The Craft of Model-Based Testing

Paul C. Jorgensen
## 5 Petri Nets

5.1 Definition and Notation ................................................................. 81
   5.1.1 Transition Enabling and Firing ............................................. 81
   5.1.2 Conventions ........................................................................ 82
   5.1.3 Non-graphical Representations ........................................... 84
       5.1.3.1 Textual Representation ............................................... 84
       5.1.3.2 Database Representation ........................................... 85
5.2 Technique ................................................................................... 86
   5.2.1 Sequence, Selection, and Repetition ..................................... 87
   5.2.2 Enable, Disable, and Activate ............................................. 88
   5.2.3 Trigger ............................................................................... 89
   5.2.4 Suspend, Resume, and Pause ............................................. 89
   5.2.5 Conflict and Priority .......................................................... 91
   5.2.6 Mutual Exclusion .............................................................. 91
   5.2.7 Synchronization .................................................................. 91
   5.2.8 Some Consequences of Marking and Enabling .................... 92
   5.2.9 Petri Nets and Finite State Machines ................................... 93
   5.2.10 Petri Net Engines ............................................................. 94
5.3 Examples ...................................................................................... 95
   5.3.1 The Producer–Consumer Problem ....................................... 95
   5.3.2 The Windshield Wiper Controller ....................................... 97
5.4 Deriving Test Cases from a Petri Net ............................................. 101
   5.4.1 The Insurance Premium Problem ....................................... 102
   5.4.2 The Garage Door Controller ............................................. 105
5.5 Lessons Learned from Experience ................................................ 111
5.6 Advantages and Limitations ....................................................... 113
References ....................................................................................... 113

## 6 Event-Driven Petri Nets

6.1 Definition and Notation ................................................................. 117
   6.1.1 Transition Enabling and Firing ............................................. 118
   6.1.2 Conventions ........................................................................ 120
   6.1.3 Non-graphical Representations ........................................... 121
       6.1.3.1 Textual Representation ............................................... 121
       6.1.3.2 Database Representation ........................................... 121
6.2 Technique ................................................................................... 123
   6.2.1 Context-Sensitive Input Events .......................................... 124
   6.2.2 Multiple-Cause Output Events .......................................... 124
   6.2.3 Event Quiescence .............................................................. 124
   6.2.4 Event-Driven Petri Net Engines ......................................... 124
   6.2.5 Deriving Test Cases from an Event-Driven Petri Net ............ 126
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.3</td>
<td>Examples</td>
<td>126</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Railroad Crossing Gate Controller</td>
<td>126</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Windshield Wiper Controller</td>
<td>128</td>
</tr>
<tr>
<td>6.4</td>
<td>Deriving Test Cases from an Event-Driven Petri Net</td>
<td>130</td>
</tr>
<tr>
<td>6.4.1</td>
<td>The Insurance Premium Problem</td>
<td>131</td>
</tr>
<tr>
<td>6.4.2</td>
<td>The Garage Door Controller</td>
<td>132</td>
</tr>
<tr>
<td>6.5</td>
<td>Lessons Learned from Experience</td>
<td>141</td>
</tr>
<tr>
<td>6.6</td>
<td>Advantages and Limitations</td>
<td>143</td>
</tr>
<tr>
<td></td>
<td>References</td>
<td>145</td>
</tr>
<tr>
<td>6.3</td>
<td>Examples</td>
<td>144</td>
</tr>
<tr>
<td>7</td>
<td>Statecharts</td>
<td>147</td>
</tr>
<tr>
<td>7.1</td>
<td>Definition and Notation</td>
<td>147</td>
</tr>
<tr>
<td>7.2</td>
<td>Technique</td>
<td>151</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Communication with the Broadcasting Mechanism</td>
<td>151</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Statechart Engines</td>
<td>152</td>
</tr>
<tr>
<td>7.2.3</td>
<td>Deriving Test Cases from a Statechart</td>
<td>154</td>
</tr>
<tr>
<td>7.3</td>
<td>Examples</td>
<td>154</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Railroad Crossing Gate Controller</td>
<td>154</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Windshield Wiper Controller</td>
<td>156</td>
</tr>
<tr>
<td>7.4</td>
<td>The Continuing Problems</td>
<td>157</td>
</tr>
<tr>
<td>7.4.1</td>
<td>The Insurance Premium Problem</td>
<td>157</td>
</tr>
<tr>
<td>7.4.2</td>
<td>The Garage Door Controller</td>
<td>157</td>
</tr>
<tr>
<td>7.5</td>
<td>Lessons Learned from Experience</td>
<td>165</td>
</tr>
<tr>
<td>7.6</td>
<td>Advantages and Limitations</td>
<td>165</td>
</tr>
<tr>
<td></td>
<td>Reference</td>
<td>166</td>
</tr>
<tr>
<td>7</td>
<td>Statecharts</td>
<td>147</td>
</tr>
<tr>
<td>7.1</td>
<td>Definition and Notation</td>
<td>147</td>
</tr>
<tr>
<td>7.2</td>
<td>Technique</td>
<td>151</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Communication with the Broadcasting Mechanism</td>
<td>151</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Statechart Engines</td>
<td>152</td>
</tr>
<tr>
<td>7.2.3</td>
<td>Deriving Test Cases from a Statechart</td>
<td>154</td>
</tr>
<tr>
<td>7.3</td>
<td>Examples</td>
<td>154</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Railroad Crossing Gate Controller</td>
<td>154</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Windshield Wiper Controller</td>
<td>156</td>
</tr>
<tr>
<td>7.4</td>
<td>The Continuing Problems</td>
<td>157</td>
</tr>
<tr>
<td>7.4.1</td>
<td>The Insurance Premium Problem</td>
<td>157</td>
</tr>
<tr>
<td>7.4.2</td>
<td>The Garage Door Controller</td>
<td>157</td>
</tr>
<tr>
<td>7.5</td>
<td>Lessons Learned from Experience</td>
<td>165</td>
</tr>
<tr>
<td>7.6</td>
<td>Advantages and Limitations</td>
<td>165</td>
</tr>
<tr>
<td></td>
<td>Reference</td>
<td>166</td>
</tr>
<tr>
<td>8</td>
<td>Swim Lane Event-Driven Petri Nets</td>
<td>167</td>
</tr>
<tr>
<td>8.1</td>
<td>Definition and Notation</td>
<td>167</td>
</tr>
<tr>
<td>8.1.1</td>
<td>Transition Enabling and Firing</td>
<td>168</td>
</tr>
<tr>
<td>8.1.2</td>
<td>Events in a Swim Lane Event-Driven Petri Net</td>
<td>169</td>
</tr>
<tr>
<td>8.2</td>
<td>Technique</td>
<td>169</td>
</tr>
<tr>
<td>8.2.1</td>
<td>Using Swim Lanes</td>
<td>169</td>
</tr>
<tr>
<td>8.2.2</td>
<td>“Model Checking”</td>
<td>170</td>
</tr>
<tr>
<td>8.2.3</td>
<td>Deriving Test Cases from a Swim Lane Event-Driven Petri Net</td>
<td>173</td>
</tr>
<tr>
<td>8.3</td>
<td>The Continuing Problems</td>
<td>176</td>
</tr>
<tr>
<td>8.3.1</td>
<td>The Insurance Premium Problem</td>
<td>176</td>
</tr>
<tr>
<td>8.3.2</td>
<td>The Garage Door Controller</td>
<td>176</td>
</tr>
<tr>
<td>8.3.2.1</td>
<td>The Light Beam Sensor</td>
<td>177</td>
</tr>
<tr>
<td>8.3.2.2</td>
<td>The End-of-Track Sensors</td>
<td>178</td>
</tr>
<tr>
<td>8.3.2.3</td>
<td>The Door Opening Interactions</td>
<td>179</td>
</tr>
<tr>
<td>8.3.2.4</td>
<td>Failure Mode Event Analysis</td>
<td>181</td>
</tr>
</tbody>
</table>

References

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181
8.4 Deriving Test Cases from Swim Lane Event-Driven Petri Nets .... 188
8.5 Lessons Learned from Experience ........................................ 189
Reference .................................................................................. 190

9 Object-Oriented Models .......................................................... 191
  9.1 Notation and Technique ..................................................... 193
    9.1.1 Use Case Diagrams ................................................. 194
    9.1.2 Activity Diagrams ................................................... 195
    9.1.3 Statechart Diagrams ............................................... 196
    9.1.4 Sequence Diagrams ............................................... 198
  9.2 Examples ........................................................................... 199
  9.3 The Continuing Problems ................................................. 199
    9.3.1 The Insurance Premium Problem .............................. 199
    9.3.2 The Garage Door Controller .................................... 201
      9.3.2.1 Activity Diagram .............................................. 201
      9.3.2.2 Use Cases for the Garage Door Controller .......... 202
      9.3.2.3 Use Case Diagram for the Garage Door Controller 213
      9.3.2.4 Sequence Diagram for the Garage Door Controller 214
      9.3.2.5 Statechart for the Garage Door Controller ......... 215
  9.4 Deriving Test Cases from UML Models .............................. 215
    9.4.1 Test Cases from Activity Diagrams ............................ 215
    9.4.2 Test Cases from Use Cases ....................................... 217
    9.4.3 Test Cases from a Use Case Diagram ....................... 217
    9.4.4 Test Cases from Sequence Diagrams ....................... 217
    9.4.5 Test Cases from Statecharts ................................... 217
  9.5 Advantages and Limitation ................................................. 218
    References .......................................................................... 220

10 Business Process Modeling and Notation ................................. 221
  10.1 Definition and Notation .................................................... 221
  10.2 Technique ........................................................................ 223
  10.3 Example ............................................................................ 223
  10.4 Deriving Test Cases from a Business Process Modeling and Notation Definition ............................................ 223
    10.4.1 The Insurance Premium Problem ............................ 223
    10.4.2 The Garage Door Controller ................................. 226
  10.5 Advantages and Limitations .............................................. 226
PART 2  THE PRACTICE OF MODEL-BASED TESTING

11 About the International Software Testing Qualification Board       231
   11.1 The ISTQB Organization                                      231
   11.2 Certification Levels                                         232
   11.3 The ISTQB MBT Syllabus                                      233
       11.3.1 Introduction to Model-Based Testing                    233
           11.3.1.1 Objectives and Motivations for MBT             233
           11.3.1.2 MBT Activities and Artifacts in the Fundamental Test Process ... 234
           11.3.1.3 Integrating MBT into the Software Development Lifecycle ... 234
       11.3.2 Model-Based Testing Modeling                             234
           11.3.2.1 Model-Based Testing Modeling                      234
           11.3.2.2 Languages for MBT Models                         234
           11.3.2.3 Good Practices for Model-Based Testing Modeling Activities ... 235
       11.3.3 Selection Criteria for Test Case Generation             235
           11.3.3.1 Classification of Model-Based Testing Test Selection Criteria ... 235
           11.3.3.2 Applying Test Selection Criteria                  235
       11.3.4 MBT Test Implementation and Execution                   236
           11.3.4.1 Specifics of Model-Based Testing Test Implementation and Execution ... 236
           11.3.4.2 Activities of Test Adaptation in Model-Based Testing ... 236
       11.3.5 Evaluating and Deploying a Model-Based Testing Approach ... 236

Reference .................................................................................. 240

12 Implementing MBT in an Organization ...................................... 241
   12.1 Getting Started .................................................................. 242
       12.1.1 Recognizing the Need for Change                      243
       12.1.2 Technology Champions                                 244
   12.2 Getting Started .................................................................. 245
       12.2.1 Candidate Model-Based Testing Products .................. 245
       12.2.2 Success Criteria                                      245
       12.2.3 Pilot Projects                                        246
   12.3 Training and Education .................................................... 246
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.4</td>
<td>Lessons Learned from Experience</td>
<td>247</td>
</tr>
<tr>
<td>12.4.1</td>
<td>The Medium</td>
<td>248</td>
</tr>
<tr>
<td>12.4.2</td>
<td>The Tools</td>
<td>249</td>
</tr>
<tr>
<td>12.4.3</td>
<td>The Ability to Use Tools Well</td>
<td>250</td>
</tr>
<tr>
<td>References</td>
<td></td>
<td>250</td>
</tr>
<tr>
<td>13</td>
<td>Information Provided to Model-Based Testing Tool Vendors</td>
<td>251</td>
</tr>
<tr>
<td>13.1</td>
<td>Chapter Template</td>
<td>251</td>
</tr>
<tr>
<td>13.2</td>
<td>A Unit Level Problem: Insurance Premium Calculation</td>
<td>252</td>
</tr>
<tr>
<td>13.2.1</td>
<td>Problem Definition</td>
<td>252</td>
</tr>
<tr>
<td>13.2.2</td>
<td>Problem Models</td>
<td>252</td>
</tr>
<tr>
<td>13.2.2.1</td>
<td>Flowchart</td>
<td>252</td>
</tr>
<tr>
<td>13.2.2.2</td>
<td>Decision Table(s)</td>
<td>253</td>
</tr>
<tr>
<td>13.2.2.3</td>
<td>Finite State Machine</td>
<td>258</td>
</tr>
<tr>
<td>13.2.3</td>
<td>Insurance Premium Problem Visual Basic for Applications Code</td>
<td>259</td>
</tr>
<tr>
<td>13.3</td>
<td>A System Level Problem: The Garage Door Controller</td>
<td>260</td>
</tr>
<tr>
<td>13.3.1</td>
<td>Problem Definition</td>
<td>260</td>
</tr>
<tr>
<td>13.3.2</td>
<td>Problem Models</td>
<td>261</td>
</tr>
<tr>
<td>13.3.2.1</td>
<td>Flowchart</td>
<td>261</td>
</tr>
<tr>
<td>13.3.2.2</td>
<td>Decision Table(s)</td>
<td>261</td>
</tr>
<tr>
<td>13.3.2.3</td>
<td>Finite State Machine</td>
<td>264</td>
</tr>
<tr>
<td>13.3.3</td>
<td>Garage Door Controller VBA Code</td>
<td>264</td>
</tr>
<tr>
<td>14</td>
<td>Smartesting Yest and CertifyIt</td>
<td>269</td>
</tr>
<tr>
<td>14.1</td>
<td>Introduction</td>
<td>269</td>
</tr>
<tr>
<td>14.1.1</td>
<td>Smartesting Products</td>
<td>270</td>
</tr>
<tr>
<td>14.1.1.1</td>
<td>Yest</td>
<td>270</td>
</tr>
<tr>
<td>14.1.1.2</td>
<td>CertifyIt</td>
<td>270</td>
</tr>
<tr>
<td>14.1.2</td>
<td>Customer Support</td>
<td>271</td>
</tr>
<tr>
<td>14.2</td>
<td>Insurance Premium Results Using Yest</td>
<td>271</td>
</tr>
<tr>
<td>14.3</td>
<td>Garage Door Controller Results Using CertifyIt</td>
<td>277</td>
</tr>
<tr>
<td>14.4</td>
<td>Vendor Advice: Best Practices with Yest and CertifyIt</td>
<td>286</td>
</tr>
<tr>
<td>References</td>
<td></td>
<td>287</td>
</tr>
<tr>
<td>15</td>
<td>TestOptimal</td>
<td>289</td>
</tr>
<tr>
<td>15.1</td>
<td>Introduction</td>
<td>289</td>
</tr>
<tr>
<td>15.1.1</td>
<td>Product Architecture</td>
<td>290</td>
</tr>
<tr>
<td>15.1.2</td>
<td>The TestOptimal Product Suite</td>
<td>290</td>
</tr>
<tr>
<td>15.1.3</td>
<td>Customer Support</td>
<td>292</td>
</tr>
<tr>
<td>15.2</td>
<td>Insurance Premium Results</td>
<td>292</td>
</tr>
<tr>
<td>15.3</td>
<td>Garage Door Controller Results</td>
<td>294</td>
</tr>
<tr>
<td>15.4</td>
<td>Vendor Advice</td>
<td>299</td>
</tr>
</tbody>
</table>
## 16 Conformiq, Inc.

16.1 Introduction .................................................................................. 301

16.1.1 Features .................................................................................. 301

16.1.1.1 Test Generation ................................................................. 301

16.1.1.2 Modeling Language (Conformiq Designer) ..................... 302

16.1.1.3 Modeling Language (Conformiq Creator) ....................... 302

16.1.1.4 User Interface .................................................................... 303

16.1.1.5 Integration with Other Tools ........................................... 303

16.1.1.6 Other Features .................................................................. 304

16.1.2 The Conformiq 360° Test Automation Product Suite ........... 304

16.1.3 Customer Support .................................................................. 305

16.2 Insurance Premium Results .......................................................... 305

16.2.1 Conformiq Creator Input ....................................................... 307

16.2.2 Generated Test Cases ......................................................... 309

16.2.3 Test Coverage Analysis ....................................................... 310

16.3 Garage Door Controller Results ................................................... 324

16.3.1 Input Diagram and QML Text Files ..................................... 325

16.3.1.1 CQA Files ...................................................................... 325

16.3.2 Generated Test Cases ......................................................... 327

16.3.3 Traceability Matrices ............................................................ 332

16.4 Vendor Advice .......................................................................... 337

References ....................................................................................... 338

## 17 Elvior

17.1 Introduction .................................................................................. 339

17.1.1 Elvior TestCast Tools Family .................................................. 339

17.1.2 Testing Related Services ....................................................... 340

17.2 Insurance Premium Results .......................................................... 341

17.2.1 System (SUT) Modeling ....................................................... 341

17.2.2 Test Coverage and Test Generation ................................... 341

17.3 Garage Door Controller Results ................................................... 345

17.3.1 System (SUT) Modeling ....................................................... 345

17.3.2 Test Coverage and Test Generation ................................... 345

17.4 Vendor Advice .......................................................................... 348

## 18 sepp.med GmbH

18.1 Introduction .................................................................................. 351

18.1.1 About sepp.med .................................................................... 351

18.1.2 About MBTsuite ................................................................. 352

18.1.3 Customer Support ............................................................... 353

18.2 Insurance Premium Results .......................................................... 353

18.2.1 Problem Input ..................................................................... 353
## 18.2.2 Generated Test Cases

### 18.2.2.1 Abstract Test Cases Generated by MBTsuite, Activity Diagram “Insurance Premium”

#### 18.2.2.2 Concrete Test Cases Generated by MBTsuite, Activity Diagram “Insurance Premium”

### 18.2.3 Other Vendor-Provided Analysis

### 18.3 Garage Door Controller Results

#### 18.3.1 Problem Input

#### 18.3.2 Generated Test Cases

### 18.3.2.1 Abstract Test Cases Generated by MBTsuite, State Diagram “Garage Door Controller”

#### 18.3.2.2 Concrete Test Cases Generated by MBTsuite, State Diagram “Garage Door Controller”

### 18.3.3 Other Vendor-Provided Analysis

### 18.4 Vendor Advice

### Reference

## 19 Verified Systems International GmbH

### 19.1 Introduction

#### 19.1.1 The RT-Tester Tool Box

#### 19.1.2 The Model-Based Testing Component RTT-MBT

#### 19.1.2.1 Test Case Generation

#### 19.1.2.2 Test Procedure Generation

### 19.2 Case Study: Insurance Premium Calculation

### 19.3 Case Study: Garage Door Controller

### 19.4 Vendor Advice

### Reference

## 20 Open-Source Model-Based Testing Tools

### 20.1 ModelJUnit 2.5

#### 20.1.1 About ModelJUnit 2.5

#### 20.1.2 Using ModelJUnit 2.5 on the Garage Door Controller

#### 20.1.3 General Comments

### 20.2 Spec Explorer

#### 20.2.1 About Spec Explorer

#### 20.2.2 Using Spec Explorer

#### 20.2.2.1 Implementation

#### 20.2.2.2 The Spec Explorer Model
20.2.2.3 The Coordination File ......................................... 408
20.2.2.4 Tool Execution ............................................. 410
20.2.3 General Comments ............................................ 411

20.3 MISTA ......................................................................... 411
20.3.1 About MISTA ..................................................... 412
20.3.1.1 MISTA Environment ...................................... 412
20.3.1.2 MISTA Capabilities ....................................... 412
20.3.1.3 Learning to Use MISTA ......................... 413
20.3.2 Using MISTA ..................................................... 413
20.3.2.1 The Garage Door Controller .................... 413
20.3.2.2 Test Code Generated by MISTA ............. 414
20.3.2.3 Test Output Generated by MISTA .......... 415
20.3.3 General Comments .............................................. 415

20.4 Auto Focus 3 .............................................................. 416
20.4.1 About Auto Focus 3 ............................................ 416
20.4.2 Using Auto Focus 3 ............................................. 418
20.4.3 General Comments ............................................. 418

20.5 Graphwalker ............................................................ 418
20.5.1 About Graphwalker ........................................... 418
20.5.2 Using Graphwalker ........................................... 419
20.5.3 General Comments ........................................... 420

20.6 fMBT ........................................................................ 421
20.6.1 About fMBT ....................................................... 421
20.6.2 Using fMBT ....................................................... 421
20.6.3 General Comments ........................................... 422

Reference ........................................................................ 422

Index ................................................................................ 423
First, a disclaimer. I use the words “craftsman” and “craftsmanship” in the gender neutral sense. No offense is meant, and I hope none is taken. I believe model-based testing (MBT) can be, and should be, a craft rather than an art. Craftsmanship entails three essential parts: understanding the medium, ability to choose appropriate tools, and the experience to use them well. The relation between tools and craft is interesting—a craftsman can do acceptable work with poor tools, but a novice cannot do good work with excellent tools. This is absolutely true for MBT as a craft.

Other than software testing, my preferred craft is woodworking. As a craft, a woodworker needs to know the medium, which, in this case, is wood. Different woods have different properties, and knowing these lets a woodworker make appropriate choices. Maple is extremely hard and requires very sharp tools. Pine is very soft, and forgiving. My preferred wood is Cherry—it is not as hard as maple or oak, but it has a fine grain and it “works” well. The tool part is obvious—take hand saws; for example, a craftsman would have crosscut and ripping saws, a back saw, a miter box, a scroll saw, and maybe specialized Japanese saws that cut on the pull stroke. Each of these is good for a special purpose, and there is no one saw that fits all of these purposes. But just having tools is not enough. A would be craftsman must know how to use the tools at his or her disposal. This is where experience comes in. Maybe the best historical example of craft is the guild system, with apprentices, journeymen, and master craftsmen. The whole point of that progression was that, to be recognized as a dependable craftsman, an individual had to go through a long, supervised learning process.

How does all this fit with model-based testing (MBT)? And what constitutes a craftsman-like approach to MBT? The medium is the software (or system) to be tested. In a simple distinction, the software may be either transformational or reactive. Just this difference has an implication for choosing the appropriate MBT tool.

The “tools” part of MBT as a craft consists of the models used to describe the software, covered in Part 1 and the products, commercial or open sourced, that generate and possibly execute test cases derived from the model, covered in Part 2. In Part 1 (“Theory of Models for Model-Based Testing”), after an introductory overview, Chapters 2 through 10 present nine models of varying complexity and expressive power. Some of these are well known—flowcharts and decision tables.
Finite state machines receive extended treatment because they are the model most extensively supported by both commercial and open sourced MBT products. Part 2 (The Practice of Model-Based Testing) presents six commercial MBT products and a final chapter that sketches six open source MBT tools.

Communicating the experience was the most challenging part of writing this book. There are two continuing examples—the Insurance Premium Problem is a transformational application and the Garage Door Controller is a reactive (event-driven) example. These two problems are modeled in an educational way in Chapters 2 through 10. They were also given to the six commercial tool vendors to show how their products support the two continuing examples. The whole MBT community agrees that the success of MBT is largely determined by how well the system to be tested is modeled, hence the importance of Chapters 2 through 10.

My father was a tool and die maker, his father and grandfather were Danish cabinet makers, my other grandfather was a painter, and my wife is an excellent cook. These family members all approach their talents as crafts, and they always had pride in their work. I believe that sense of pride helps elevate ordinary work to a craft. My goal for readers of this book is that they (you) can use the material here to become an MBT craftsperson.

Paul C. Jorgensen
Rockford, MI
About the Author

Paul C. Jorgensen, PhD, spent 20 years of his first career in all phases of software development for telephone switching systems. He began his university career in 1986 teaching graduate courses in software engineering at Arizona State University, Tempe, Arizona, and since 1988 at Grand Valley State University, Allendale, Michigan, where he is a full professor. His consulting business, Software Paradigms, hibernates during the academic year, and emerges for a short time in the warmer months. He has served on major COnference on DAta SYstems Languages (CODASYL), Association for Computing Machinery (ACM), and Institute of Electrical and Electronics Engineers (IEEE) standards committees, and in 2012, his university recognized his lifetime accomplishments with its “Distinguished Contribution to a Discipline Award.”

In addition to the fourth edition of his software testing book, *Software Testing: A Craftsman’s Approach*, he is also the author of *Modeling Software Behavior: A Craftsman’s Approach*. He is a coauthor of *Mathematics for Data Processing* (McGraw-Hill, 1970) and *Structured Methods—Merging Models, Techniques, and CASE* (McGraw-Hill, 1993). More recently, Dr. Jorgensen has been involved with the International Software Testing Certification Board (ISTQB) where he is a coauthor of the advanced level syllabi and served as the vice-chair of the ISTQB Glossary Working Group. He was a reviewer for the ISTQB Model-Based Testing syllabus.

Living and working in Italy for three years made him a confirmed “Italophile.” He, his wife Carol, and daughters Kirsten and Katia have visited friends there several times. Paul and Carol have volunteered at the Porcupine School on the Pine Ridge Reservation in South Dakota every summer since 2000. His university email address is jorgensp@gvsu.edu, and when he becomes a professor emeritus in the summer of 2017, he can also be reached at pauljorgensen42@gmail.com.
Chapter 1

Overview of Model-Based Testing

All testing, software, hardware, or in daily life, consists of checking the response that comes from a stimulus. Indeed, one of the early requirements methods focused on stimulus–response pairs. In model-based testing (MBT), we consider models that express, to some extent, the stimuli and responses of a system we wish to test. Some clarifying vocabulary will help our discussion.

There are three generally accepted levels of testing: unit, integration, and system. Each of these has distinct goals and methods. Unit testing occurs at the class or procedure level, integration testing considers sets of interacting units, and system testing occurs at the port boundary of a system under test (SUT). There are test coverage metrics that apply to each of these levels. (For a more complete discussion, see [Jorgensen 2013].) At any of these levels, a test case consists of some identification name and identifier, preconditions for test execution, a sequence (possibly interleaved) of inputs and expected outputs, a place to record observed outputs, postconditions, and a pass/fail judgment.

1.1 Initial Terminology

Definition: A system under test, usually abbreviated as SUT, is a system being tested. A SUT can be a full system of software-controlled hardware, a system of hardware alone, or software alone, or even a system of SUTs. A SUT can also be a single software unit, or a collection of units.
Definition: The port boundary of a system under test is the set of all points at which input stimuli and output responses occur.

Every system, hardware, software, firmware, or some combination of these, has a port boundary. Identifying the “true” port boundary of a SUT is essential to the MBT process. Why “true”? It is easy to confuse user-caused physical events with their electronic recognition (stimuli). In a web-based application, the user interface is likely the location of both system-level inputs and outputs. In an automobile Windshield Wiper Controller, the port boundary typically includes a lever and a dial to determine wiper speeds and a motor that drives the wiper blades. Many examples in this book use a Garage Door Controller (more completely defined later). The port boundary of this system includes devices that send a control signal, safety devices, end-of-track sensors, and a driving motor. The port boundary of a unit is the mechanism by which the unit is activated (a message in object-oriented software, or a procedure call in traditional software).

Definition: A port input event is a stimulus that occurs at the port boundary of a given SUT; similarly, a port output event is a response that occurs at the SUT port boundary.

There is a curious potential point of confusion between developers and testers. Consider port input events. A tester thinks in terms of generating, or causing, inputs to a SUT, whereas a developer thinks more in terms of sensing and acting on them. This dichotomy obviously applies to output events, which are observed or sensed by a tester, and caused or generated by a developer. Part of this confusion is the result of design and development models that were created by the software development community. This becomes a problem for MBT if the underlying model uses the developer viewpoint, but a test case uses the tester viewpoint.

1.2 Events

The following terms are roughly synonymous, but they need clarification: port input event, stimulus, and input; symmetrically, we have the following synonyms: port output event, response, and output. Events occur in “layers”, maybe “sequences” is a better term. For now, consider the Garage Door Controller example (see Section 1.8.2 for a full description), specifically the light beam sensor safety device. When the door is closing, if anything interrupts the light beam (near the floor), the motor immediately stops and reverses direction to open the door. The “event sequence” begins with some physical event, perhaps an animal crossing the path of the beam while the door is closing. When the light beam sensor detects the interruption, it sends a signal to the controller; this is a port input event and is a true electronic signal. The software internal to the controller would consider this to be a logical event.
Port input events may occur in different logical contexts. The physical event of a cat crossing the light beam can occur in several contexts: when the door is open, when it is opening, or when it is closing. The logical event only occurs when the door is closing. Frequently, event contexts are represented as states in some finite state machine (FSM). As we examine various models for the way they support MBT, the ability to represent, and recognize, context sensitive input events will be important. Also this forces attention to the port input devices themselves. Suppose, for example, that a tester wanted to test the light beam sensor, particularly its failure modes. The common device failures are Stuck-at-1 (SA-1) and Stuck-at-0 (SA-0). With a SA-1 failure, the light beam sensor will ALWAYS send a signal, regardless of a physical input event that may or may not occur. Note that it will be impossible to close the garage door with this fault. (See use case EECU-SA-1 in the table given below.) The SA-0 fault is more insidious—the door will not reverse after the physical interruption. I am sure the lawyers will get very upset about SA-0 faults on a safety device. It will be modeled in Chapter 8.

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Light Beam Sensor Stuck-at-1.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case ID</td>
<td>EEUC-SA-1.</td>
</tr>
<tr>
<td>Description</td>
<td>A customer attempts to close an open door with a control device signal. The Light Beam Sensor has a SA-1 fault.</td>
</tr>
<tr>
<td>Preconditions</td>
<td>1. The garage door is open.</td>
</tr>
<tr>
<td></td>
<td>2. The Light Beam Sensor has a SA-1 fault.</td>
</tr>
<tr>
<td><strong>Event Sequence</strong></td>
<td></td>
</tr>
<tr>
<td>Input events</td>
<td>Output events</td>
</tr>
<tr>
<td>1. Control device signal.</td>
<td>2. Start motor down.</td>
</tr>
<tr>
<td>3. Light beam SA-1 fault.</td>
<td>4. Stop and reverse motor.</td>
</tr>
<tr>
<td>5. End of up track reached.</td>
<td>6. Stop motor.</td>
</tr>
<tr>
<td>Postconditions</td>
<td>1. The garage door is open.</td>
</tr>
<tr>
<td></td>
<td>2. The Light Beam Sensor has a SA-1 fault.</td>
</tr>
</tbody>
</table>

Stuck-at faults, and indeed other failure modes, are difficult to anticipate. They may or may not appear in the requirement specification. Even if they do, they are hard to model in many MBT models. We will revisit this in great detail in
Chapter 8. Is it likely that a customer would offer use cases such as EEUC-SA-1? Possibly, based on past experience, but this would be a challenge in an agile development.

### 1.3 Test Cases

There are two fundamental forms of a test case—abstract and real (some parts of the MBT community refer to the latter as “concrete test cases”). An abstract test case can usually be derived from a formal model; what makes it “abstract” is that the inputs are usually expressed as variables. A real (concrete) test case contains actual values of input variables and values of expected output values. Both forms should include pre- and post-conditions.

### 1.4 An Architecture for Test Case Execution

Figure 1.1 sketches a generalized architecture for automated test case execution. It is based on a system my team developed for regression testing of telephone switching systems in the early 1980s. The computer houses the overall test case processor, which controls and observes test case execution. Test cases are expressed in a simple language that can be executed interpretively. The language consists of CAUSE and VERIFY statements that refer to the port boundary of the SUT. A CAUSE statement typically has parameters that refer to port input events and the devices where they occur (these may have additional parameters). Similarly, VERIFY statements refer to expected port output events. In a telephone SUT, we might have the following two statements in a test case:

```plaintext
CAUSE InputDigit(9) on Line12
VERIFY DigitEcho(9) on Line12
```

In these statements, InputDigit refers to a parameterized port input event that occurs on the device named Line12 and a port output event that occurs on the

---

**Figure 1.1** A generic test execution architecture.
same device. The key to making this work is to develop a “harness” that connects the test case processor to the SUT. The harness essentially performs logical-to-physical translations of port input events in a CAUSE statement, and physical-to-logical translations of port output events in a VERIFY statement.

All of this is directed at system level testing. Given the popularity of unit level automatic testing support programs such as the nUnit family, the CAUSE and VERIFY statements are replaced by ASSERT statements that contain both unit level inputs and outputs, thereby replacing the harness needed for system testing. In this book, we ignore integration testing, as there is little MBT tool support for this level. This chapter ends with examples of MBT at both the unit and the system levels. For now, it is important to understand that, for MBT to be successful, the underlying models must refer to both stimuli and responses, whether at the unit or the system level.

1.5 Models for MBT

Software (and system) design models are of two general types—structural or behavioral. In the Unified Modeling Language (UML), the de facto standard, structural models focus on classes, their attributes, methods, and connections among classes (inheritance, aggregation, and association). There are two main behavioral models—Statecharts and activity (or sequence) diagrams. Part 1 of this book presents nine behavioral models: Flowcharts, Decision Tables, Finite State Machines, Petri nets, Event-Driven Petri Nets, Statecharts, Swim Lane Event-Driven Petri Nets, UML (use cases and activity charts), and the Business Process Modeling and Notation (BPMN). Each of these, except Swim Lane Event-Driven Petri Nets [Jorgensen 2015] and BPMN, is explained and illustrated fully in [Jorgensen 2008]. The focus here is the extent to which these models support MBT. One inescapable limitation of MBT is that the derived test cases are only as good as the information in the model from which they are derived. Thus a recurring emphasis in Part 1 is the expressive power as well as the limitations of the various models.

1.6 The ISTQB MBT Extension

This book is intended to coordinate with the MBT extension to the ISTQB Foundation Level Syllabus, which was released in October 2015. The ISTQB (International Software Testing Qualification Board) is a nonprofit organization that, in 2013, had certified more than 336,000 software testers in 100 countries [http://www.istqb.org/]. Part 2 of this book consists of six vendor-supplied chapters describing their commercial products, and the results of their products on the two continuing examples (defined in Section 1.8).
1.7 Forms of Model-Based Testing

There are three commonly identified forms of model-based testing: manual, semi-automated, and fully automated [Utting 2010]. In manual MBT, a model of the SUT is developed and analyzed to identify test cases. For example, if the SUT is modeled by a finite state machine, paths from initial states to final states can be visually identified and converted into test cases. The next step is to apply some selection criterion to choose which test cases should be executed. The selection criterion is most likely some coverage metric. These will then need to be “concretized” (the popular term in MBT circles) to replace abstract terms, such as Personal ID number with a real value, such as “1234.” The final step is to execute the concretized test cases on the SUT. As a side note, Craig Larman [Larman 2001] identifies four levels of Use Cases (more on this in Chapter 9). The third level, Expanded Essential Use Cases, contains abstract variable names; Larman’s fourth level, Real Use Cases, replaces the abstract terms with actual values that can be tested. This is exactly the sense of the concretization process. The early use of tools distinguishes manual from semiautomated MBT. Tools are usually some engine that can execute a suitable model and generate abstract test cases. The next step could be either manual or automated: selecting a set of test cases from among the generated set. In a finite state machine example, possible selection criteria can be those test cases that

1. Cover all states.
2. Cover all transitions.
3. Cover all paths.

The selection process can be automated. In fully automated MBT, the steps of semiautomated MBT are followed by automated test case execution. (More on this in Part 2.)

1.8 Continuing Examples

1.8.1 A Unit Level Problem: Insurance Premium Calculation

Premiums on an automobile insurance policy are computed by cost considerations that are applied to a base rate. The inputs to the calculation are as follows:

1. The base rate ($600).
2. The age of the policy holder (16 $\leq$ age $< 25$; $25 \leq$ age $< 65$; $65 \leq$ age $< 90$).
3. People less than age 16 or more than 90 cannot be insured.
4. The number of “at fault” claims in the past five years (0, 1–3, and 3–10).
5. Drivers with more than 10 at fault claims in the past five years cannot be insured.
6. The reduction for being a goodStudent ($50).
7. The reduction for being a nonDrinker ($75).

The calculation values are shown in Tables 1.1 through 1.3.

### 1.8.2 A System Level Problem: The Garage Door Controller

A system to open a garage door is composed of several components: a drive motor, the garage door wheel tracks with sensors at the open and closed positions, and a control device. In addition, there are two safety features: a laser beam near the

<table>
<thead>
<tr>
<th>Table 1.1</th>
<th>Premium Multiplication Values for Age Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age Ranges</td>
<td>ageMultiplier</td>
</tr>
<tr>
<td>16 &lt;= age &lt; 25</td>
<td>x = 1.5</td>
</tr>
<tr>
<td>25 &lt;= age &lt; 65</td>
<td>x = 1.0</td>
</tr>
<tr>
<td>65 &lt;= age &lt; 90</td>
<td>x = 1.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 1.2</th>
<th>Premium Penalty Values at Fault Claims</th>
</tr>
</thead>
<tbody>
<tr>
<td>“At Fault” Claims in the Past Five Years</td>
<td>claimsPenalty</td>
</tr>
<tr>
<td>0</td>
<td>$0</td>
</tr>
<tr>
<td>1 to 3</td>
<td>$100</td>
</tr>
<tr>
<td>4 to 10</td>
<td>$300</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 1.3</th>
<th>Decision Table for goodStudent and nonDrinker Reductions</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1. goodStudent</td>
<td>T</td>
</tr>
<tr>
<td>c2. nonDrinker</td>
<td>T</td>
</tr>
<tr>
<td>a1. Apply $50 reduction</td>
<td>x</td>
</tr>
<tr>
<td>a2. Apply $75 reduction</td>
<td>x</td>
</tr>
<tr>
<td>a3. Do nothing</td>
<td>—</td>
</tr>
</tbody>
</table>
floor and an obstacle sensor. These latter two devices operate only when the garage door is closing. While the door is closing, if either the light beam is interrupted (possibly by a pet) or if the door encounters an obstacle, the door immediately stops, and then reverses its direction. To reduce the size of models in subsequent chapters, only the light beam sensor is considered. The corresponding analyses for the obstacle sensor are almost identical. When the door is in motion, either closing or opening, and a signal from the control device occurs, the door stops. A subsequent control signal starts the door in the same direction as when it was stopped. Finally, there are sensors that detect when the door has moved to one of the extreme positions, either fully open or fully closed. When either of these occurs, the door stops. Figure 1.2 is a SysML context diagram of the Garage Door Controller.

In most garage door systems, there are several control devices: a digital keyboard mounted outside the door, a separately powered button inside the garage, and possibly several in-car signaling devices. For simplicity, we collapse these redundant signal sources into one device. Similarly, as the two safety devices generate the same response, we will drop consideration of the obstacle sensor and just consider the light beam device.

1.8.3 Additional Examples

There are several other examples that are used to illustrate and compare modeling theory and techniques: these are deliberately chosen to illustrate model-specific features. Table 1.4 describes the utility of various modeling choices on the examples in Table 1.5.
### Table 1.4  Model Choices for the Examples in Table 1.5

<table>
<thead>
<tr>
<th>Model</th>
<th>Flowchart</th>
<th>Decision</th>
<th>Finite State Machines</th>
<th>(Ordinary) Petri nets</th>
<th>Event-Driven Petri Nets</th>
<th>Statecharts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good Choice</td>
<td>WCF</td>
<td>ND</td>
<td>EVM</td>
<td>EVM</td>
<td>EVM</td>
<td>WCF</td>
</tr>
<tr>
<td></td>
<td>IP</td>
<td>WW</td>
<td>RRX</td>
<td>RRX</td>
<td>RRX</td>
<td>IP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Can Work, But...</td>
<td>ND</td>
<td>IP</td>
<td>WW</td>
<td></td>
<td></td>
<td>WW</td>
</tr>
<tr>
<td></td>
<td>EVM</td>
<td>EVM</td>
<td></td>
<td></td>
<td></td>
<td>GDC</td>
</tr>
<tr>
<td></td>
<td>RRX</td>
<td>RRX</td>
<td></td>
<td></td>
<td></td>
<td>PCP</td>
</tr>
<tr>
<td></td>
<td>WW</td>
<td>RRX</td>
<td></td>
<td></td>
<td></td>
<td>GDC</td>
</tr>
<tr>
<td>Poor Choice</td>
<td>PCP</td>
<td>WCF</td>
<td>WCF</td>
<td>WCF</td>
<td>WCF</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GDC</td>
<td>PCP</td>
<td>IP</td>
<td>IP</td>
<td>IP</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ND</td>
</tr>
</tbody>
</table>

#### Table 1.5  Chapter-Specific Supplemental Examples Used in Part 1 Chapters

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Model</th>
<th>Chapter-Specific Supplemental Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Flowcharts</td>
<td>Espresso Vending Machine (EVM), NextDate (ND), Windchill Formula (WCF)</td>
</tr>
<tr>
<td>3</td>
<td>Decision tables</td>
<td>NextDate (ND)</td>
</tr>
<tr>
<td>4</td>
<td>Finite state machines</td>
<td>Railroad Crossing Gate Controller (RRX), Windshield Wiper Controller (WW)</td>
</tr>
<tr>
<td>5</td>
<td>Petri nets</td>
<td>Producer–consumer problem (PCP)</td>
</tr>
<tr>
<td>6</td>
<td>Event-Driven Petri Nets</td>
<td>Windshield Wiper Controller (WW)</td>
</tr>
<tr>
<td>7</td>
<td>Statecharts</td>
<td>Windshield Wiper Controller (WW)</td>
</tr>
<tr>
<td>8</td>
<td>Swim Lane EDPNs</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Unified Modeling Language</td>
<td></td>
</tr>
</tbody>
</table>
1.9 The MBT Surveys

In its manual form, model-based testing has been in use since the 1980s. The advent of open-source MBT tools (primarily in the academic community) expanded the interest in MBT. More recently, the availability of commercial MBT products has brought the technology into the industrial practice. Some of the motivating factors for adopting MBT are highlighted in two surveys conducted by Robert V. Binder. The initial survey of MBT users was in 2011, and a follow-up survey was made in 2014. As this book goes to press, the 2016 survey is in progress. These surveys summarize the hopes and concerns of the early MBT adopters.

Binder highlights the following observations of the 2011 survey [Binder 2012]:

- “MBT usage spans a wide range of application stacks, software processes, application domains, and development organizations.
- MBT is accessible and practical: half of the respondents report becoming minimally proficient with their MBT tool with 80 or fewer hours of training or coaching; 80% with 100 hours or less.
- On average, respondents report MBT reduced escaped bugs by 59%.
- On average, respondents report MBT reduced testing costs by 17%.
- On average, respondents report MBT reduced testing duration by 25%.”

The survey was repeated in 2014, this time two other MBT practitioners, Anne Kramer (see Chapter 18) and Bruno Legeard (see Chapter 14) joined the effort [Binder 2014]. There were exactly 100 responses. The referenced report highlights the following points (quoted directly or paraphrased):

Testing levels

- 77.4% used MBT for system testing.
- 49.5% used MBT for integration testing.
- 40.9% used MBT for acceptance testing.
- 31.2% used MBT for component testing.

Generated artifacts

- 84.2% automated test scripts.
- 56.6% manual test cases.
- 39.5% test data.
- 28.9% other documents.

Biggest benefits

- Test coverage.
- Mastering complexity.
Overview of Model-Based Testing

- Automatic test case generation.
- Reuse of models and model elements.

Biggest limitations

- Tool support.
- Skill availability for MBT.
- Resistance to change.

General observations

- 96% used MBT for functional testing.
- 81% used graphical models.
- 59% modeled behavioral aspects.
- Approximately 80 hours needed to become a proficient user.
- 72% of participants were very likely to continue using MBT.

User expectations for MBT

- 73.4% more efficient test design.
- 86.2% more effective test cases.
- 73.4% manage complexity of system testing.
- 44.7% improve communication.
- 59.6% start test design earlier.

Overall effectiveness of MBT

- 23.6% extremely effective.
- 40.3% moderately effective.
- 23.6% slightly effective.
- 5.6% no effect.
- 1.4% slightly ineffective.
- 2.8% moderately ineffective.
- 2.8% extremely ineffective.

References

[Binder 2012]

[Binder 2014]
[Jorgensen 2008]

[Jorgensen 2013]

[Jorgensen 2015]

[Larman 2001]

[Utting 2010]