A-L-V
Automating the Left Side of the V

Presenters, Ford Chassis Controls:
Nate Rolfes   John Broderick   Jeff Cotter

With Ford MBSE Tools & Methods:
Kyu Sohn   Yuping Jiang   Tapan Kasaragod

May 2, 2018
Modern automotive vehicles contain between 100-150 million lines of code across 30-80 networked ECU’s with up to 30,000 physical parts, making them one of the most complex engineered systems in the modern world.

SINGLE LINES OF CODE (MILLIONS)

Hubble Telescope: 2
US Military Drone: 3.4
Mars Curiosity Rover: 5
Android OS: 12
Boeing 787: 13.7
F-35 Fighter: 23.5
Windows 7: 39.5
Microsoft Office: 44.3
Large Hadron Collider: 50
Facebook: 61
Mac OS X: 84.6
Ford F-150: 100
Google Internet Services: 150

Sources:
https://informationisbeautiful.net/visualizations/million-lines-of-code/
THE SOURCE OF SOFTWARE ISSUES

“The IV&V Program documented 10,677 software artifact defects on 22 NASA projects in 2007...The IV&V Program analyzed the defects sorting them by severity and type of defect.”

Requirements are the leading source of software defects. System requirements are the leading source of requirement defects.
41% of Software issues found during development of the 2016MY F-150 Pro Trailer Backup Assist Feature were related to the requirements, and 38% of all software issues were system-related.

“Requirement models ... capture the functional requirement in a clear and executable manner that can be used to evaluate the interaction and compatibility of requirements from disparate sources”

Lee/Friedman, The Mathworks, SAE 2013-01-2237
“Requirement models ... capture the functional requirement in a clear and executable manner that can be used to evaluate the interaction and compatibility of requirements from disparate sources”

Lee/Friedman, The Mathworks, SAE 2013-01-2237
The collection of requirement models can be collected into a larger system model which enables up-front design and testing to ensure the system behaves as intended.
DISTRIBUTED SYSTEM MODEL TYPES

Model-in-the-Loop (MIL)

Software-in-the-Loop (SIL)

Hardware-in-the-Loop (HIL)

Simulate!

Code Gen

Code Gen

Code Build for Target

Upload to Hardware

Power and Connect Hardware

Code Wrapper

Hex file
Maximize design & testing in the virtual world!

MIL & SIL issues should not be found in real world testing!
THE SYSTEM V

- Concept Design
- Architecture & Requirement Development
- Iterative Design
- Module Code Integration & Verification
- System Integration & Verification
- Product Integration & Verification
- Product Complete

Models (MIL)

Software (SIL)

Hardware (HIL) & Vehicle

Development Time

Requirements Released

Implementation To Requirements

Models (MIL)
THE REAL WORLD SYSTEM V

80 weeks of Development Time prior to Software availability!

MIL can be used as the oracle to check SIL and HIL Testing

Weeks before Production
180 140 100 80 40 0
Kickoff Production

Software Available

Multi ECU HIL
Single ECU HIL
SIL Testing
MIL TO HIL SWITCH

Instrument Cluster Hardware (HIL)

Vehicle Network Toolbox

Instrument Cluster Requirement Model (MIL)

CAN Gateway (MIL)
DISTRIBUTED SYSTEM TESTING
DISTRIBUTED SYSTEM TESTING
DISTRIBUTED FEATURE SIMULATOR

Network Interface Builder

Vehicle Interface Builder

Qt HMI Integration
Manually creating a Simulink model from a network architecture file can take a lot of time and effort. Each network model is different depending on feature, vehicle, and build configuration.

Building the model also requires the time and effort of an engineer who is skilled in Simulink and the Vehicle Network Toolbox in order to robustly build and debug that the model is working correctly.
NETWORK INTERFACE BUILDER
It takes 4 minutes to manually model a CAN Message in Simulink… and would take 14 hours to build this model by hand!

Feature Model Example

15 ECUs
13,801 Simulink Blocks
218 CAN Messages
1,397 CAN Signals
By automating the Feature Model build process, we open up new possibilities to improve design, testing, and validation results.

New System models can be generated at each Program Milestone on demand, or whenever new DBC files are released.

Eliminates the bottleneck of needing an expert at Simulink to generate feature models and update them.

Eliminates errors in manual build and the tedious updates of the model when changes to network message or signal designs are changed.
Testing the network ECU model helps to identify and resolve issues with functional logic, state machines, and interfaces.

However, for more realistic dynamic testing of the control systems it is helpful to connect a vehicle model simulator which can provide realistic virtual feedback for sensors and actuators that the functional ECU’s control. Several “out of the box” simulators exist on the market, such as CarSim, CarMaker, and the Vehicle Dynamics Blockset.

Manually setting up the interface to a vehicle model simulator is tedious and error-prone as it can involve mapping up to hundreds of actuation and sensing signals between the ECU controls model and the Vehicle Model.

It also requires an engineer who has expertise in both Simulink and the Vehicle Model tool.

Developing a way to automatically generate this interface is a key efficiency step to eliminate time
VEHICLE INTERFACE BUILDER

CAN Interface
- Defined by vehicle program in DBC file
- Minimal updates only when DBC files updated
- Completely modeled in Simulink

Vehicle Model Interface
- ECU-dependent interfaces
- Model interfaces may change based on model fidelity
- Must accommodate different 3rd Party vehicle models

Vehicle model interface must be modular and flexible
## Example Specification (Subset of Brake ECU Interface)

<table>
<thead>
<tr>
<th>Input/Output</th>
<th>ECU Signal Name</th>
<th>System Signal Name</th>
<th>Vehicle Model (VM) Signal Name</th>
<th>Unit Gain (ECU)</th>
<th>Unit Gain (VM)</th>
<th>VM Specific Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>WheelSpeed_FL</td>
<td>W_WhlFl</td>
<td>AVY_L1</td>
<td>1</td>
<td>2π/60</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>BrakeTorqueFL</td>
<td>Tq_BrkFL</td>
<td>IMP_MYBK_L1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**Diagram:**
- **Vehicle Model**
  - Plant Outputs
    - Plant Receive
      - ECU Library
  - Plant Inputs
    - Plant Transmit

**Legend:**
- VM Signal Name
- System Signal Name
- ECU Signal Name
VEHICLE INTERFACE BUILDER

System to Controller conversion

Controller to System conversion

Vehicle Model to System Conversion

Vehicle Model Conversion

All Signal Routing and Gains automatically generated from specification files
• Most features have “Driver in the Loop” test scenarios
• HMI simulators help improve the quality and robustness of a feature
• The more realistic the HMI displays and simulators can be made, the better testing results can be achieved.
• Developing high quality HMI displays and interactions is tedious and is not a typical skillset of a controls or simulation engineer.
• Integrating HMI systems such as Qt with the Distributed Feature Simulator helps overcome this.
QT HMI INTEGRATION

- Connect ECU’s to HMI Model for Vehicle Simulator Experience
- Co-simulation with Qt model over Virtual CAN or physical device (e.g. Raspberry Pi)
QT HMI INTEGRATION

QT HMI Model and Simulink Integration

- Virtual CAN / Physical CAN Communication
- QT C++ Reads/Writes CAN Messages
- QT C++ Sets QML Properties
QT HMI INTEGRATION

- Re-use of work already being done by HMI teams
- Enables co-simulation of production-quality HMI development tools with production-quality controls and software development tools
- Saves controls engineer time of having to develop a Simulink replication of already existing HMI models.
A-L-V RESULTS

<table>
<thead>
<tr>
<th>Feature</th>
<th>Initial Build Hours</th>
<th>Milestone 1 Update Hours</th>
<th>Milestone 2 Update Hours</th>
<th>Milestone 3 Update Hours</th>
<th>Milestone 4 Update Hours</th>
<th>Total Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Interface</td>
<td>15</td>
<td>10</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>38</td>
</tr>
<tr>
<td>Vehicle Interface</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>22</td>
</tr>
<tr>
<td>HMI Integration</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>29</strong></td>
<td><strong>16</strong></td>
<td><strong>11</strong></td>
<td><strong>11</strong></td>
<td><strong>9</strong></td>
<td><strong>76</strong></td>
</tr>
</tbody>
</table>

- 76 engineering hours (9 days) saved per feature model
- For a program concurrently developing 10 features, this equates to 90 days of saved engineering time!
- In addition to saving time, the automated processes eliminate modeling mistakes
- Lastly, the automated methods frees up the time of valuable simulation engineers to focus on simulating and testing to improve our quality and robustness rather than tedious model-building tasks.
“The problem is that we are attempting to build systems that are beyond our ability to intellectually manage.”

– Nancy Leveson (MIT), The Coming Software Apocalypse (The Atlantic, September 2017)
SPECIAL THANKS

THANK YOU FOR YOUR ATTENTION 😊

The presenters wish to extend a special thank you to their management and colleagues as well as project services support from the Mathworks listed below

**Ford Motor Company**

Robert ter Waarbeek  Nick Adams  
Dr. Darrel Recker  Dave Messih  
Kathi Dobies

**The Mathworks**

Scott Furry  John Boyd