

---

# Table of Contents

Preface.....	ix
--------------	----

Introduction.....	xvii
-------------------	------

---

## Part I. Building an Architecture to Support Domain Modeling

<b>1. Domain Modeling.....</b>	<b>5</b>
What Is a Domain Model?	6
Exploring the Domain Language	9
Unit Testing Domain Models	10
Dataclasses Are Great for Value Objects	15
Value Objects and Entities	17
Not Everything Has to Be an Object: A Domain Service Function	19
Python’s Magic Methods Let Us Use Our Models with Idiomatic Python	20
Exceptions Can Express Domain Concepts Too	20
<b>2. Repository Pattern.....</b>	<b>23</b>
Persisting Our Domain Model	24
Some Pseudocode: What Are We Going to Need?	24
Applying the DIP to Data Access	25
Reminder: Our Model	26
The “Normal” ORM Way: Model Depends on ORM	27
Inverting the Dependency: ORM Depends on Model	28
Introducing the Repository Pattern	31
The Repository in the Abstract	32
What Is the Trade-Off?	33
Building a Fake Repository for Tests Is Now Trivial!	37

What Is a Port and What Is an Adapter, in Python?	37
Wrap-Up	38
<b>3. A Brief Interlude: On Coupling and Abstractions.....</b>	<b>41</b>
Abstracting State Aids Testability	43
Choosing the Right Abstraction(s)	46
Implementing Our Chosen Abstractions	47
Testing Edge to Edge with Fakes and Dependency Injection	49
Why Not Just Patch It Out?	51
Wrap-Up	53
<b>4. Our First Use Case: Flask API and Service Layer.....</b>	<b>55</b>
Connecting Our Application to the Real World	57
A First End-to-End Test	57
The Straightforward Implementation	58
Error Conditions That Require Database Checks	60
Introducing a Service Layer, and Using FakeRepository to Unit Test It	61
A Typical Service Function	63
Why Is Everything Called a Service?	66
Putting Things in Folders to See Where It All Belongs	67
Wrap-Up	68
The DIP in Action	68
<b>5. TDD in High Gear and Low Gear.....</b>	<b>71</b>
How Is Our Test Pyramid Looking?	72
Should Domain Layer Tests Move to the Service Layer?	72
On Deciding What Kind of Tests to Write	73
High and Low Gear	74
Fully Decoupling the Service-Layer Tests from the Domain	75
Mitigation: Keep All Domain Dependencies in Fixture Functions	76
Adding a Missing Service	76
Carrying the Improvement Through to the E2E Tests	78
Wrap-Up	79
<b>6. Unit of Work Pattern.....</b>	<b>81</b>
The Unit of Work Collaborates with the Repository	83
Test-Driving a UoW with Integration Tests	84
Unit of Work and Its Context Manager	85
The Real Unit of Work Uses SQLAlchemy Sessions	86
Fake Unit of Work for Testing	87
Using the UoW in the Service Layer	88
Explicit Tests for Commit/Rollback Behavior	89

Explicit Versus Implicit Commits	90
Examples: Using UoW to Group Multiple Operations into an Atomic Unit	91
Example 1: Relocate	91
Example 2: Change Batch Quantity	91
Tidying Up the Integration Tests	92
Wrap-Up	93
<b>7. Aggregates and Consistency Boundaries.....</b>	<b>95</b>
Why Not Just Run Everything in a Spreadsheet?	96
Invariants, Constraints, and Consistency	96
Invariants, Concurrency, and Locks	97
What Is an Aggregate?	98
Choosing an Aggregate	99
One Aggregate = One Repository	102
What About Performance?	104
Optimistic Concurrency with Version Numbers	105
Implementation Options for Version Numbers	107
Testing for Our Data Integrity Rules	109
Enforcing Concurrency Rules by Using Database Transaction	
Isolation Levels	110
Pessimistic Concurrency Control Example: SELECT FOR UPDATE	111
Wrap-Up	111
Part I Recap	113

---

## Part II. Event-Driven Architecture

<b>8. Events and the Message Bus.....</b>	<b>117</b>
Avoiding Making a Mess	118
First, Let's Avoid Making a Mess of Our Web Controllers	118
And Let's Not Make a Mess of Our Model Either	119
Or the Service Layer!	120
Single Responsibility Principle	120
All Aboard the Message Bus!	121
The Model Records Events	121
Events Are Simple Dataclasses	121
The Model Raises Events	122
The Message Bus Maps Events to Handlers	123
Option 1: The Service Layer Takes Events from the Model and Puts Them on the Message Bus	124
Option 2: The Service Layer Raises Its Own Events	125
Option 3: The UoW Publishes Events to the Message Bus	126

Wrap-Up	130
<b>9. Going to Town on the Message Bus.....</b>	<b>133</b>
A New Requirement Leads Us to a New Architecture	135
Imagining an Architecture Change: Everything Will Be an Event Handler	136
Refactoring Service Functions to Message Handlers	137
The Message Bus Now Collects Events from the UoW	139
Our Tests Are All Written in Terms of Events Too	141
A Temporary Ugly Hack: The Message Bus Has to Return Results	141
Modifying Our API to Work with Events	142
Implementing Our New Requirement	143
Our New Event	143
Test-Driving a New Handler	144
Implementation	145
A New Method on the Domain Model	146
Optionally: Unit Testing Event Handlers in Isolation with a Fake Message Bus	147
Wrap-Up	149
What Have We Achieved?	150
Why Have We Achieved?	150
<b>10. Commands and Command Handler.....</b>	<b>151</b>
Commands and Events	151
Differences in Exception Handling	153
Discussion: Events, Commands, and Error Handling	155
Recovering from Errors Synchronously	158
Wrap-Up	160
<b>11. Event-Driven Architecture: Using Events to Integrate Microservices.....</b>	<b>161</b>
Distributed Ball of Mud, and Thinking in Nouns	162
Error Handling in Distributed Systems	165
The Alternative: Temporal Decoupling Using Asynchronous Messaging	167
Using a Redis Pub/Sub Channel for Integration	168
Test-Driving It All Using an End-to-End Test	169
Redis Is Another Thin Adapter Around Our Message Bus	170
Our New Outgoing Event	171
Internal Versus External Events	172
Wrap-Up	172
<b>12. Command-Query Responsibility Segregation (CQRS).....</b>	<b>175</b>
Domain Models Are for Writing	176
Most Users Aren't Going to Buy Your Furniture	177

Post/Redirect/Get and CQS	179
Hold On to Your Lunch, Folks	181
Testing CQRS Views	182
“Obvious” Alternative 1: Using the Existing Repository	182
Your Domain Model Is Not Optimized for Read Operations	183
“Obvious” Alternative 2: Using the ORM	184
SELECT N+1 and Other Performance Considerations	184
Time to Completely Jump the Shark	185
Updating a Read Model Table Using an Event Handler	186
Changing Our Read Model Implementation Is Easy	188
Wrap-Up	189
<b>13. Dependency Injection (and Bootstrapping).....</b>	<b>191</b>
Implicit Versus Explicit Dependencies	193
Aren’t Explicit Dependencies Totally Weird and Java-y?	194
Preparing Handlers: Manual DI with Closures and Partials	196
An Alternative Using Classes	198
A Bootstrap Script	199
Message Bus Is Given Handlers at Runtime	201
Using Bootstrap in Our Entrypoints	203
Initializing DI in Our Tests	204
Building an Adapter “Properly”: A Worked Example	205
Define the Abstract and Concrete Implementations	206
Make a Fake Version for Your Tests	206
Figure Out How to Integration Test the Real Thing	207
Wrap-Up	209
<b>Epilogue.....</b>	<b>211</b>
<b>A. Summary Diagram and Table.....</b>	<b>229</b>
<b>B. A Template Project Structure.....</b>	<b>231</b>
<b>C. Swapping Out the Infrastructure: Do Everything with CSVs.....</b>	<b>239</b>
<b>D. Repository and Unit of Work Patterns with Django.....</b>	<b>245</b>
<b>E. Validation.....</b>	<b>255</b>
<b>Index.....</b>	<b>265</b>

