Developing Train Propulsion Controls using automatic Model Generation and automated Build and Test

Han Geerligs
System Designer
Vehicle Engineering
Strukton Rolling Stock
19 June 2018
• Strukton Rolling Stock develops traction converters from 100 kW up to 6 MW

• Vehicle Engineering develops control software for traction converters

• The application development team consists of approximately 10 engineers

• Embedded software is developed using Model Based Design and code generation

• Using MBD since 2000, developing Embedded Targets for Analog Devices DSP
Strukton Rolling Stock

Driverless vehicle (France)

5 MW traction (India)

Control board including DSP and FPGA

Hybrid EMU (UK)

Monorail (Kuala Lumpur)

Power Module driver board
1. Improve development using generated network interface models
2. Detect errors as they are introduced by automatic unit testing
3. Enable concurrent engineering by automated builds
Innovation Challenges and Achievements

CHALLENGES
• Developing interfaces repetitive, error prone and shared
• Build process slow and not repeatable
• Regression

ACHIEVEMENTS
• Interface models generated automatically
• The build process is scalable
• Prompt feedback on errors
How did we get there and leverage MathWorks

- Using the Simulink API to generate interface models
- MATLAB Unit test framework
- Use MATLAB command line interface
- Benefits include:
  - Decreasing application build time (by 50%)
  - All defined by code
  - Build and test scripts start running automatically

```matlab
add_block('simulink/Math Operations/Gain','vdp/Five','Gain','5')
```
Generating interface models

- Multiple network types (e.g. CAN, MVB, Ethernet)
- Using interface definition in Excel
• Packing and unpacking
• Timing semaphores
• Multiplexed messages
• Configurable output: ports or data stores
• Configurable naming conventions
• Configurable test points and fixations
• Optimise models to enable optimisation
IT landscape and workflows
Automate the build process

- Technology stack: Jenkins, Bitbucket, Jira and MATLAB/Simulink
- Web technology
- Build process is defined for Jenkins and MATLAB, stored in BitBucket
- Using Git and web-applications enables concurrent distributed developing
### Build server

#### Stage View

<table>
<thead>
<tr>
<th>Stage</th>
<th>Checkout</th>
<th>Cleanup</th>
<th>Build</th>
<th>Postbuild</th>
<th>CheckStyle</th>
<th>Plot</th>
<th>compare memory thresholds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10min 14s</td>
<td>114ms</td>
<td>21min 44s</td>
<td>2s</td>
<td>2s</td>
<td>496ms</td>
<td>3s</td>
</tr>
<tr>
<td>2</td>
<td>9min 27s</td>
<td>85ms</td>
<td>18min 33s</td>
<td>2s</td>
<td>2s</td>
<td>304ms</td>
<td>4s</td>
</tr>
<tr>
<td>3</td>
<td>9min 19s</td>
<td>96ms</td>
<td>32min 13s</td>
<td>3s</td>
<td>255ms</td>
<td>577ms</td>
<td>3s</td>
</tr>
<tr>
<td>4</td>
<td>10min 21s</td>
<td>110ms</td>
<td>32min 22s</td>
<td>8s</td>
<td>6s</td>
<td>740ms</td>
<td>5s</td>
</tr>
<tr>
<td>5</td>
<td>6min 48s</td>
<td>120ms</td>
<td>9min 39s</td>
<td>6s</td>
<td>3s</td>
<td>267ms</td>
<td>3s</td>
</tr>
</tbody>
</table>

#### Percentages (%)

![Percentages Graph](image)

#### Checkstyle Trend

![Checkstyle Trend Graph](image)
Automate the unit tests

- Cobertura and TAP output capabilities
- Define tests in MATLAB, stored in BitBucket and run on Jenkins
- ‘Codifying’ enables quick recreation of test environment
Code coverage report

Trend

File Coverage summary

<table>
<thead>
<tr>
<th>Name</th>
<th>Classes</th>
<th>Lines</th>
<th>Conditionals</th>
</tr>
</thead>
<tbody>
<tr>
<td>@Track/CompleteSegmentConstants.m</td>
<td>100%</td>
<td>101</td>
<td>100%</td>
</tr>
</tbody>
</table>

Coverage Breakdown by Class

<table>
<thead>
<tr>
<th>Name</th>
<th>Lines</th>
<th>Conditionals</th>
</tr>
</thead>
<tbody>
<tr>
<td>@Track/CompleteSegmentConstants</td>
<td>77%</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Source

```c
function CompleteSegmentConstant(obj, segmentNumber)
  \*COMPLETESEGMENT Summary of this function goes here
  \* Detailed explanation goes here
  
  1) obj.TraceLog.StartSection('fn function ' + filename));

  5) \# Constant Force
  6) trainSpeed = min(min(obj, Segment/VehMtrias(segmentNumber,1), obj, route, Segment(segmentNumber,1), SpeedResult[end]));
  9) [Frum, Planet, Frashir, Fraschk, FrasTrain] = CalculateForce(obj, trainSpeed, segmentNumber);
  10) Poconst = (Frenbr + Frenhol + Frestrns + Planet + Frum)^obj.EnableResistanceValue;

  12) \# calculate distance, speed and time
  13) segmentDistance = obj, route, Segment(segmentNumber,1), Distance = ...,
  14) obj, route, Segment(segmentNumber,1), DistanceResult[end];

  16) if obj, route, Segment(segmentNumber-1), SpeedResult[end] > 0
  17) %segment = (segmentDistance/(min(min(obj, Segment/VehMtrias(segmentNumber,1), obj, route, Segment(segmentNumber-1), SpeedResult[end])/3.6));
  19) \# else
  20) if segmentDistance/(min(min(obj, Segment/VehMtrias(segmentNumber,1) + 2.5));

  end
```

14
Our best practices, learnings and recommendations

- Consolidate knowledge, workflows and infrastructure by ’codifying’
- Use BitBucket to deploy build features automatically, e.g. display of code and data size
- Connect to standard, web-based applications
- Prefer "push" over "pull"
- Helps improving quality (non-regression)
Forward looking plans

• Generate documentation from models and publish to Confluence (already started)

• Add model coverage and guideline checking to the automated V&V activities

• Deploy applications to test setups automatically

• Explore Simulink Test and Simulink Check

![Rotor Flux Calculation Diagram](https://bitbucket.strukton.com/projects/AD/repos/models/browse/rotor_flux_calculation)