VIRTUAL VEHICLE built in MINUTES instead of MONTHS
AGENDA

- Why & What is FASST
  - Robert ter Waarbeek

- How does it work
  - Raghu Baskaran

- Creating full Vehicle Simulation
  - Steven Foster

- Scaling to production
  - Nick Adams

- Inner Sourcing for Model Development
  - Nate Rolfes

- Closing Statement
  - Robert ter Waarbeek
WHAT IS FASST?
WHY DO WE NEED IT?

ROBERT TER WAARBEEK
Software and digital systems provide tremendous power in building complex systems not previously possible. But this increase in power comes with a price – large software systems are fiendishly difficult to get correct. The difficulty of building such software is often underestimated by engineers.”

– Nancy Leveson, Professor of Aeronautics and Astronautics at MIT
Widely recognized as a preeminent expert in system and software safety

Automotive Vehicles are extremely complex mass produced mechatronics systems with rising complexity it is essential to detect system issues early in development.
THE CHALLENGE OF FULL VEHICLE SIMULATION

For virtual development of distributed systems all teams have to work together.

ADAS Feature
EESE
Matlab 2012a 32bit
In-house + Supplier C

Steering controls
Chassis
Matlab 2011b 32bit + target link
Supplier A + In-house

Vehicle Dynamics Models
VehDyn CAE
ADAMS
converted into:
CarSim
Dspace ASM
IPG-Carmaker
Mathworks-VDBS

Powertrain Models
Powertrain
Matlab 2015b 64bit + MBD
In-house

Brake Controls
Chassis
Matlab 2014b 32bit
Supplier B + In-house

Status Q1-2017
Simulation
“system” debugging for simulation purpose

Debug Models (computational issues)

Build Virtual Vehicle

Trace and Collect Component Models

WHERE ARE CAE ENGINEERS SPENDING THERE TIME
Modeling and Simulation
fully integrated into carline development

Debug Models (computational issues)

Build Virtual Vehicle

Trace and Collect Component Models

Initial State

Desired State

FASST is an enabler to work more effective
FORD AUTOMATED SYSTEM SIMULATION TOOLCHAIN (FASST)

GitHub

40 million+ Global Users
Ford: 10,000+ Users

Most widely used source control management tool

git

MATLAB® & SIMULINK®

3 million+ Global Users / 4,500 Employees / 31 Offices Globally
Ford: 7,000+ Matlab / 4,000+ Simulink Users

A Smart Cross Organizational team

500+ Members / Passive Users
100+ Active Users
~30 Members on “DevOps” Team

FASST leverages modern tools & standard
FASST: BUILDING THE SKELETON MODEL

Vehicle controls architecture

Bill Of Models

Skeleton/System Model

MathWorks

AUTOMOTIVE CONFERENCE 2020
FASST VIRTUAL VEHICLE BUILD: CREATING A VEHICLE MODEL

Populate ECU contents from functional software model developers

Model Repository

Skeleton/System Model

Component Models

Vehicle models
FASST: VIRTUAL DEVELOPMENT

There is no single virtual vehicle

DVM, DVP with metrics

or

Exploratory testing

Test Plan

Virtual development
FORD AUTOMATED SYSTEM SIMULATION TOOLCHAIN (FASST)

Vehicle controls architecture

Feature model BOM (Bill Of Models)

FASST

Build system/skeleton model & Include components

Virtual development & Verification

FASST reduced virtual vehicle build from months into minutes
HOW DOES IT WORK?

THE MECHANICS OF BUILDING A FASST MODEL

RAGHU BASKARAN
WHAT DO WE NEED TO BUILD A FASST MODEL?

- GitHub
  - Cloud-Based Distributed Version Control
- ECU Models & Components
- Vehicle Plant Models
- Model BOM
- ECU Architecture (DBC)
- Plant Interface Sheet
## IT ALL STARTS WITH THE BILL OF MODELS

<table>
<thead>
<tr>
<th>Program_Name</th>
<th>Feature Name</th>
<th>Variant Name</th>
<th>Model Parts</th>
<th>GitHub Organization and Repository</th>
<th>Branch/Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ADAS</td>
<td></td>
<td>BRAKES</td>
<td>FASST/BRAKES</td>
<td>BehvM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CAMERA</td>
<td>FASST/CAMERA</td>
<td>BehvM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BODY</td>
<td>FASST/BODY</td>
<td>SkeM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>STEERING</td>
<td>FASST/STEERING</td>
<td>ReqtM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FEATURE</td>
<td>FASST/FEATURE</td>
<td>Feature1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DISPLAY</td>
<td>FASST/DISPLAY</td>
<td>Display1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DRIVER</td>
<td>FASST/Driver</td>
<td>Driver1</td>
</tr>
<tr>
<td>Vehicle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CarSim</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CarMaker</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VDBS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BehvM</td>
</tr>
<tr>
<td>Ford_PowerTrain</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADAMS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Additional sections (not shown) for:
- Build Options
- System Model
- Test Procedures
- Optional Tools
- Documentation
- Test Results
- Miscellaneous

Program Name
Feature Name
Variant Name
ECU Component Models
Location in GitHub
Model Fidelity or Release Tag
Vehicle Dynamics
Model Selection
Network / ECU Architecture
THE FASST “ONE CLICK” SYSTEM MODEL BUILD

FASST is an automation wrapper for several linked tools that builds the full system vehicle model from components in the GitHub cloud.
FASST generates an ECU framework model that can accept “plug and play” components to capture as many use cases as possible.
CREATING FULL VEHICLE SIMULATION:
CONNECTING SYSTEM COMPONENTS TO THE VEHICLE MODEL

STEVEN FOSTER
Why Support Multiple Vehicle Plant Models?

Simulation Speed

Licensing Cost / Availability

User Experience

Department Preference

Model Fidelity

Vehicle Plant Option Flexibility

VEHICLE PLANT MODEL FLEXIBILITY IS CRITICAL
THE FASST PLANT VEHICLE BUILD PROCESS

The Plant Interface Builder automates the connections within FASST.
THE VEHICLE PLANT TO ECU CONNECTION INTERFACE

ECU-to-Vehicle Interface Sheets

<table>
<thead>
<tr>
<th>Interface</th>
<th>GPSDB Name</th>
<th>Controller Name</th>
<th>Unit Gain (GPSDB -&gt; Controller)</th>
<th>Unit Gain (Controller -&gt; GPSDB)</th>
<th>VDBS Name</th>
<th>Unit Gain (GPSDB -&gt; VDBS)</th>
<th>Unit Gain (VDBS -&gt; GPSDB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receive from Plant</td>
<td>An_SteerWhlPosn_rad</td>
<td>Steer_Ang</td>
<td>$180/\pi$ (rad to deg)</td>
<td>$\pi/180$ (rad to deg)</td>
<td>AngIn</td>
<td>1 (rad to rad)</td>
<td>1 (rad to rad)</td>
</tr>
<tr>
<td>Transmit to Plant</td>
<td>Tq_SteerColumnTqAsst_Nm</td>
<td>Column_Torque</td>
<td>1000 (Nm to Nm)</td>
<td>1/1000 (Nmm to Nm)</td>
<td>TrqIn</td>
<td>1 (Nm to Nm)</td>
<td>1 (Nm to Nm)</td>
</tr>
</tbody>
</table>

Global Plant Signal Database

ECU Models & Components

GPSDB—GLOBAL PLANT SIGNAL DATABASE
VDBS—VEHICLE DYNAMIC BLOCKSET

AUTOMOTIVE CONFERENCE 2020
SCALING TO PRODUCTION:
UNLEASHING THE POWER OF CONTINUOUS INTEGRATION (CI)

NICK ADAMS
CONTINUOUS INTEGRATION: JENKINS TO VALIDATE COMPONENTS

Component ECU Repos
- BRAKES
- CAMERA
- STEERING
- BODY

Jenkins

F-150 Brakes
Compile → Unit Tests

F-150 Camera
Compile → Unit Tests

F-150 Steering
Compile → Unit Tests

F-150 Body
Compile → Unit Tests

At least 10 carlines × 30 ECUs
Tests run nightly

250+ tested models are viewable from a browser
FASST Garage is the enabler for finding the right model based on the use case and user’s needs.
CONTINUOUS INTEGRATION: JENKINS TO AUTOMATE VEHICLE BUILDS

Bill of Models Library

Component ECU Repositories

ECU Network (DBC)

GitHub

Jenkins

F-150 Lane Assist
FASST > Compile > Unit Tests > Feature Tests

F-150 Park Assist
FASST > Compile > Unit Tests > Feature Tests

F-150 Trailer Assist
FASST > Compile > Unit Tests > Feature Tests

Dozens of BOMs are built and tested in parallel for a combined total of 40+ hours each day

One pipeline takes between 10 min to over an hour
INNER SOURCING FOR MODEL DEVELOPMENT

“All Models Are Wrong... Some are Useful”
- George E.P. Box

NATE ROLFES
**FASST AND THE POWER OF INNER SOURCING**

**Inner Source**

is the adoption of **open source software development** best practices and establishment of **open source culture within an enterprise**.

### Collaboration

Maximize the pool of engineering brainpower for advancing a project, meeting user needs, or finding and fixing bugs. **Never start from scratch, always build upon others work!**

### Communication

Transparent, self-documenting, and “searchable” problem solving and decision-making creates trust & alignment in the goals and **makes it easy for new users to get on-board and start contributing!**

### Egalitarian

Users are Developers & Developers are Users leads to a culture void of “politics” as recognition is inherently merit-based. Can work around organizational barriers and provide everyone the opportunity to influence the project direction and success!

### The “Plausible Promise”

“Your program can be crude, buggy, incomplete, and poorly documented. What it must not fail to do is (a) run, and (b) convince potential co-developers that it can be evolved into something really neat in the foreseeable future.”

*— Eric S. Raymond, The Cathedral and the Bazaar*

Establishing standard interfaces, terminology, and metrics around model types & capabilities is critical to gain traction for inner source
Evidence indicates that over 60% of System software issues emerge from these three phases!

**MODEL FIDELITY: SIMPLIFIED AND FUNCTIONAL MODELS**

**Skeleton Model**
- Interfaces (I/O) with Changeable Outputs (No behavior or logic)

**Requirement Model**
- Executable representation of functional design requirements and specifications

**Behavior Model**
- Design Intent Behavior

> "Descriptive, well structured, concise, unambiguous, readable, easy to understand."

> "Capture the functional requirement in a clear and executable manner."
> Lee & Friedman, Requirements Modeling & Automated Requirements-Based Test Generation (2014)

Evidence indicates that over 60% of System software issues emerge from these three phases!
Implementation Model / Production Code

Model-Based Implementation of the production code model. Typically only available for in-house model-based code. Difficult to obtain for supplier written code (use HIL instead).

System simulations which utilize Implementation Models can take dozens of minutes to compile and don’t simulate at real-time speeds!
Continuous Modeling Integration using scaled modeling fidelities is critical to ensure the systems work correctly the first time!
FASST “TO GO”: QUICK DEPLOYMENT TO SPEEDGOAT

FASST “To Go” Provides flexible options for offloading model components to real-time hardware as well as an easy method to pass protected binary files to breadboard teams and suppliers.
FASST toolchain...
... is developed in collaborative, modern, inner source and agile fashion, together with the MathWorks
... helps to detect system issues throughout the development
... reduced Virtual Vehicle build time from months to minutes
... the automated processes eliminate modeling mistakes
... in combination with CI enables scaling up modeling and simulation to enterprise level.

The challenge of “All models are wrong, but some of them are useful” will always stay
“Plug and Play” components are a critical key to success

FASST doesn’t solve all the issues,
but makes the daily life of an engineer more effective and enabled

Cross Organizational Collaboration
With special thanks to the entire FASST team:
THANK YOU

Ford