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

Controller Functional Requirements

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

- 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:

- Revision information:
- Links
  - Implemented by: power_window_control_system
**Viewing details**

<table>
<thead>
<tr>
<th>Requirement Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement 1</td>
</tr>
<tr>
<td>1.1 Quantitative Requirements</td>
</tr>
<tr>
<td>1.2 Activity diagram: Power window control</td>
</tr>
<tr>
<td>1.2.1 Fully Open</td>
</tr>
<tr>
<td>1.2.2 Fully Close</td>
</tr>
<tr>
<td>1.3 Move Response</td>
</tr>
<tr>
<td>1.4 Detect Object</td>
</tr>
<tr>
<td>1.5 Validate Driver</td>
</tr>
<tr>
<td>1.6 Validate Passenger</td>
</tr>
<tr>
<td>1.7 Driver commands</td>
</tr>
<tr>
<td>1.8 Passenger Commands</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 The power window must have two o...</td>
</tr>
<tr>
<td>2.2 The power window should not move...</td>
</tr>
<tr>
<td>2.3 The power window can be operated...</td>
</tr>
<tr>
<td>2.3.1 The driver side for the passenger...</td>
</tr>
<tr>
<td>2.4 Driver Move down operation</td>
</tr>
<tr>
<td>2.4.1 Driver down button press</td>
</tr>
<tr>
<td>2.4.2 Move down to end stop</td>
</tr>
<tr>
<td>2.4.3 Move down automatically performance</td>
</tr>
<tr>
<td>2.4.4 Enter neutral when fully down</td>
</tr>
<tr>
<td>2.5 Driver Move up operation</td>
</tr>
<tr>
<td>2.5.1 The driver power window should ...</td>
</tr>
<tr>
<td>2.5.2 The driver power window should ...</td>
</tr>
<tr>
<td>2.5.3 The move up operation must be fi...</td>
</tr>
<tr>
<td>2.5.4 Once the window fully up, the dr...</td>
</tr>
<tr>
<td>2.6 Passenger Move down operation</td>
</tr>
<tr>
<td>2.6.1 The Passenger power window should...</td>
</tr>
<tr>
<td>2.6.2 The Passenger power window should...</td>
</tr>
<tr>
<td>2.6.3 The move down operation must be ...</td>
</tr>
<tr>
<td>2.6.4 Once the window fully down, the ...</td>
</tr>
</tbody>
</table>
Organizing and creating requirement hierarchies
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

---

**Tools and Features**

- **Test Browser**
- **Test Results**
- **Report Generated by Test Manager**
- **Main Model**
- **Component under test**
- **Test Harness**
- **Signal Editor**
- **Time-Series Data**
- **Temporal Assessments**
- **Test Sequence**

---

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

Create Test Harness

Specify the properties of the test harness. The component under test is the system for which the harness is being created. After creation, use the block badge to find and open harnesses.

Component under Test: slexPowerWindowExample/power_window_control_system

Basic Properties  Advanced Properties  Description

Name: slexPowerWindowExample_Harness2

Harnesses saved internally. More information

Sources and Sinks

Input \(\rightarrow\) Component under Test \(\rightarrow\) Output

- Create scalar inputs
- Generate function-call signals using: Test Sequence
- Add separate Test Assessment block
- Open harness after creation

OK  Cancel  Help
Specify inputs

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

- MAT file
- Excel
- Assessments
Created Test Harness to isolate Component Under Test

Test Harness

Main Model
Authoring tests using Signal Editor
Use templates and wizards to automate test case creation
Use templates and wizards to automate test case creation
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
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

```
  u1  y1
  u2  y2
  u3  y3
  u4  y4
```
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
- Manage Requirements
- Author and Execute Tests
- Measure Test Completeness
- Refactor and Verify Compliance
- Continuous Integration
- Organize, Manage and Share

- Simulink and Stateflow
- Simulink Requirements
- Simulink Test
- Simulink Coverage
- Simulink Check
- MATLAB Plug in for Jenkins
- 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