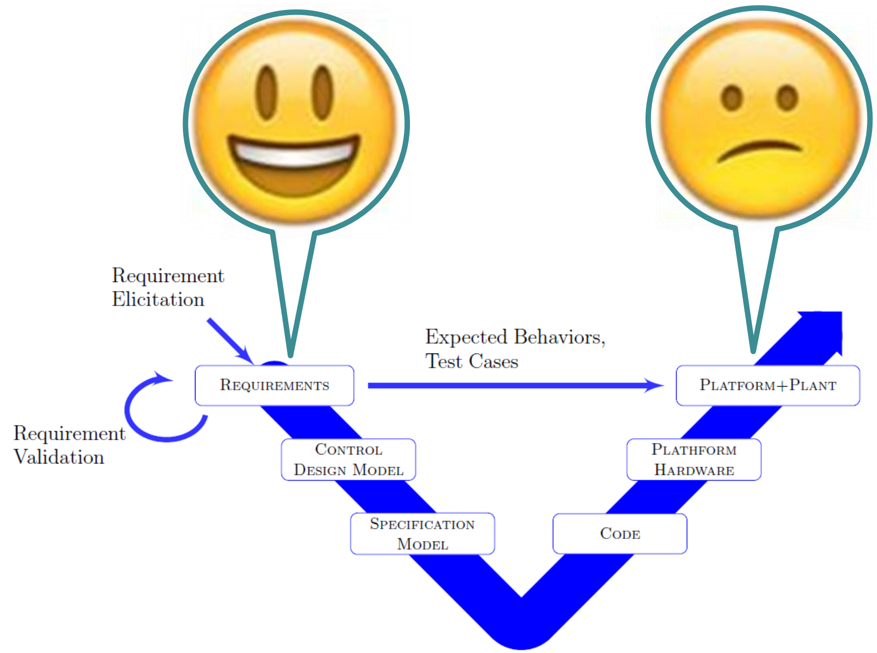
TOYOTA TECHNICAL CENTER

ST-LIB IN PRACTICE: LESSONS LEARNED AND CHALLENGES

MBD
TOYOTA TECHNICAL CENTER

ST-Lib Challenges

- Challenges applying ST-Lib in practice
  - Works well for simulation models and engineered input patterns
  - Does not work well with real data – particularly real input patterns
  - Does not account for subjective nature of many evaluation practices



TOYOTA TECHNICAL CENTER

ST-Lib Example

- ST-Lib application example
  - Applied versions of the following ST-Lib templates to a fuel cell (FC) vehicle powertrain application
    - Overshoot
    - Settling time
    - Rise time
    - Steady-state error

TOYOTA TECHNICAL CENTER

ST-Lib Example

- ST-Lib application example
  - Example: used the following version of the overshoot requirement

OVERSHOOT := alw ((STEPUP and $alw_{[dt,sstime]}$ not(STEP)) => $alw_{[dt,sstime]}$ (OVERSHOOTLIMIT))

STEPUP := in[t+dt]-in[t] > StepThresh
STEPDOWN := in[t]-in[t+dt] > StepThresh
STEP := STEPUP or STEPDOWN

OVERSHOOTLIMIT := out[t] < 1.1*(in[t])

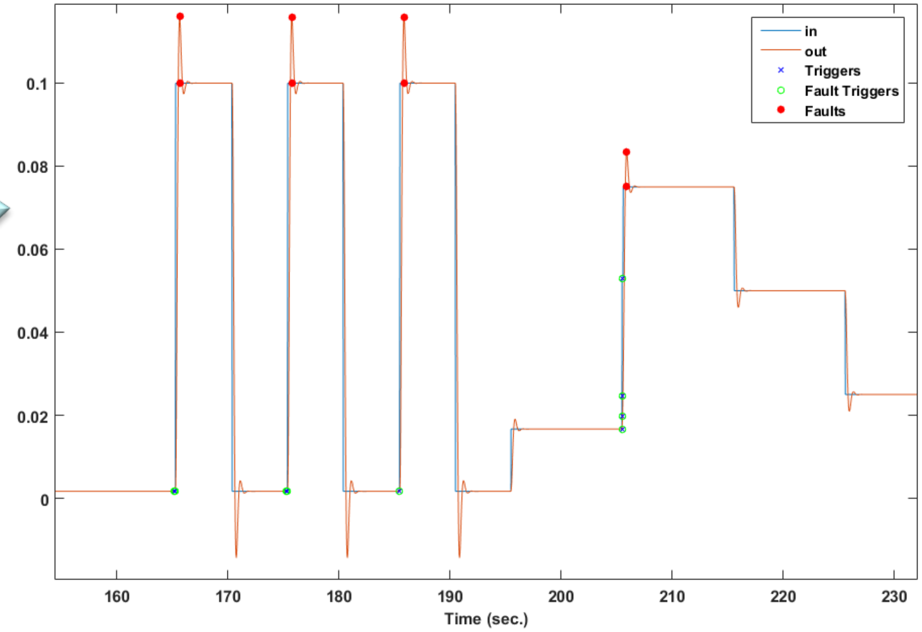
Comment:
- No other step should be present when checking the overshoot

sstime: Time over which steady-state is assumed to be reached
dt: Small constant, comparable to a sampling step size

ST-Lib Application Example

- Overshoot requirement performance
  - Good (expected) requirement performance for control model, using engineered input patterns

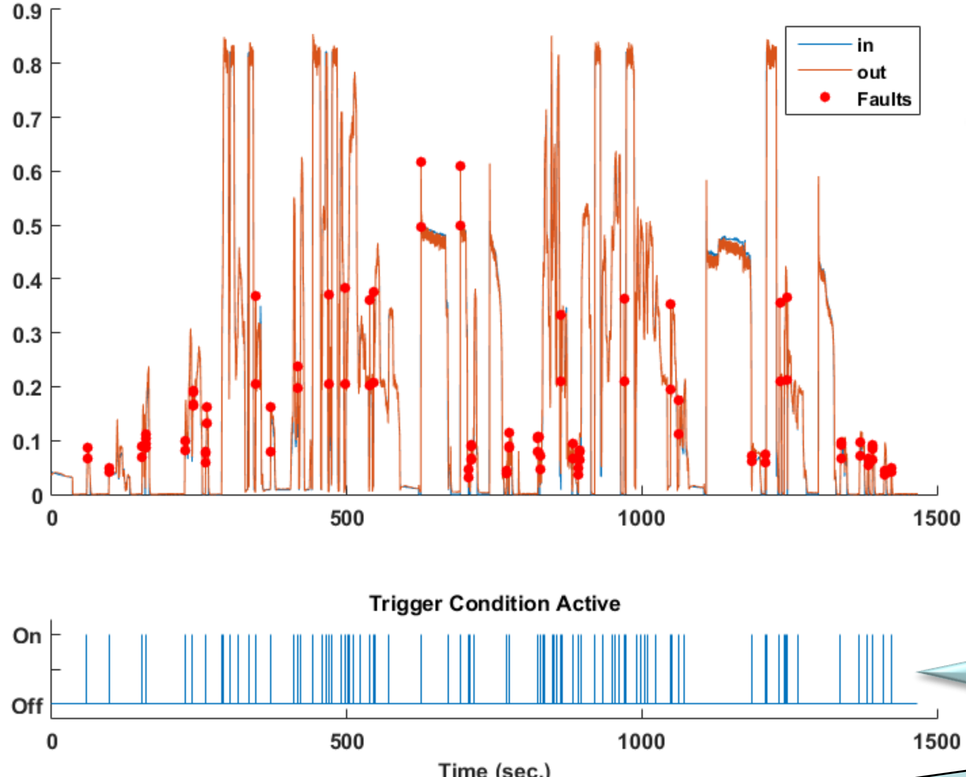
Anonymized data



Comments:
- Overshoot values are appropriately detected
- New fault localization tool use to highlight instants when faults occur

ST-Lib Application Example

- Overshoot requirement performance
  - Bad (unexpected) requirement performance for real data



Comments:
- Many unexpected behaviors are identified
- Other behaviors are mischaracterized

Lower plot shows moments when OVERSHOOT *antecedent* is true

antecedent

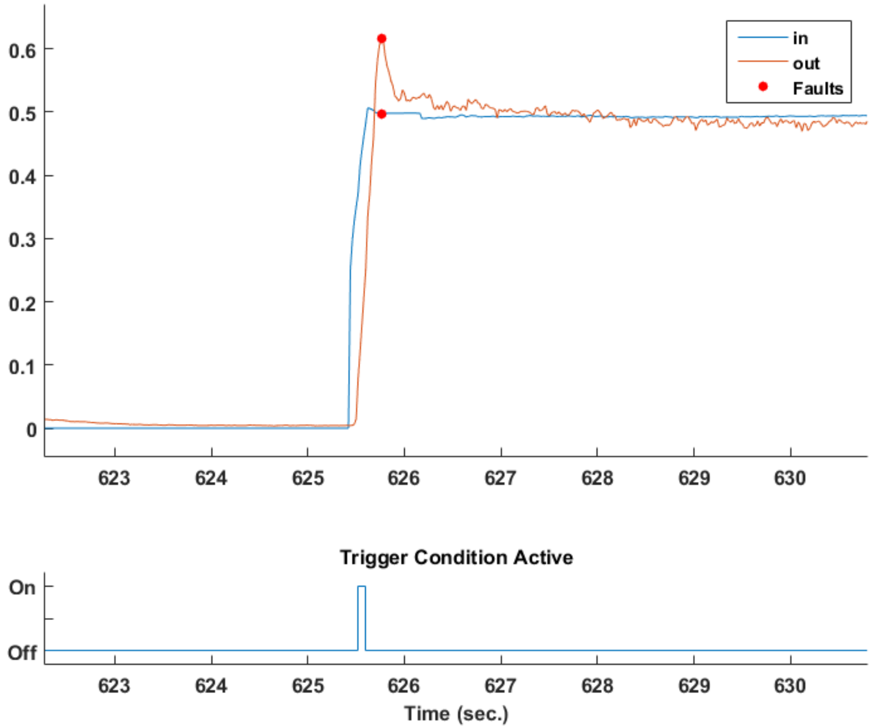
OVERSHOOT := alw ((STEPUP and alw$_{[dt,sstime]}$ not(STEP)) => alw$_{[dt,sstime]}$ (OVERSHOOTLIMIT))

Let's look at some reasons why there are problems…

TOYOTA TECHNICAL CENTER

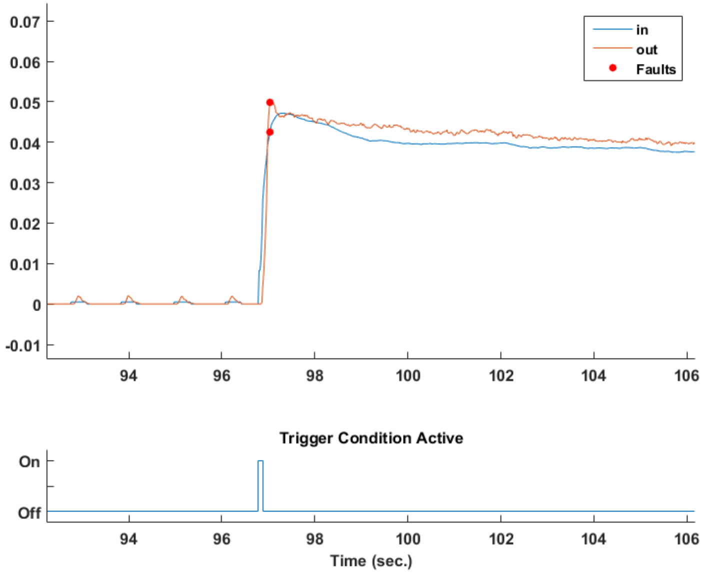
ST-Lib Application Example

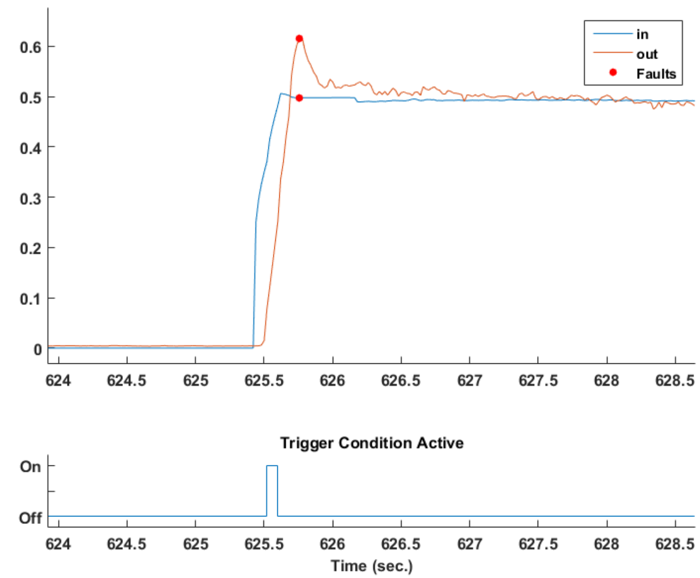
- This behavior is appropriately identified as a fault



ST-Lib Application Example

- Problem: Overshoot error tolerance fixed
  - Engineer wants a.) relative error limit for large reference values and b.) absolute error limit for small reference values



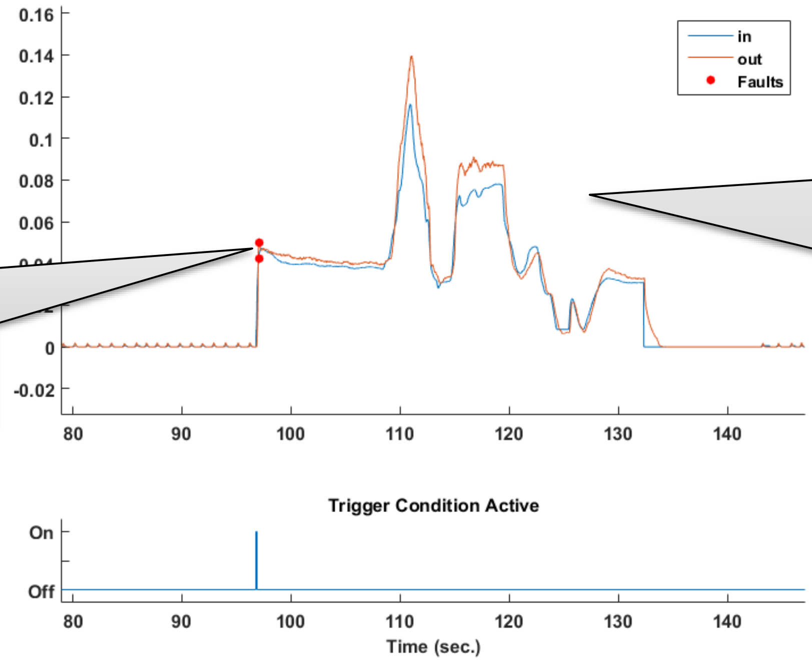


Comment:
- Using 10% overshoot error limit is too small for small reference values
  - Engineer not so concerned about error at low reference values

TOYOTA TECHNICAL CENTER

ST-Lib Application Example

- Problem: Many important behaviors are neither steps nor steady-state
  - These behaviors should fall under some other category of inputs, like an input *ramp*, with corresponding requirements



Comment:
- This is appropriately identified as an overshoot failure

Comments:
- These behaviors are not steep enough to be steps but not small enough to be steady-state
  - So the behaviors are not constrained in any way
- Want to make sure that all behaviors are somehow evaluated

TOYOTA TECHNICAL CENTER

ST-Lib Application Example

- Other challenges
  - Engineer had a notion of an ideal response (included a time-shifted, rate-limited version of the command signal)
    - Not easily captured in STL
    - Addressed by a priori defining a new signal
  - Using a fault localization (in time) tool
    - Very difficult for complex STL formulas

ST-Lib Challenges

- ST-Lib shortcomings
  - Scaling constants relative to command magnitude
  - Step is not the only meaningful input
  - For real data, need to define a partition on the input and relate a corresponding behavioral constraint on output
  - Need to allow for more subjective classification of reference signal class (step, ramp, SS, others…)

ST-Lib Challenges

- We created some STL-based solutions for all of the requirements, but…
  - The requirements represent approximations of what they actually want
    - The requirements do not capture faults with 100% accuracy (there are false positives/negatives)
  - The requirements are **very complicated**
    - Difficult to read/understand (this reduces the value of the requirement)
    - Are computationally expensive to monitor in Breach

MBD
TOYOTA TECHNICAL CENTER

ST-Lib Challenges

- General formal requirements challenge
  - Subjective nature of behavior expectation difficult to capture with temporal logic (like STL)
    - We are capturing (poorly) right now using complex STL requirements
  - Need improved methods to capture designer intentions
    - Alternatives:
      - Write code that would monitor appropriate behaviors
        - » **Downside**: specification in the same language as the design (uses a program to specify correct behavior of a program)
      - Train a NN to classify (good/bad) behaviors like the engineer would
        - » **Downside**: Essentially provides a specification that is a black box
    - Other ideas?

MBD TOYOTA TECHNICAL CENTER

Summary

- CPS is everywhere and is safety critical
  - Verification for CPS is hard!
  - New simulation-based analysis techniques
    - Simulation-based falsification methods can perform automated bug-finding
- Requirement engineering an ongoing challenges for CPS
  - ST-Lib intended to support V&V activities for CPS applications
    - Application results are promising but many challenges revealed
  - Need to think about improved methods to capture designer intentions

MBD TOYOTA TECHNICAL CENTER

Other CPS Test & Verification Challenges

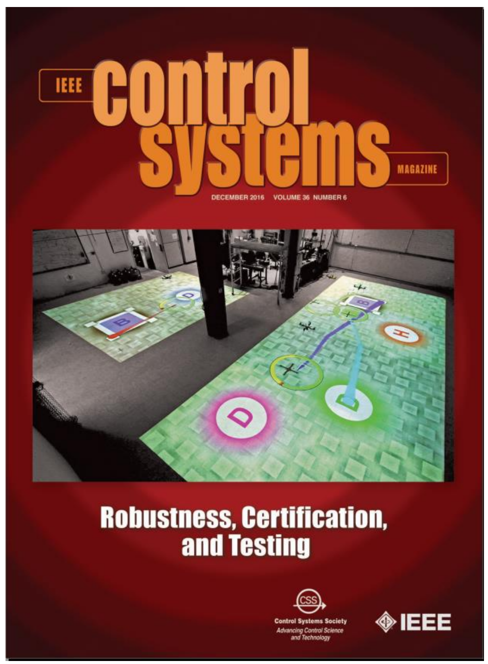
- Building appropriate models
  - Model creation is time consuming and error-prone
  - How to automate model construction
  - How to check model accuracy
- Verification techniques
  - Scaling model-checking/theorem proving techniques for CPS
  - Dealing with black-box models
- Advanced testing/evaluation techniques
  - Continue to develop new/better simulation-based falsification approaches
  - Need automated testing approaches for calibration
- Control synthesis
  - Can we create safe-by-construction control designs?
- Systems based on machine learning/AI
  - Lots of immediate applications: autonomous cars, advanced driver assist
  - Not clear how to test/certify

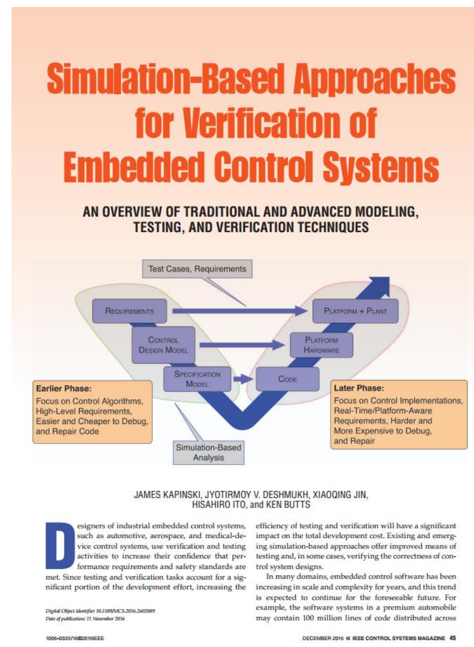
Super hot topic!!

MBD TOYOTA TECHNICAL CENTER

Thanks for your attention!

- Thanks to…
  - Toyota collaborators: Hisahiro Ito, Ken Butts, Jyotirmoy Deshmukh, Xiaoqing Jin
  - Academic collaborators: Alexandre Donzé, Tommaso Dreossi, Sanjit Seshia (**UC Berkeley**), Ayca Balkan, Paulo Tabuada (**UCLA**), Georgios Fainekos (**Arizona State University**), Nikos Arechiga, (**Toyota Info Technology Center**), Aditya Zutshi, Sriram Sankaranarayanan (**CU Boulder**)
- Special thanks to [Necmiye Ozay](#) for the invitation!
- Questions? Comments?





Simulation-Based Approaches for Verification of Embedded Control Systems

AN OVERVIEW OF TRADITIONAL AND ADVANCED MODELING, TESTING, AND VERIFICATION TECHNIQUES

JAMES KAPINSKI, JYOTIRMOY V. DESHMUKH, XIAOQING JIN, HISAHIRO ITO, and KEN BUTTS

Designers of industrial embedded control systems, such as automotive, aerospace, and medical-device control systems, use verification and testing activities to increase their confidence that performance requirements and safety standards are met. Since testing and verification tasks account for a significant portion of the development effort, increasing the efficiency of testing and verification will have a significant impact on the total development cost. Existing and emerging simulation-based approaches offer improved means of testing and, in some cases, verifying the correctness of control system designs.

In many domains, embedded control software has been increasing in scale and complexity for years, and this trend is expected to continue for the foreseeable future. For example, the software systems in a premium automobile may contain 100 million lines of code distributed across

Digital Object Identifier 10.1109/MCS.2016.2602089
Date of publication: 11 November 2016

1066-033X/16©2016IEEE

DECEMBER 2016 « IEEE CONTROL SYSTEMS MAGAZINE 45

Please read about these and related issues in our article in the Dec. 2016 issue of IEEE Control System Magazine