


Applying Domain-Driven Design and Patterns

With Examples in C# and .NET

Jimmy Nilsson

 Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Cape Town • Sydney • Tokyo • Singapore • Mexico City

Contents

the Author	xxv
Keywords	xxvii
Foreword: Bridging Gaps	xxxi
I: BACKGROUND	1
Chapter 1: Values to Value	3
Overall Values	3
Architecture Styles to Value	4
Focus on the Model	4
Use Case Focus	5
If You Have a Model Focus, Use the Domain Model Pattern	9
Handle the Database with Care	14
The Impedance Mismatch Between Domain Model and Relational Database	19
Handle Distribution with Care	24
Messaging Focus	26
Process Ingredients to Value	28
Up-Front Architecture Design	29
Domain-Driven Design	32
Test-Driven Development	33
Refactoring	37
Which Ingredient or a Combination?	39
Continuous Integration	40
The Integration Problem	40
The Solution (Or at Least a Big Step in the Right Direction)	41
Lessons Learned in My Organization	41
Further Information	42
Don't Forget About Operations	42
An Example of When a Mechanism Is Needed	42
Some Examples of Operational Mechanisms	44
It's Not Just Our Fault	45
Summary	45

Chapter 2: A Head Start on Patterns.	47
A Little Bit About Patterns.	48
Why Learn About Patterns?.	48
Is There Something to Look Out for Regarding Patterns?.	50
Design Patterns.	52
An Example: State Pattern.	53
Architectural Patterns.	60
An Example: Layers.	61
Another Example: Domain Model Pattern.	62
Design Patterns for Specific Types of Applications.	62
An Example: Query Objects.	63
Domain Patterns.	69
An Example: Factory.	70
Summary.	75
Chapter 3: TDD and Refactoring	77
Test-Driven Development (TDD).	77
The TDD Flow.	78
Time for a Demo.	78
Design Effects.	85
Problems.	88
The Next Phase?.	90
Mocks and Stubs.	90
A Typical Unit Test.	90
Declaration of Independence.	91
Working with Difficult Team Members.	92
Replacing Collaborators with Testing Stubs.	93
Replacing Collaborators with Mock Objects.	95
Design Implications.	97
Consequences.	98
Further Information.	98
Refactoring.	98
Let's Clean Some Smelly Code.	99
Summary.	110



PART II: APPLYING DDD.111
Chapter 4: A New Default Architecture.113
The Basis of the New Default Architecture.113
From Database Focus to Domain Model Focus.115
More Specifically, a DDD Focus.115
Layering According to DDD.116
A First Sketch.117
Problems/Features for Domain Model Example.118
Dealing with Features One by One.120
The Domain Model to This Point.132
Making a First Attempt at Hooking the UI to the Domain Model. .134	
A Basic Goal.134
The Current Focus of the Simple UI.135
List Orders for a Customer.135
Add an Order.136
What Did We Just See?.137
Yet Another Dimension.138
Location of the Domain Model.139
Isolating or Sharing Instances.140
Stateful or Stateless Domain Model Instantiation.141
Complete or Subset Instantiation of the Domain Model.141
Summary.142
Chapter 5: Moving Further with Domain-Driven Design.143
Refining the Domain Model Through Simple TDD	
Experimentation.143
Starting with the Creation of <i>Order</i> and <i>OrderFactory</i>144
Some Domain Logic.148
Second Task: The <i>OrderRepository</i> + <i>OrderNumber</i>150
Reconstituting an Entity from Persistence: How to Set	
Values from the Outside.155
Fetching a List of Orders.160
It's Time to Talk About Entities.161
Back to the Flow Again.162
The Bird's-Eye View.163
Faking the <i>OrderRepository</i>165
A Few Words About Saving.167



Total Amount for Each Order.	167
Historic Customer Information.	172
The Life Cycle of an Instance.	175
Type of Order.	176
Reference Person for an Order.	177
Fluent Interface.	179
Summary.	180
Chapter 6: Preparing for Infrastructure.	181
POCO as a Lifestyle.	182
PI for Our Entities and Value Objects.	183
PI or not PI?.	188
Runtime Versus Compile Time PI.	188
The Cost for PI Entities/Value Objects.	189
PI for Our Repositories.	191
The Cost for Single-Set Repositories.	197
Dealing with Save Scenarios.	198
Reasons for the Decisions.	199
Let's Build the Fake Mechanism.	203
More Features of the Fake Mechanism.	204
The Implementation of the Fake.	205
Affecting the Unit Tests.	207
Database Testing.	211
Reset the Database Before Each Test.	212
Maintain the State of the Database During the Run.	214
Reset the Data Used by a Test Before the Test.	214
Don't Forget Your Evolving Schema!.	215
Separate the Testing of the Unit from the Testing of the Call to the Database.	216
Querying.	219
Single-Set of Query Objects.	220
The Cost for Single-Set of Query Objects.	222
Where to Locate Queries.	224
Aggregates as a Tool Again.	225
Specifications as Queries.	227
Other Querying Alternatives.	228
Summary.	228

Chapter 7: Let the Rules Rule	229
Categorization of Rules	229
Principles for Rules and Their Usage	230
Two-Way Rules Checking: Optional (Possible) to Check Proactively, Mandatory (and Automatic) to Check Reactively	230
All States, Even When in Error, Should be Savable	230
Rules Should Be Productive to Use	231
Rules Should Optionally be Configurable so that You Can Add Custom Rules	231
Rules Should Be Located with State	231
Rules Should be Extremely Testable	232
The System Should Stop You from Getting into a Bad State	232
Starting to Create an API	232
Context, Context, Context!	234
Database Constraints	234
Bind Rules to Transitions Related to the Domain <i>or</i> the Infrastructure?	235
Refining the Principle: "All States, Even when in Error, Should Be Savable"	236
Requirements for a Basic Rules API Related to Persistence	238
Back to the Found API Problems	239
<i>What</i> Was the Problem?	240
We Allowed an Incorrect Transition	240
What If We Forgot to Check?	241
Focus on Domain-Related Rules	241
Rules that Require Cooperation	243
Locating Set-Based Processing Methods	245
Service-Serviced Validation	247
Trying to Transition when We Shouldn't	247
Business ID	249
Avoiding Problems	252
Aggregates as the Tool Again	253
Extending the API	254
Ask for Rules to Be Used to Set Up UI	254
Make It Possible to Inject Rules	254
Refining the Implementation	255
A Naive Implementation	255
Creating Rule Classes—Leaving the Most Naive Stage	261

Setting Up a List of Rules	264
Using the List of Rules	264
Dealing with Sublists	265
An API Improvement	266
Customization	267
Providing the Consumer with Metadata	268
A Problem Suitable for a Pattern?	269
What About the Complex Rules?	269
Binding to the Persistence Abstraction	270
Make the Validation Interface Pluggable	270
Alternative Solution for Approaching the Reactive Validation on Save	271
Reuse Mapping Metadata	272
Generics and Anonymous Methods to the Rescue	273
What Others Have Done	275
Summary	275
PART III: APPLYING POEAA	277
Chapter 8: Infrastructure for Persistence	279
Requirements on the Persistence Infrastructure	280
Where to Store Data	282
RAM	282
File System	285
Object Database	285
Relational Database	287
One or Several Resource Managers?	287
Other Factors	288
Choose and Move On	288
Approach	288
Custom Manual Code	289
Code Generation of Custom Code	290
Metadata Mapping (Object Relational (O/R) Mapper)	291
Choosing Again	293
Classification	294
Domain Model Style	294
Mapper Style	295
Starting Point	295
API Focus	297

Query Language Style	297
Advanced Database Support	298
Other Functionality	300
Another Classification: Infrastructure Patterns	301
Metadata Mapping: Type of Metadata	301
Identity Field	302
Foreign Key Mapping	304
Embedded Value	305
Inheritance Solutions	305
Identity Map	306
Unit of Work	307
Lazy Load/Eager Load	307
Controlling Concurrency	308
Summary	309
Chapter 9: Putting NHibernate into Action	311
Why NHibernate?	311
A Short Introduction to NHibernate	312
Preparations	312
Some Mapping Metadata	314
A Tiny API Example	320
Transactions	322
Requirements of the Persistence Infrastructure	323
High Level of Persistent Ignorant	323
Certain Desirable Features for the Life Cycle of the Persistent Entities	324
Deal Carefully with the Relational Database	326
Classification	328
Domain Model Style	328
Mapper Style	328
Starting Point	329
API Focus	330
Query Language Style	330
Advanced Database Support	331
Other Functionality	333
Another Classification: Infrastructure Patterns	335
Metadata Mapping: Type of Metadata	335
Identity Field	335
Foreign Key Mapping	337



Embedded Value	337
Inheritance Solutions	338
Identity Map	340
Unit of Work	340
Lazy Load/Eager Load	340
Controlling Concurrency	341
Bonus: Validation Hooks	342
NHibernate and DDD	342
Overview of the Assemblies	343
<i>ISession</i> and Repositories	343
<i>ISession</i> , Repositories, and Transactions	344
What Did We Gain?	344
Summary	345
PART IV: WHAT'S NEXT?	347
Chapter 10: Design Techniques to Embrace	349
Context Is King	350
Layers and Partitions	350
Reasons for Partitioning	351
Bounded Context	352
How Do Bounded Contexts and Partitions Relate?	353
Scaling up DDD Projects	353
Why Partition a Domain Model—SO?	353
An Introduction to SOA	354
So What Is SOA Anyway?	354
Why Do We Need SOA?	354
How Is SOA Different?	355
What Is a Service?	355
What Goes in a Service?	356
Where Do the Four Tenets Lead Me?	357
What Is a Service? Take 2	358
The Place of OO in SOA	359
Client-Server and SOA	359
One-Way Asynchronous Messaging	360
How SOA Improves Scalability	361
The Design of a SOA Service	361
How Would a Service Interact with Other Services?	362
SOA and Unavailable Services	365
Complex Messaging Processes	366



Scaling Services	367
Summary	367
Inversion of Control and Dependency Injection	368
No Object Is an Island	368
Factories, Registries, and Service Locators	370
Constructor Dependency Injection	373
Setter Dependency Injection	376
Inversion of Control	377
Dependency Injection with the Spring.NET Framework	378
Auto-Wiring with PicoContainer.NET	380
Nested Containers	382
Service Locator Versus Dependency Injection	384
Summary	385
Aspect-Oriented Programming (AOP)	386
What Is the Buzz About?	387
AOP Terminology Defined	390
AOP in .NET	391
Summary	404
Summary	405
Chapter 11: Focus on the UI	407
A Prelogue	407
The Model-View-Controller Pattern	409
Example: Joe's Shoe Shop	410
Simplifying the View Interfaces Through Adapters	416
Decouple the Controller from the View	417
Combining Views/Controllers	417
Is It Worth It?	418
Test-Driving a Web Form	419
Background	419
An Example	419
Domain Model	420
TDD of GUI	421
The Web Form Implementation	427
Summary	429
Mocking with NMock	429
Mapping and Wrapping	431
Mapping and Wrapping	432
Wrapping the Domain Model with the Presentation Model	433