BDD in Action

Behavior-Driven Development for the whole software lifecycle

JOHN FERGUSON SMART

MANNING
SHELTER ISLAND
PART 1 FIRST STEPS

Building software that makes a difference 3

1.1 BDD from 50,000 feet 5

1.2 What problems are you trying to solve? 7

Building the software right 7 • Building the right software 9
The knowledge constraint—dealing with uncertainty 10

1.3 Introducing Behavior-Driven Development 12

BDD was originally designed as an improved version of TDD 12
BDD also works well for requirements analysis 14
BDD principles and practices 15

1.4 Benefits of BDD 28

Reduced waste 28 • Reduced costs 29 • Easier and safer changes 29 • Faster releases 29
1.5 Disadvantages and potential challenges of BDD  29

- BDD requires high business engagement and collaboration  29
- BDD works best in an Agile or iterative context  30
- BDD doesn’t work well in a silo  30
- Poorly written tests can lead to higher test-maintenance costs  30

1.6 Summary  30

2 BDD—the whirlwind tour  32

2.1 Introducing the train timetable application  33

2.2 Determining the value proposition of the application  35

2.3 Requirements analysis: discovering and understanding features  35

- Describing features  35
- Breaking features down into stories  37
- Illustrating the stories with examples  38

2.4 Implementation: building and delivering features  38

- Going from examples to acceptance criteria  39
- Setting up Maven and Git  40
- Executable specifications: automating the acceptance criteria  42
- Automated tests: implementing the acceptance criteria  46
- Tests as living documentation  55

2.5 Maintenance  56

2.6 Summary  58

PART 2 WHAT DO I WANT? DEFINING REQUIREMENTS USING BDD  59

3 Understanding the business goals: Feature Injection and related techniques  61

3.1 Introducing Flying High Airlines  63

3.2 Feature Injection  63

- Hunt the value  64
- Inject the features  65
- Spot the examples  65
- Putting it all together  66

3.3 What do you want to achieve? Start with a vision  67

- The vision statement  68
- Using vision statement templates  69

3.4 How will it benefit the business?

- Identify the business goals  70

- Writing good business goals  71
- Show me the money—business goals and revenue  72
- Popping the “why stack”: digging out the business goals  73
- Impact Mapping: a visual approach  76
3.5 Who will benefit? Identify stakeholders and their needs 80
3.6 What do you need to build? Identify capabilities 82
3.7 What features will provide the most ROI?
   The Purpose-Based Alignment Model 84
   Differentiating features 85 • Parity features 86
   Partner features 86 • Minimum impact 86
3.8 Summary 86

4 Defining and illustrating features 87
4.1 What is a “feature”? 88
   Features deliver capabilities 90 • Features can be broken down into more manageable chunks 93 • A feature can be described by one or more user stories 95 • A feature is not a user story 98
   Epics are really big user stories 99 • Not everything fits into a hierarchy 100
4.2 Illustrating features with examples 100
4.3 Real Options: don’t make commitments before you have to 107
   Options have value 108 • Options expire 109
   Never commit early unless you know why 110
4.4 Deliberate Discovery 110
4.5 From examples to working software: the bigger picture 111
4.6 Summary 113

5 From examples to executable specifications 114
5.1 Turning concrete examples into executable scenarios 115
5.2 Writing executable scenarios 119
   A feature file has a title and a description 119 • Describing the scenarios 120 • The “Given ... When ... Then” structure 121
   Ands and buts 122 • Comments 123
5.3 Using tables in scenarios 124
   Using tables in individual steps 125 • Using tables of examples 125
5.4 Expressive scenarios: patterns and anti-patterns 128
   Writing expressive Given steps 129 • Writing expressive When steps 130 • Writing expressive Then steps 131
   Providing background and context 132 • Avoid dependencies between scenarios 133
5.5 Organizing your scenarios using feature files and tags 134
The scenarios go in a feature file 135 • A feature file can contain one or more scenarios 136 • Organizing the feature files 136
Annotating your scenarios with tags 137

5.6 Summary 138

6 Automating the scenarios 140
6.1 Introduction to automating scenarios 143
Step definitions interpret the steps 143 • Keep the step definition methods simple 145

6.2 Implementing step definitions: general principles 146
Installing BDD tools 147 • Implementing step definitions 147
Passing parameters to step implementations 148
Maintaining state between steps 149 • Using table data from step definitions 150 • Implementing example-based scenarios 151
Understanding scenario outcomes 152

6.3 Implementing BDD more effectively with Thucydides 154

6.4 Automating scenarios in Java with JBehave 154
Installing and setting up JBehave 154 • JBehave step definitions 156 • Sharing data between steps 157
Passing tables to steps 158 • Step definitions for tables of examples 158 • Pattern variants 159 • Failures and errors in the scenario outcomes 160

6.5 Automating scenarios in Java using Cucumber-JVM 161
Cucumber-JVM project setup and structure 162 • Cucumber-JVM step definitions 163 • Pattern variants 163 • Passing tables to steps 164 • Step definitions for tables of examples 165
Sharing data between steps 165 • Pending steps and step outcomes 166

6.6 Automating scenarios in Python with Behave 166
Installing Behave 167 • The Behave project structure 167
Behave step definitions 167 • Combining steps 168
Step definitions using embedded tables 168 • Step definitions for tables of examples 169 • Running scenarios in Behave 169

6.7 Automating scenarios in .NET with SpecFlow 170
Setting up SpecFlow 170 • Adding feature files 170
Running scenarios 171 • SpecFlow step definitions 172
Sharing data between steps 173 • Step definitions using example tables 173
CONTENTS

6.8 Automating scenarios in JavaScript with Cucumber-JS

Setting up Cucumber-JS 175 • Writing feature files in Cucumber-JS 175 • Implementing the steps 176

Running the scenarios 177

6.9 Summary 178

PART 3  HOW DO I BUILD IT? CODING THE BDD WAY ...179

7 From executable specifications to rock-solid automated acceptance tests 181

7.1 Writing industrial-strength acceptance tests 183

7.2 Automating your test setup process 185

Initializing the database before each test 186 • Initializing the database at the start of the test suite 186 • Using initialization hooks 186 • Setting up scenario-specific data 190

Using personas and known entities 192

7.3 Separating the what from the how 194

The Business Rules layer describes the expected outcomes 195

The Business Flow layer describes the user’s journey 196

The Technical layer interacts with the system 198

How many layers? 199

7.4 Summary 200

8 Automating acceptance criteria for the UI layer 201

8.1 When and how should you test the UI? 203

The risks of too many web tests 203 • Web testing with headless browsers 204 • How much web testing do you really need? 205

8.2 Automating web-based acceptance criteria using Selenium WebDriver 206

Getting started with WebDriver in Java 207 • Identifying web elements 210 • Interacting with web elements 218

Working with asynchronous pages and testing AJAX applications 219 • Writing test-friendly web applications 221

8.3 Using page objects to make your tests cleaner 222

Introducing the Page Objects pattern 223 • Writing well-designed page objects 227 • Using libraries that extend WebDriver 233

8.4 Summary 235
CONTENTS

9 Automating acceptance criteria for non-UI requirements  236
9.1 Balancing UI and non-UI acceptance tests  238
9.2 When to use non-UI acceptance tests  240
9.3 Types of non-UI automated acceptance tests  243
  Testing against the controller layer  243 • Testing business
  logic directly  247 • Testing the service layer  251
9.4 Defining and testing nonfunctional requirements  255
9.5 Discovering the design  257
9.6 Summary  259

10 BDD and unit testing  260
10.1 BDD, TDD, and unit testing  261
  BDD is about writing specifications, not tests, at all levels  263
  BDD builds on established TDD practices  264 • BDD unit-testing
  tools are there to help  264
10.2 Going from acceptance criteria to
  implemented features  264
  BDD favors an outside-in development approach  264
  Start with a high-level acceptance criterion  266
  Automate the acceptance criteria scenarios  267
  Implement the step definitions  268 • Understand the
  domain model  268 • Write the code you’d like to have  269
  Use the step definition code to specify and implement
  the application code  269 • How did BDD help?  274
10.3 Exploring low-level requirements, discovering design,
  and implementing more complex functionality  274
  Use step definition code to explore low-level design  275
  Working with tables of examples  277 • Discover new classes
  and services as you implement the production code  279
  Implement simple classes or methods immediately  280
  Use a minimal implementation  281 • Use stubs and
  mocks to defer the implementation of more complex code  281
  Expand on low-level technical specifications  283
10.4 Tools that make BDD unit testing easier  285
  Practicing BDD with traditional unit-testing tools  285
  Writing specifications, not tests: the RSpec family  287
  Writing more expressive specifications using Spock
  or Spec2  291
10.5 Using executable specifications as living documentation 293
  Using fluent coding to improve readability 294 • Fluent assertions in JavaScript 295 • Fluent assertions in static languages 295

10.6 Summary 297

PART 4 TAKING BDD FURTHER ..............................299

11 Living Documentation: reporting and project management 301
11.1 Living documentation: a high-level view 302
11.2 Are we there yet? Reporting on feature readiness and feature coverage 304
  Feature readiness: what features are ready to deliver 304
  Feature coverage: what requirements have been built 305
11.3 Integrating a digital product backlog 308
11.4 Organizing the living documentation 309
  Organizing living documentation by high-level requirements 310
  Organizing living documentation using tags 311
  Living documentation for release reporting 312
11.5 Providing more free-form documentation 314
11.6 Technical living documentation 315
  Unit tests as living documentation 316 • Living Documentation for legacy applications 319
11.7 Summary 319

12 BDD in the build process 321
12.1 Executable specifications should be part of an automated build 322
  Each specification should be self-sufficient 322 • Executable specifications should be stored under version control 324
  You should be able to run the executable specifications from the command line 325
12.2 Continuous integration speeds up the feedback cycle 325
12.3 Continuous delivery: any build is a potential release 328
12.4 Continuous integration used to deploy living documentation 330
  Publishing living documentation on the build server 331
  Publishing living documentation to a dedicated web server 332
12.5 Faster automated acceptance criteria 332

Running parallel acceptance tests within your automated build 332 • Running parallel tests on multiple machines 335
Running parallel web tests using Selenium Grid 337

12.6 Summary 342

12.7 Final words 342

index 344