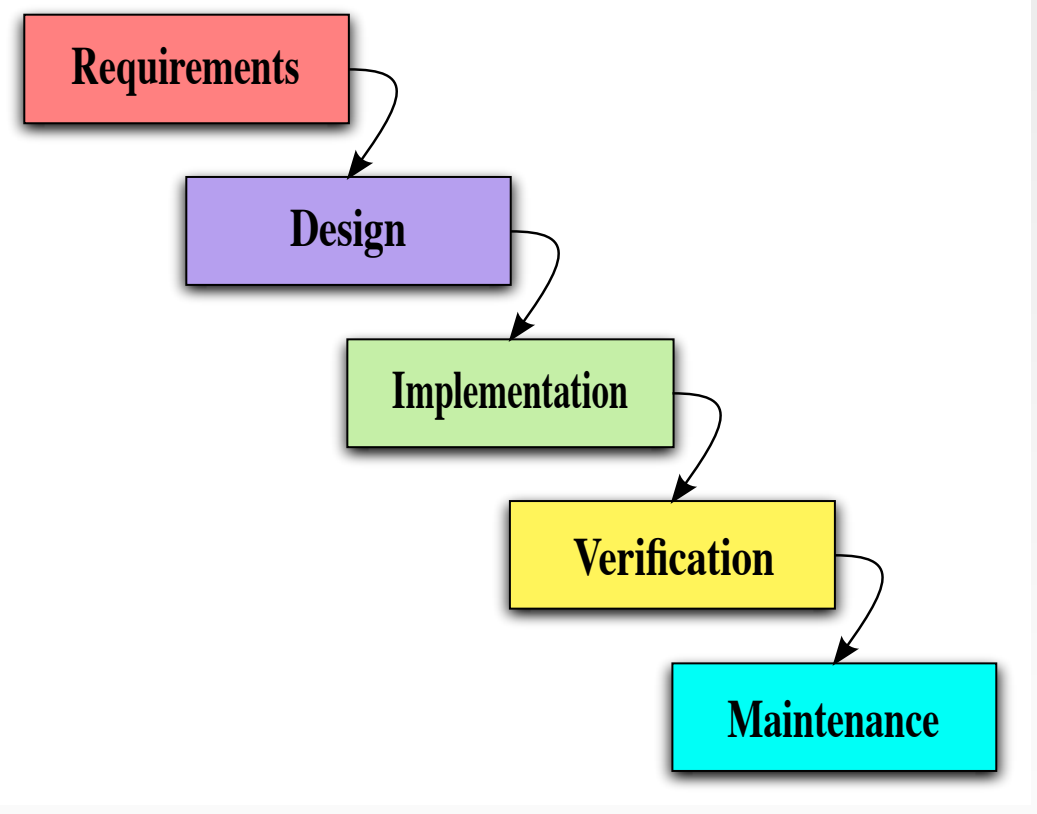
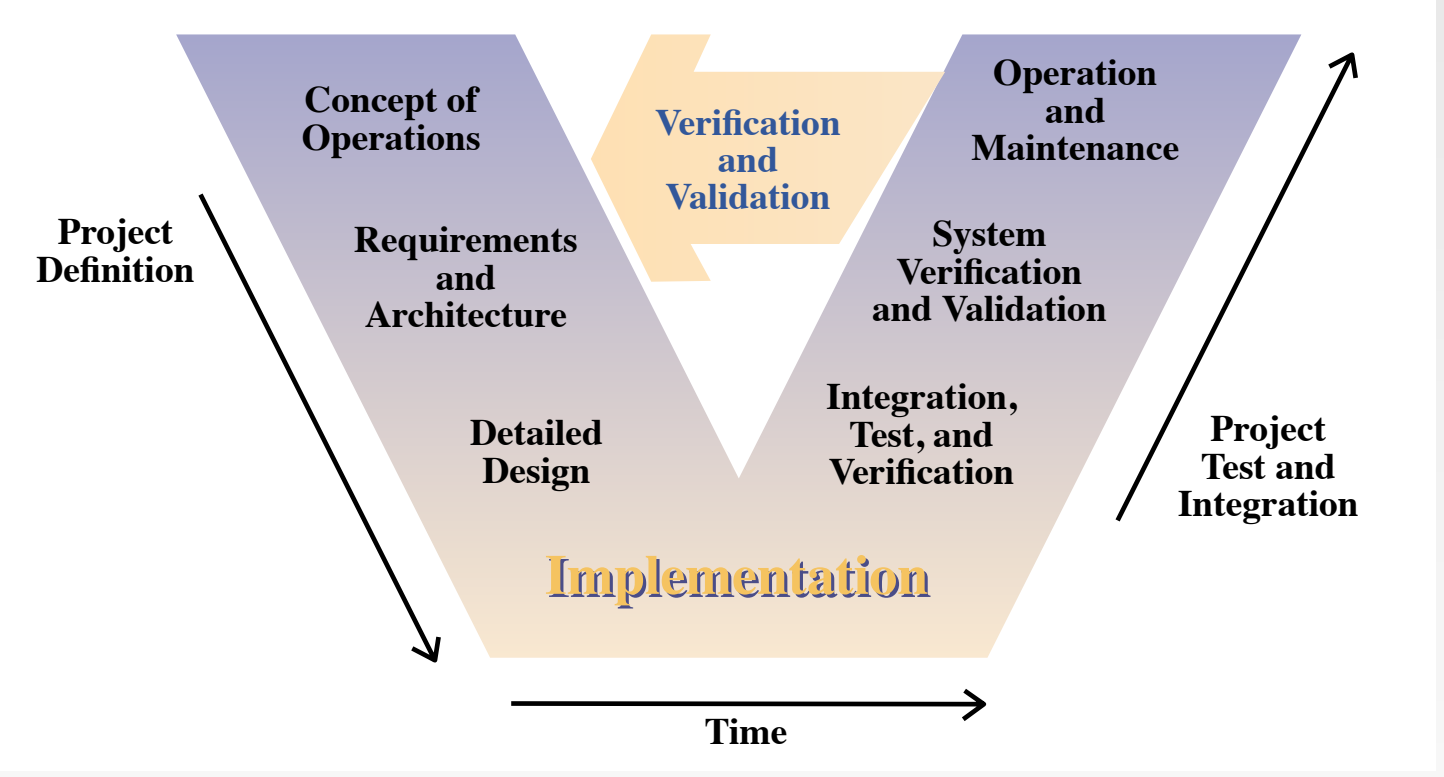
Automatic Code Generation

Embedded Control Systems
Fall 2012

Software Development: "Waterfall Model"

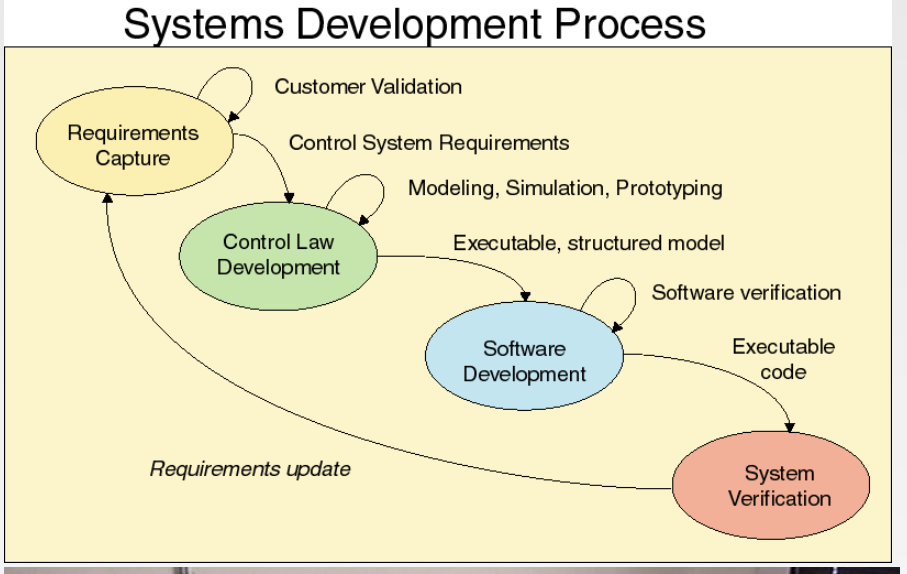


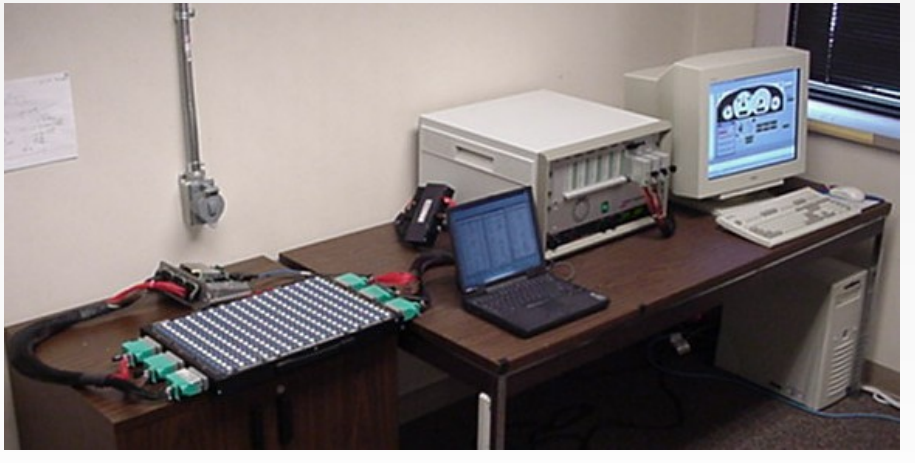
Software Development: "V diagram"



Model-based SW Engineering: Process

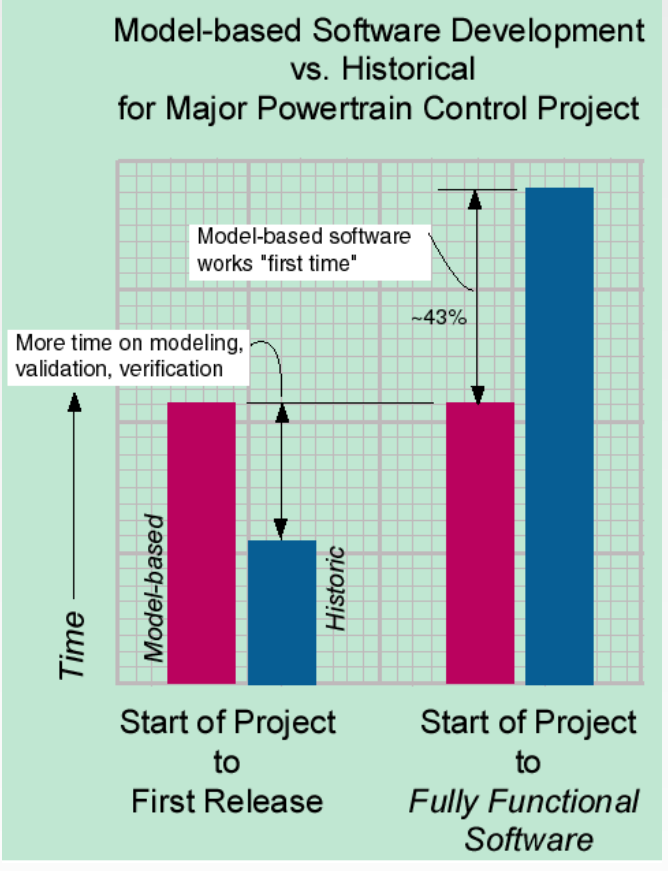
- Control law validated by simulation and rapid prototyping
- Executable software specification (algorithm model)
- Software Development (or automatic generation)
- HIL verification of embedded implementation
- Models permit V&V at every step of the process from requirements to implemented code.

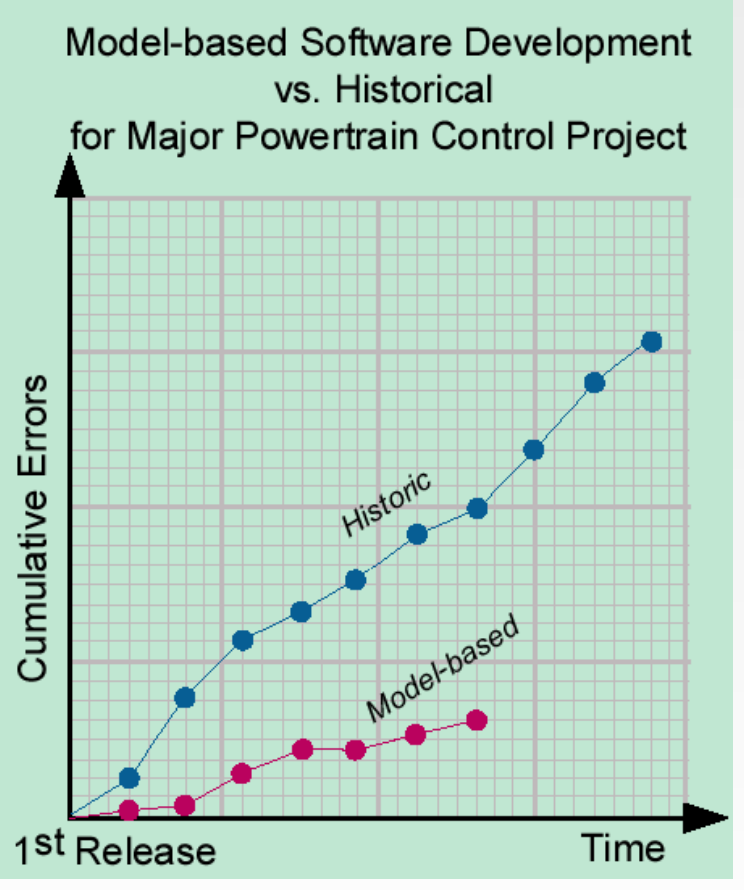




Typical Requirements Analysis

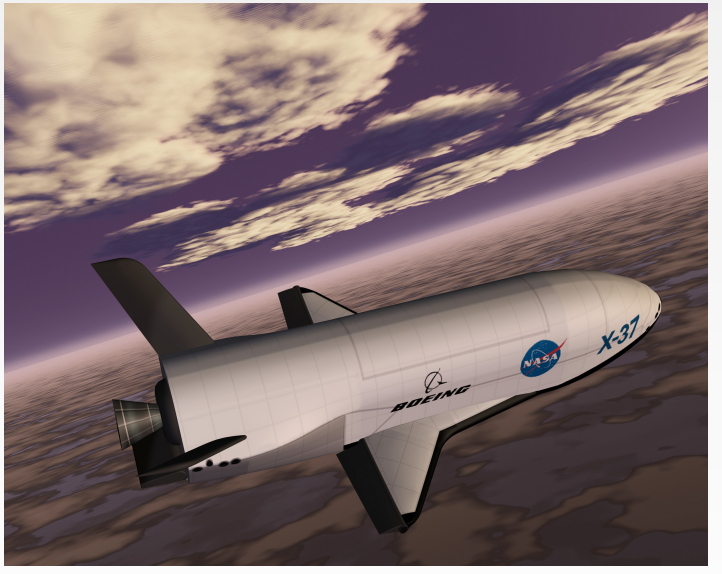
Model-based Software Engineering: Statistics





Examples: Automatic Code Generation

- Deep Space 1
- Deep Impact
- Pluto-Kuiper Belt
- MER
- MRO
- MESSENGER
- X-37
- DAWN



Report on the Utility of the MAAB Style Guide for V&V/IV&V of NASA Simulink/Stateflow Models, NASA 2004

Advantages of Automatic Code Generation (dSPACE website with editorial comments in red)

- Less time-consuming and error-prone than hand coding
- Shorter development times, often reduced by more than 40%
- Model and C code always consistent (really?)
- Uniform standard for coding (really?)
- No implementation errors (assuming the model is correct!)
- Code documentation always up-to-date (really?)

Disadvantages of Automatic Code Generation

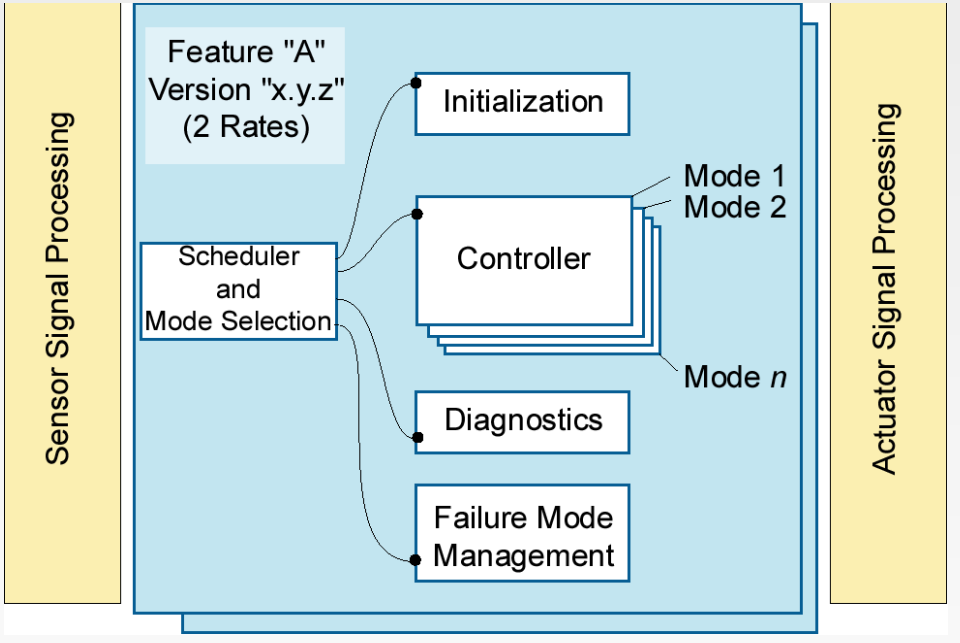
- Code size (not as big a problem as it once was)
- Integration with legacy code
- Consistency between model and code (temptation to tweek the code rather than revise the model and re-generate)

Automatic Code Generation Tools

- **dSPACE Targetlink**
  - Code generation from Simulink/Stateflow
  - Extended Targetlink block set for fixed-point code generation and implementation specific information
  - http://www.dspaceinc.com/ww/en/inc/home.cfm
- **ETAS ASCET**
  - Code generation from ETAS graphical modeling environment
  - New product supports translation from Simulink/Stateflow
  - http://en.etasgroup.com/index.shtml
- **National Instruments LabVIEW**
  - FPGA code generation from LabVIEW “Virtual Instrument” modeling environment
  - http://www.ni.com/
- **The MathWorks**
  - Code generation from Simulink/Stateflow
  - Real-time Workshop (RTW) and RTW with Embedded Coder
  - http://www.mathworks.com/

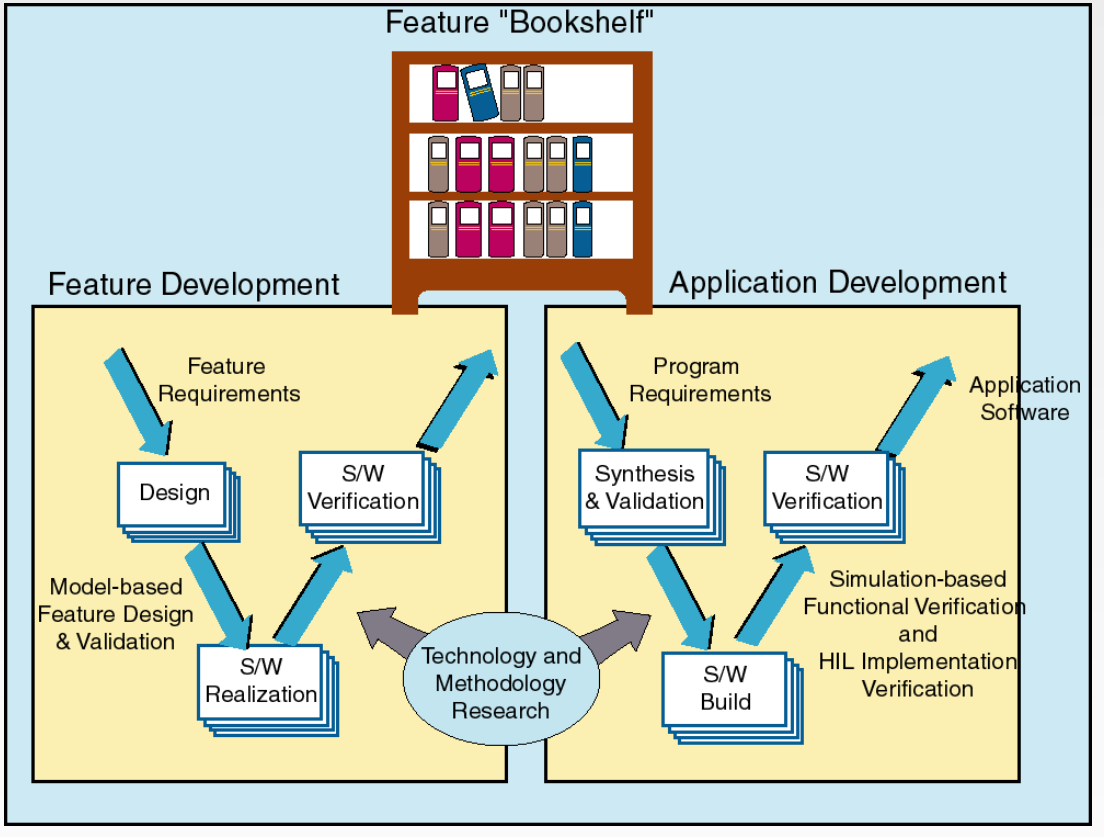
Real Embedded Software

- Large, complex, developed by many people, integrated at the end, and expected to work.
- Typical automotive control "feature"
  - Much more than just the control law
  - Multiple versions address program-to-program variability
  - Average feature has
    - 1-2 execution contexts
    - 20 inputs
    - 14 outputs
- ~60-100 features per vehicle with more than 2000 connections among features



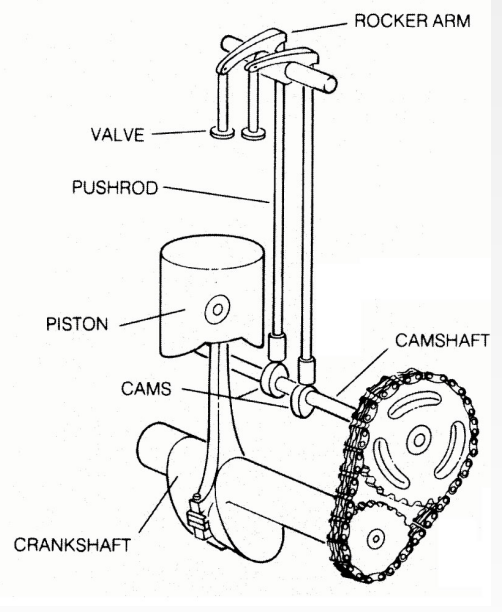
Model-based Software Engineering

- Modeling environment requires
  - Flexible, interchangeable and reusable model components
  - Seamless process for component “plug-and-play”
  - Data and complexity management
  - Systems and software analysis tools



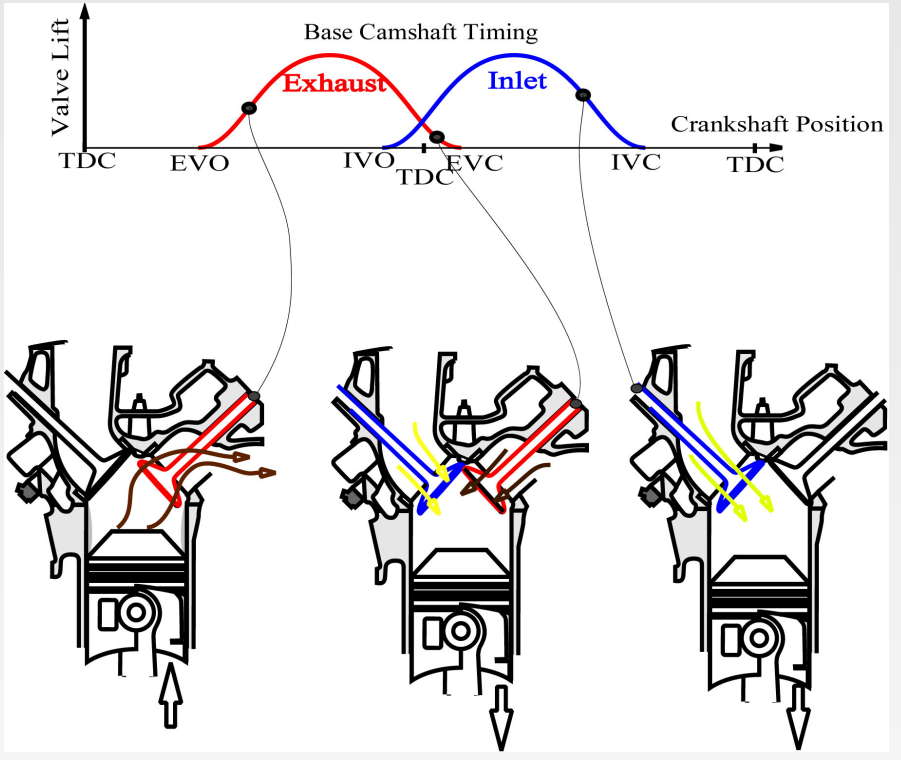
Conventional Cam Timing

- Conventional engine
  - Camshaft linked mechanically to crankshaft
  - Hence valve motion is fixed
  - Tradeoffs between fuel economy, combustion stability, maximum torque performance



Variable Cam Timing

- VCT/VVT/camless engine
  - Phase valve opening and closing with respect to firing event



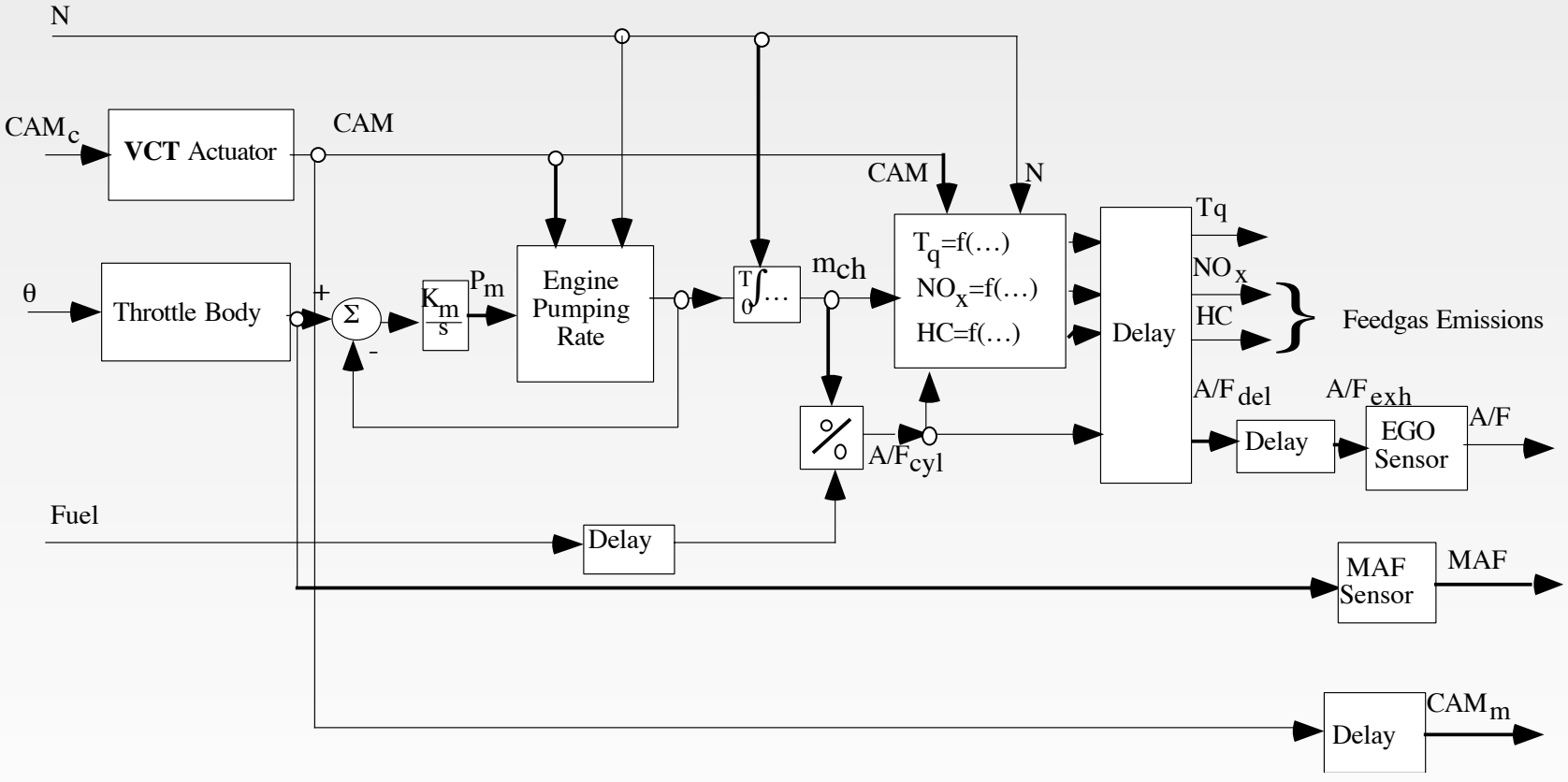
Potential Benefits

- Eliminate pumping losses (fuel economy)
- Increase burned gas fraction (NOx emissions)
- Reduce the effects of manifold filling dynamics (driveability)

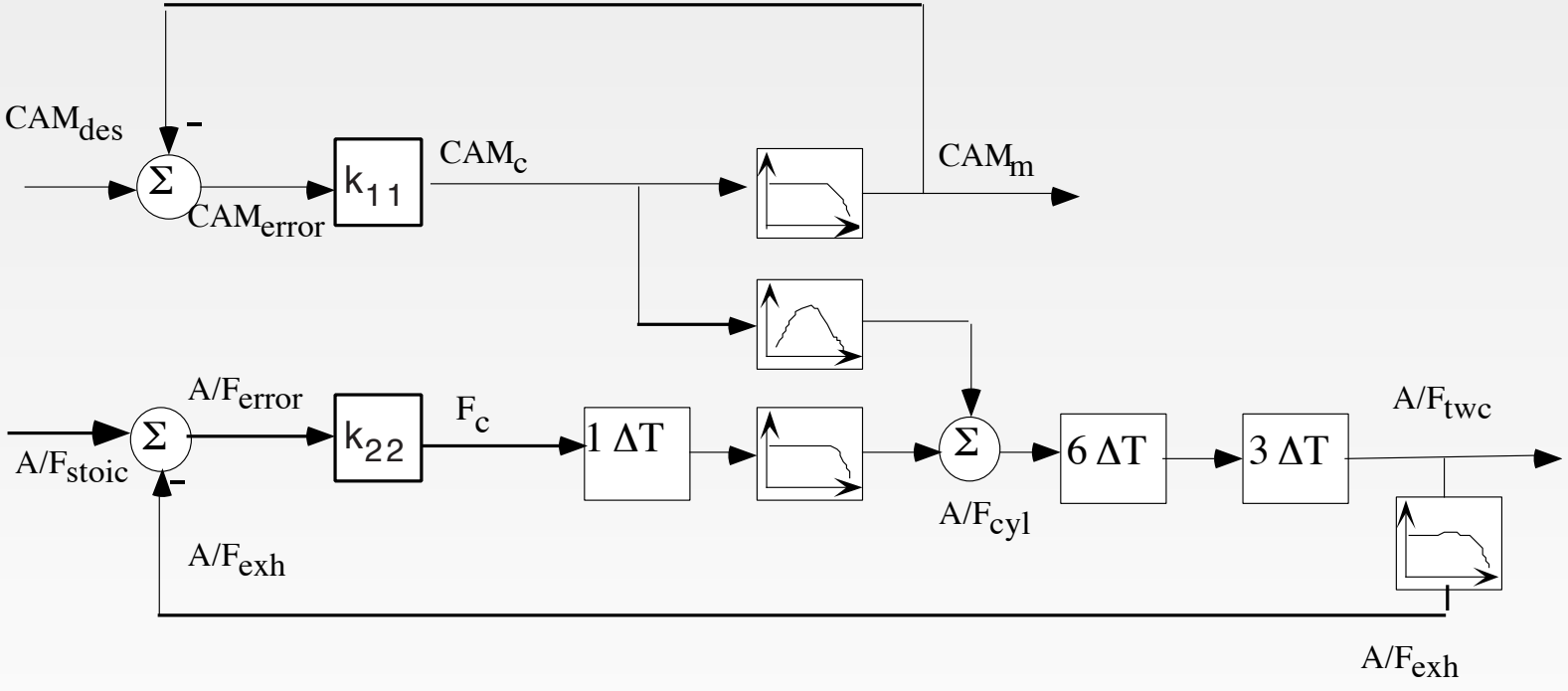
Issue

- the cam activity must not adversely affect other measures of engine performance

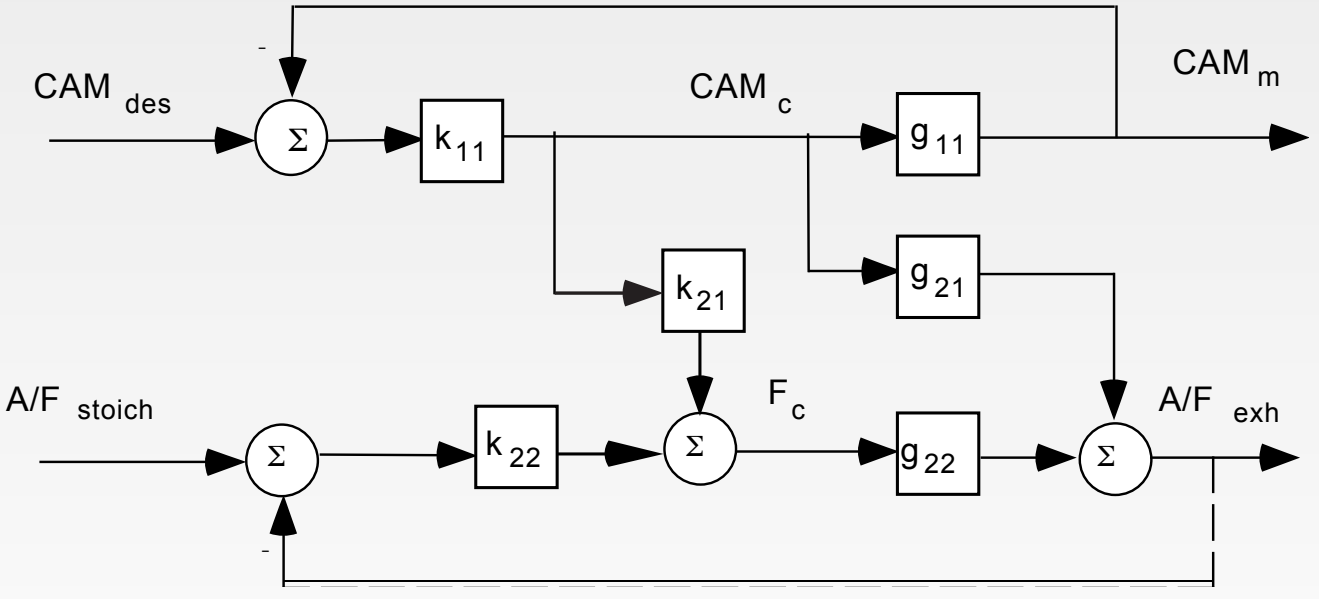
VCT Diagram


N
CAM_c
VCT Actuator
CAM
CAM
N
θ
Throttle Body
+
Σ
-
K_m/s
P_m
Engine Pumping Rate
T ∫ 0 …
m_ch
T_q=f(…)
NO_x=f(…)
HC=f(…)
Delay
Tq
NO_x
HC
Feedgas Emissions
A/F_del
A/F_exh
Delay
EGO Sensor
A/F
%
A/F_cyl
Fuel
Delay
MAF Sensor
MAF
Delay
CAM_m

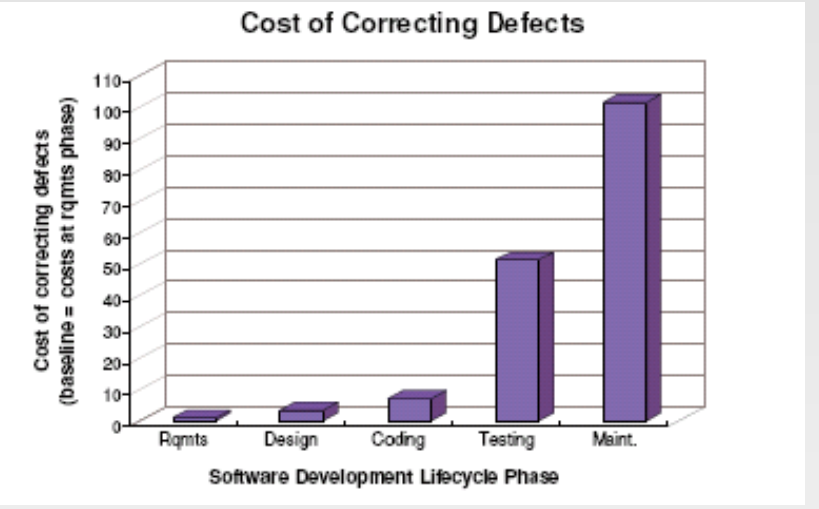
VCT: Time delay



VCT: Control solution



Style Matters



- Models must be clear, readable, modular, documented and precise
- Automatic code generation does not eliminate human error – just moves it higher in the process
- Order of execution, execution context, data types – must be specified in the model!
- Naming conventions, data scoping, annotations and comments, …

Reference: "Style Matters - Applying the lessons from the software industry to Autocoding with Simulink" by Peter Gilhead, Ricardo Tarragon

Hatley-Pirbhai Model Methodology

- Simulink diagrams model data flow; Stateflow diagrams model control flow
- Process specifications (P-specs) modeled using Simulink blocks and/or Stateflow diagrams, depending on the nature of the algorithm
- Control specifications (C-specs) are modeled using Stateflow
- One Simulink subsystem per execution context (10ms, 100ms, etc.)

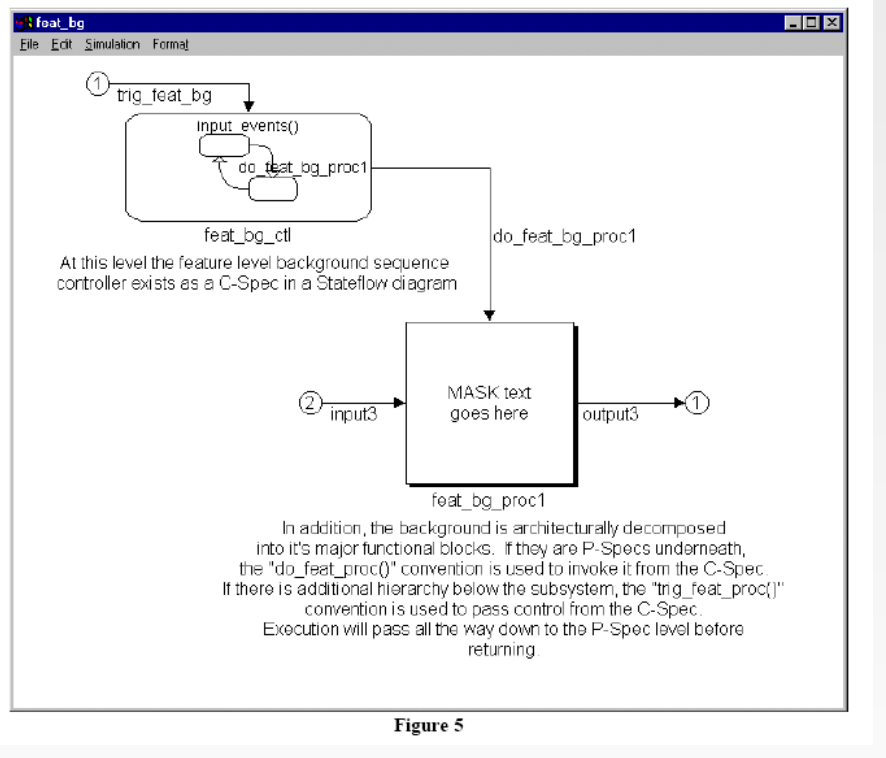


Figure 5

Reference: D. J. Hatley and I. A. Pirbhai, Strategies for Real Time System Specification. New York: Dorset House, 1988.

Mathworks Automotive Advisory Board

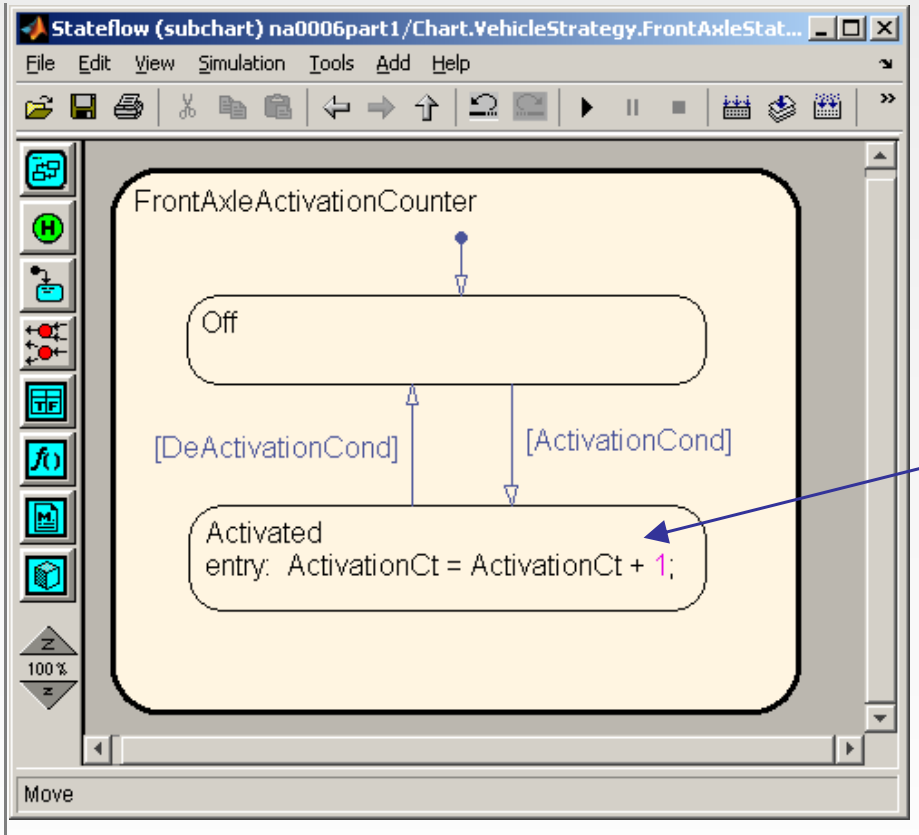
- Originally established to coordinate feature requests from several key customers in the automotive industry.
- Inaugural meeting in July 1998 involved Ford, Daimler Benz, and Toyota.
- Meetings now involve many of the major automotive OEMs and suppliers.
- Focus on the usage and enhancements of MathWorks controls, simulation, and code generation products including Simulink, Stateflow, and Embedded Coder.
- MAAB Control Algorithm Modeling Guidelines Using Matlab, Simulink, Stateflow
  - http://www.mathworks.com/automotive/standards/maab.html

MAAB Style Guide: Simulink or Stateflow?

- The choice of whether to use Simulink or Stateflow to model a given portion of the control algorithm functionality should be driven by the nature of the behavior being modeled.
  - If the function primarily involves complicated logical operations, Stateflow should be used.
  - Stateflow should be used to implement modal logic – where the control function to be performed at the current time depends on a combination of *past and present logical conditions.*
- If the function primarily involves numerical operations, Simulink should be used.

Stateflow w/ simple math

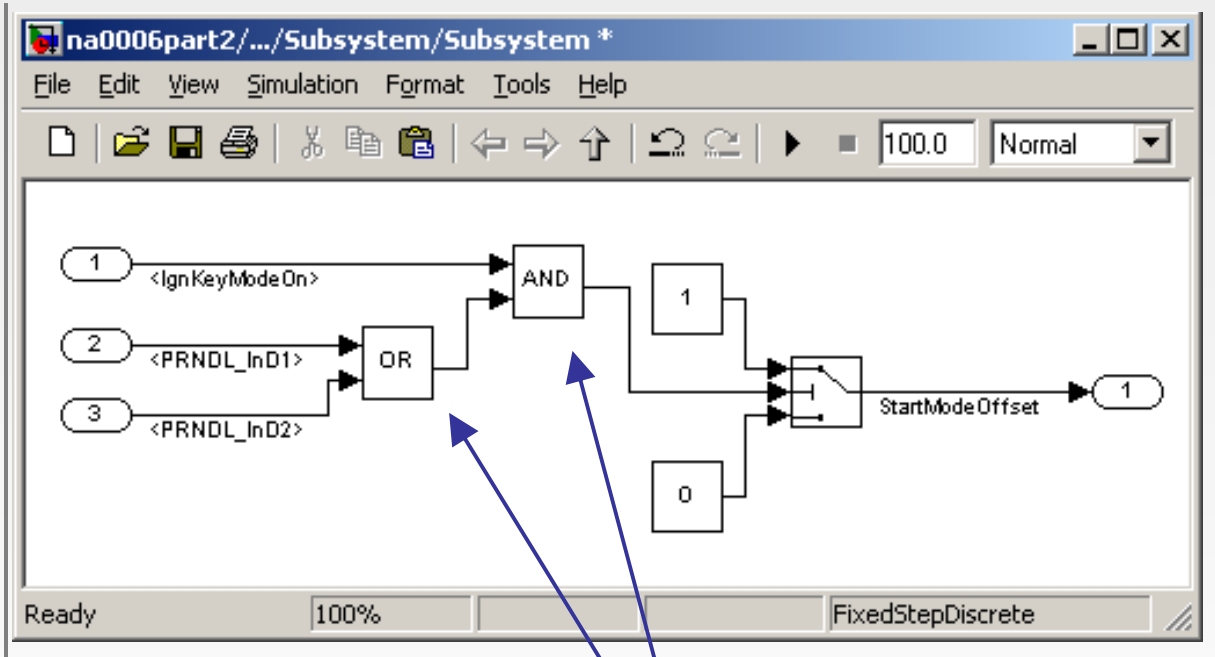
- If the primary nature of the function is logical, but some simple numerical calculations are done to support the logic, it is preferable to implement the simple numerical functions using the Stateflow action language.



Embedded simple math operation

Simulink w/ simple logic

- If the primary nature of the function is numerical, but some simple logical operations are done to support the arithmetic, it is preferable to implement the simple logical functions within Simulink.



Embedded simple logic operations