Test Driven Development in Agile Model-Based Design

Paul Urban
Marco Dragic
Marco Dragic
Senior Product Manager
Simulink Platform

Paul Urban
Senior Product Manager
Simulink Verification and Validation
Building Algorithms in Everything…
Building Algorithms in Everything…

…but how do you deliver faster, meet changing customer requirements, and ensure quality?
Test Driven Development Cycle

1. Create a test
2. Implement enough for test to pass
3. Refactor
Simulink provides an integrated framework for TDD
Starting with high level customer requirements

User Requirements:

• Both driver and passenger can control the window
• Window stops closing if an object is detected
• Window should have option to fully open and close
Capturing requirements

<table>
<thead>
<tr>
<th>Index</th>
<th>Requirement Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Activity diagram: Power window control</td>
</tr>
<tr>
<td>1.2</td>
<td>Open and Close Window</td>
</tr>
<tr>
<td>1.2.1</td>
<td>Fully Open</td>
</tr>
<tr>
<td>1.2.2</td>
<td>Fully Close</td>
</tr>
<tr>
<td>1.3</td>
<td>Move Response</td>
</tr>
<tr>
<td>1.4</td>
<td>Detect Object</td>
</tr>
<tr>
<td>1.5</td>
<td>Validate Driver</td>
</tr>
<tr>
<td>1.6</td>
<td>Validate Passenger</td>
</tr>
<tr>
<td>1.7</td>
<td>Driver commands</td>
</tr>
<tr>
<td>1.8</td>
<td>Passenger Commands</td>
</tr>
</tbody>
</table>

Controller Functional Requirements

<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>The power window must have two o...</td>
</tr>
<tr>
<td>2.2</td>
<td>The power window should not move...</td>
</tr>
<tr>
<td>2.3</td>
<td>The power window can be operated...</td>
</tr>
<tr>
<td>2.3.1</td>
<td>The driver side for the passenger...</td>
</tr>
<tr>
<td>2.4</td>
<td>Driver Move down operation</td>
</tr>
<tr>
<td>2.4.1</td>
<td>Driver down button press</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Move down to end stop</td>
</tr>
<tr>
<td>2.4.3</td>
<td>Move down automatically performance</td>
</tr>
<tr>
<td>2.4.4</td>
<td>Enter neutral when fully down</td>
</tr>
<tr>
<td>2.5</td>
<td>Driver Move up operation</td>
</tr>
<tr>
<td>2.5.1</td>
<td>The driver power window should a...</td>
</tr>
<tr>
<td>2.5.2</td>
<td>The driver power window should a...</td>
</tr>
<tr>
<td>2.5.3</td>
<td>The move up operation must be fi...</td>
</tr>
<tr>
<td>2.5.4</td>
<td>Once the window fully up, the dr...</td>
</tr>
<tr>
<td>2.6</td>
<td>Passenger Move down operation</td>
</tr>
<tr>
<td>2.6.1</td>
<td>The Passenger power window should...</td>
</tr>
<tr>
<td>2.6.2</td>
<td>The Passenger power window should...</td>
</tr>
<tr>
<td>2.6.3</td>
<td>The move down operation must be ...</td>
</tr>
<tr>
<td>2.6.4</td>
<td>Once the window fully down, the ...</td>
</tr>
</tbody>
</table>

Keywords:
- Obstacle position
- Bottom

Rationale:
Both the driver and passenger can send commands to the window to move it up and down. The controller infers the correct command to send to the window actuator (e.g., the driver command has priority over the passenger command). In addition, diagram monitors the state of the window system to establish when the window is fully opened and closed and to detect if there is an object between the window and frame.

Keywords: Obstacle position, Bottom

Implemented by:
- power_window_control_system
Organizing and creating requirement hierarchies
Specifying details

Description / Rationale Fields

Links Pane
1. Create a test

2. Implement enough for test to pass

3. Refactor
Develop, manage, and execute simulation-based tests

**Simulink Test**

**Test Manager**
- Author, manage, organize tests

**Test Harnesses**
- Isolate Component Under Test

**Test Authoring**
- Specify test inputs, expected outputs, and tolerances

- **Test Browser**
- **Test Results**
- **Test Authoring**
  - **Signal Editor**
  - **Time-Series Data**
  - **Temporal Assessments**
- **Main Model**
- **Component under test**
- **Test Harness**

**Report Generated by Test Manager**
- Title: LandignStartControlRegressionTests
  - Author: Jane Doe
  - Date: 20 Mar 2013 18:30:33

**Test Sequence**
Creating a Test Harness to isolate Component Under Test
Specify properties of the Test Harness
Specify inputs

- MAT file (input)
- Excel file (input)
- Test Sequence
- Signal Editor
Specify outputs

MAT file: MAT file
Excel: Excel
Assessments: Assessments

Outputs
Created Test Harness to isolate Component Under Test
Authoring tests using Signal Editor
Use templates and wizards to automate test case creation
Use templates and wizards to automate test case creation

Test Case Templates

Simulation Test
- Input
- Output
- Assessment Criteria

Baseline Test
- Input
- Output
- File
- Expected Outputs
- Assessment Criteria

Equivalence Test
- Input
- Output
- Assessment Criteria
Create Simulation Test and link to requirement

- Link to requirements
- Specify model to test
Test fails due to compilation error
1. Create a test

2. Implement enough for test to pass

3. Refactor
Implement enough to get test to pass
Linking implementation to requirements

If the up command is issued for between 200 ms and 1 s, the window must fully open.
Managing artifacts with source control directly from Projects
Scale and automate testing with Continuous Integration

- Schedule automatic code and model testing
- Access MATLAB Plugin for Jenkins
Executing test with Test Manager

- Group into suites and test files
- Execute individual or batch
Analyzing and debugging results with Test Manager

- View result summary
- Debug using Simulation Data Inspector
- Archive, export, and report results
Executing all tests until they pass

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>DriverMoveDown</td>
<td>✔</td>
</tr>
<tr>
<td>DriverMoveUp</td>
<td>✔</td>
</tr>
<tr>
<td>EmergencyObstacle</td>
<td>✔</td>
</tr>
<tr>
<td>Detect Object</td>
<td>✔</td>
</tr>
<tr>
<td>Fully Close Window</td>
<td>✔</td>
</tr>
<tr>
<td>PassengerMovedown</td>
<td>✔</td>
</tr>
</tbody>
</table>
Measuring testing completeness with coverage

- Identify testing gaps
- Missing requirements
- Unintended functionality
- Design errors
Generating test reports for audits and reviews
1. Create a test

2. Implement enough for test to pass

3. Refactor
Refactoring

- Refactoring is the process of changing software in such a way that it does not alter the external behavior of the code yet improves its internal structure.
Refactoring takes many shapes and forms

- Rearranging Layout
Refactoring takes many shapes and forms

- Rearranging Layout
Refactoring takes many shapes and forms

- Rearranging Layout
- Restructuring Hierarchy
Refactoring takes many shapes and forms

- Rearranging Layout
- Restructuring Hierarchy
- Optimizing Implementation
Refactoring takes many shapes and forms

- Rearranging Layout
- Restructuring Hierarchy
- Optimizing Implementation
- Project-wide Renaming

…. and many more!
Refactor by consolidating redundant Stateflow chart

Driver and Passenger Controls are identical
Detecting clones with Clone Detector App
Test Driven Development Cycle

1. Create a test
2. Implement enough for test to pass
3. Refactor
Conclusion and key takeaways

Simulink provides an integrated framework for TDD

Systematically verify requirements

Automate testing to deliver working systems faster
Test Driven Development
powered by MATLAB and Simulink

- Model-Based Design – Simulink and Stateflow
- Manage Requirements – Simulink Requirements
- Author and Execute Tests – Simulink Test
- Measure Test Completeness – Simulink Coverage
- Refactor and Verify Compliance – Simulink Check
- Continuous Integration – MATLAB Plug in for Jenkins
- Organize, Manage and Share – Projects
Learn more

- Agile System Development with Model-Based Design
- Agile Model-Based Design: Accelerating Simulink Simulations in Continuous Integration Workflows
- Verification, Validation, and Test Solution Page
- Continuous Integration Solution Page