February 2019

New Tool and Process to Keep System Documentation Current with the Increasing Pace of Agile Product Development

Jim Allen and Bharath Sundar

Delphi Technologies
Table Of Contents

1. Overview
2. Model-Based Algorithm Architecture
3. Model-Based Design Documentation Process Overview
4. Requirements Management Process
5. Requirements Linking Process
6. Conclusions
Overview

➤ The Automotive industry’s shortening design cycles, increased software content, and new software methodologies like AGILE are impacting systems engineering.

➤ New control strategies need developed faster through modeling, auto-coded, verified and delivered to customers with accompanying use documentation.

➤ The fast pace of the AGILE methodology requires Delphi Technologies to work hand-in-hand with our customers and suppliers to develop and refine requirements to make quick interim deliveries.

➤ Delphi Technologies has created a new tool and process to automate system documentation and link requirements to models.
Challenges

➢ Get the model, the software for the controls or diagnostic strategy and the documentation manuals in the hands of the customer to test in a short Sprint Design Phase.

➢ Keep documentation up to date through the design/test/fix cycle and available in the same time frame as the software delivery so the engineers understand how to calibrate the controls or diagnostics strategy.

➢ Show that the requirements in the corporate deployed requirements management tools are satisfied by the models.
Challenges

➢ A typical software release consists of 10 new “Software Change Requests” SCR’s requires about 8 hours to update calibration/tuning guides manually.

➢ If entire new features are added (like Fuel Level & Range, or Traction Control, or Air Charge Determination), it takes many days.

➢ Calibration/Tuning Guides are based on SCR’s. Implementation details will often be missed or stated incorrectly through the traditional documentation approach.

➢ Calibration/Tuning Guides are on the critical path for Systems and Software delivery.
Status Quo with Manual method and Simulink® Report Generator

• **Word processing with model screenshots**
  - Take screenshots of the model and include it in a word processing file.
  - **Pro’s**: Documentation resides in a common word processing software that is widely available.
  - **Con’s**: Time Consuming.

• **Generate PDF using Simulink Report Generator**
  - Automatic screen grabs of the model to generate a .pdf file.
  - **Pro’s**: Documentation resides in a common .pdf processing software that is widely available.
  - **Con’s**: Static screen grabs of the model makes it difficult to read through models with many layers.
New tool and process to keep system documentation current with the increasing pace of Agile product development
Proposal

- **GOAL:** Automate a calibration guide that shows calibration relevant material with requirement links and coverage.

- Seek to overcome some of the shortfalls of the manual method and the Simulink report generator.

- Introduce automation to simplify and standardize model documentation layout.

- Provide a process to document models early in the Agile design cycle.

- Provide a process to link requirements to Simulink® models using the Polarion®ALM™ connector.
In order to better understand the auto-generation process, an overview of the model-based algorithm architecture is discussed.

A typical automotive program consists of over hundred’s of controls, diagnostics and security modules.

Legacy tools tended to use .m scripts for variable declarations instead of native Simulink Data Dictionary (.sldd).

Model-Based Algorithm Architecture
Model-Based Algorithm Architecture

- The Master Model consists of various components that interact with each other to execute specific functions.

- Every component contains its own respective data dictionary (.sldd) also known as a component data dictionary.

- The Master Model also has its own data dictionary (.sldd), also known as Master data dictionary, that is essentially a concatenation of all the individual component data dictionaries (.sldd).
The Master Model also consists of task schedulers that will call on the specific modules at specified execution rates.

Some popular scheduled calls:
- Initialization Calls
- 4, 8, 16, 32, 64, 128, 256, 512, 1000 ms.
- Event based calls.
Model-Based Design Documentation Process Overview

- The auto-generation of calibration guides from models hinges on two important components.
  - Model
    - Component Model
  - Data Dictionary
    - Master Data Dictionary
    - Component Data Dictionary

- The output is a HTML document that includes the documentation and an interactive rendition of the model in an embedded web-view with two-way hyperlinks between the calibration guide content and the model.
Add an introduction to the model.

Add calibration guide content or images to subsystems.

Prioritize Order Of Appearance for subsystems.

Add calibration guide content to StateFlow charts.

Conceal Intellectual property

Assign classes to model data.

Add model data to the data dictionary.
New tool and process to keep system documentation current with the increasing pace of Agile product development.
Requirements Management Process

Polarion®ALM™

- Polarion®ALM™ is a web-interface tool that manages requirements.

- Requirements traceability hinges upon two-way links between Stakeholder Requirements, System Requirements, Test Cases and Test Runs.

- Simulink® models link to System Requirements in Polarion®ALM™.

- Requirements definitions:
  - Stakeholder Requirement = Customer Requirement
  - Test Cases = Functional Test Plan that outlines how to run the test.
  - Test Runs = Pass or Fail indication of the test case.

New tool and process to keep system documentation current with the increasing pace of Agile product development.
Requirements Linking Process

**Polarion®ALM™ to MATLAB® Setup**

1. In order to link a requirement in **Polarion®ALM™** to a model object in Simulink® perform the following steps:

2. Right click on a object or subsystem in MATLAB Simulink. Go To Polarion -> Link Block/Subsystem with Existing Item

3. If your “WorkItemSelectionDialog” is populated with Work ID’s from your project.

4. Once the requirement is linked, a .slmx requirements traceability file is generated.

---

New tool and process to keep system documentation current with the increasing pace of Agile product development
Requirements Linking Process

Polarion®ALM™ to MATLAB ® Setup

- In an Agile environment, it is important to convey algorithm development progress during interim deliveries.

- A summary of linked requirements and % coverage can be automatically populated in the auto-generated calibration guide of the model.

- This is performed using the “rmi” api function – Requirements Management Interface.
Keeping system documentation current with the increasing pace of Agile product development
Conclusions

At Delphi Technologies we created our own tool and process using MATLAB and SIMULINK Report Gen API’s.

The auto-generation of calibration guides from models has streamlined systems and overall program workflow in the AGILE Development Process by building on engineering effort that is performed upstream of the process.

The result has made it quicker to deliver accurate and current documentation by 75%.
Thank you
APPENDIX
Model-Based Design Documentation Process Overview

- The foundation of the automation philosophy is based on the literature titled: “Model Description Documentation Recommended Practice for Ground Vehicle System and Subsystem Simulation”.

- The literature discusses fifteen types of documentation guidelines out of which seven types have been pre-selected for the auto-generation of the calibration guides.

- The seven types that are chosen:
  - Model title
  - High level description of model
  - Feature and capabilities
  - External interface variables (or inputs and outputs)
  - Internal variables
  - Parameters and calibration procedures
  - Detailed functional description
Model-Based Design Documentation Process Overview

- The fundamental building blocks for the auto-generation process is dependent on the model and data dictionary and it’s respective components.

- A model consists of the following:
  - Blocks – Main elements used to build models. Examples – Add, subtract, divide etc.
  - Subsystems – Grouping of blocks to create a hierarchical model comprising of many layers.
  - Stateflow Charts – Modeling sequential logic based on state machines and flow charts.
Model-Based Design Documentation Process Overview

- A data dictionary consists of the following:
  - Class – Type of signal – Variable, Calibration or Constant.
  - Name – Signal name.
  - Data Type – Boolean, integer, float etc.
  - Definition File – C file that defines this variable, calibration or constant.
  - Description – Purpose of this type of variable, calibration or constant.
  - Header File – C header file that defines this variables calibration or constant.
  - Min – Minimum value of variable, calibration or constant.
  - Max – Maximum value of variable, calibration or constant.
  - Value – Typical value assigned to calibration.
  - Units – Units associated with the variable, calibration or constant.
Model-Based Design Documentation Process Overview

- To facilitate code generation, Simulink allows a modeler to assign a class name to variables, calibrations and constants that share similar characteristics.

- The calibration guide generation can use these classes to distinguish between constants, variables and calibrations.

- For example, the model’s data dictionary can facilitate these distinctions by assigning model data to the following standard classes:
  - Variables defined as “Simulink.Signal” or “mpt.Signal” class.
  - Calibrations defined as “mpt.Parameter” class.
  - Constants defined as “Simulink.Parameter” class.
Conclusions

Future Work

➢ Provide further enhances to the HTML documentation by providing clickable two-way links between signals summarized in the table(s) and the embedded web-view of the model.

➢ Work with Polarion®ALM™ to further enhance the ability to better calculate requirements coverage.

➢ Establish a process and automation methodology to include simulation results of a model in the documentation to better understand an algorithms viability.
Code Snippets
Code Snippets

Create a subsection for each subsystem

- To create subsections for each subsystem various API functions need to be used.

- Use the find_system function to get a list of the subsystems in the model.

- Then use the get_param function to extract the description field of each subsystem. Then iterate through the subsystems, using the append function to add the description to the HTML guide.

```plaintext
%% Create a section for each subsystem.
blockPaths = find_system(diagHandle, 'SearchDepth', 0);
% Extract subsystem description
blockDescription = get_param(blockPaths,'Description');
p = Paragraph(blockDescription); % Convert to Paragraph
for path = blockPaths'
p.Style = {WhiteSpace('preserve')}; % Append Descriptions to document
append(sdd,p);
end
```
Code Snippets

Generate tables from the data dictionary

- Section 2 of the guide is populated with the help of the data dictionary. The code shows the variety of functions required to extract the relevant information from the data dictionary.
- getSection gets the section of the data dictionary
- The find_system function is used to get the model's input and output blocks
- The information is then used to create a table of the inputs and outputs containing their descriptions and signal (i.e. variable) attributes.
- The FormalTable function is used to create an empty table that is then populated with the input and output data.
- Once the table is populated, the table is appended to the HTML guide. A similar approach is used to generate the internal tables in section 3.

```plaintext
dDataSectObj = getSection(myDictionaryObj,'Design Data');
modelName = h.ExportOptions.Diagrams;
input_blocks = find_system(modelName,'SearchDepth', 2, 'BlockType', 'Inport');
inputData = get_param(input_blocks, 'Name');
output_blocks = find_system(modelName, 'SearchDepth', 2, 'BlockType', 'Outport');
outputData = get_param(output_blocks, 'Name');
Entries_mpt = find(dDataSectObj,'-value','-class',mpt.Signal);
Entries_simulinksignal = find(dDataSectObj,'-value','-class',Simulink.Signal);
table_input = FormalTable({'Input Variable Name', 'Description', 'DataType', 'Min', 'Max', 'Units'}, InputTable);
append(h,table_input);
```
Requirements Linking Process

_Polarion®ALM™ to MATLAB® Setup_

1. Polarion Read/Write Access Privileges

2. MATLAB® 2013a or higher.

3. MATLAB® Simulink Requirements v 1.0.

4. Simulink to Polarion Connector -> Available through Polarion®ALM™
Requirements Linking Process

Polarion® ALM™ to MATLAB® Setup

1. Simulink to Polarion Connector is a zip file. Once downloaded, unzip the contents of the file.

2. The unzipped file will have many items in it. The most important are startup.p & documentation:
   1. Documentation is the help guide.
   2. Startup.p is the initialization file.

3. Open MATLAB.

4. Type “run startup.p” in the command window.

5. You should see the following confirmation:

```
Polarion-addon initialization finished
```
Code Snippets

*List of supporting API’s (Application Program Interface)*

- Simulink®, the MATLAB® Report Generator™, and the Simulink® Report Generator™ provide API’s that facilitate development of MATLAB® programs capable of generating calibration guides from models.

- These APIs consist of functions that extract and format data from a model. The table lists some of the API functions used to autogenerate the example guide.

<table>
<thead>
<tr>
<th>API Name</th>
<th>API Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createDiagramLink</td>
<td>Create a two-way link between guide content and a Web View element.</td>
</tr>
<tr>
<td>get_param</td>
<td>Return the name of a specified model or block object.</td>
</tr>
<tr>
<td>find_system</td>
<td>Find blocks, charts, signals and other model elements.</td>
</tr>
<tr>
<td>append</td>
<td>Append text, tables, images, lists, figures, etc., to a document container.</td>
</tr>
<tr>
<td>getSection</td>
<td>Get section of a data dictionary.</td>
</tr>
<tr>
<td>FormatTable</td>
<td>Define a table that has a body and optionally a table header, a table footer,</td>
</tr>
<tr>
<td></td>
<td>or both.</td>
</tr>
</tbody>
</table>
Requirements Housing Process

Example

The ECM shall provide a means to control fuel enrichment during transient events that minimizes engine out emissions during transient throttle opening.

AE Truncation for Decreasing Load
If throttle position decreases more than AE_Neg_Throttle_Disable per fuel event, then the AE_Fuel[x] accumulators are cleared.

2.1.2 FC: Test for AE Truncation for decreasing load – AE Cleared
1. Set AE_Throttle_Filter_Coeff_Fast = 0
2. Set AE_Throttle_Filter_Coeff_Slow = 0

Test Result  Test Run
✓ Passed  ECM_DEL_POC_21L_015_000 - ECM_DEL_POC_21L_015_000

New tool and process to keep system documentation current with the increasing pace of Agile product development.
Requirements Linking Process

Polarion®ALM™ to MATLAB ® Setup

1. In order to verify that the Polarion Connector has been added to Simulink, open up an existing, blank or demo Simulink model in MATLAB.

2. Go to Tools in the top right corner and Polarion Settings will be available to choose as an option. This shows that the Polarion Connector has been installed successfully.

3. Once you click the Polarion settings option, the following pop-up window will display as shown on the right:
Requirements Linking Process

**Polarion® ALM™ to MATLAB® Setup**

```matlab
function appendRequirements(h)
    import mlreportgen.dom.*
    append(h,mlreportgen.domHeading(1,'Linked Requirements'));
    modelName = h.ExportOptions.DIAGRAMS;
    blockPaths = find_system(modelName, 'SearchDepth', 5);
    for i=1:length(blockPaths)
        reqs = rmi('get', blockPaths(i));
        if ~isempty(reqs)
            x = reqs.description;
            y = reqs.keywords;
            rectTable(i,1) = cellstr(x);  % Polaron Work ID
            rectTable(i,2) = cellstr(y);  % Polaron Keyword
        end
    end
end
```

```matlab
%% Requirements Coverage
% Find Linked vs unlinked requirements and calculate requirements
% coverage
total = length(blockPaths);
linked_WI = length(fullValz);
unlinked_WI = total-linked_WI;
req_pctcov = (linked_WI/total)*100;
req_pctcov_rounded = round(req_pctcov,0);
req_cov = [linked_WI unlinked_WI req_pctcov_rounded];
h_reqCoverage = append(h, mlreportgen.domHeading(2, 'Requirements Coverage'));
```