Updated to cover version 1.3 OMG UML standard

USING UML

SOFTWARE ENGINEERING WITH OBJECTS AND COMPONENTS

PERDITA STEVENS
WITH ROB POOLEY

UPDATED EDITION

Contents

Preface xvi

Acknowledgments xx

PART I Conceptual background 1

Chapter 1 **Software engineering with components** 2
1.1 What is a good system? 2
1.2 Do we have good systems? 3
1.2.1 Problems... 3
1.2.2 ... even drastic failures 4
1.2.3 Promises, promises 5
1.3 What are good systems like? 6
1.3.1 Encapsulation: low coupling 7
1.3.2 Abstraction: high cohesion 10
1.3.3 Architecture and components 11
1.3.4 Component-based design: pluggability 12
1.4 How are good systems built? 13

Chapter 2 **Object concepts** 14
2.1 What is an object? 14
2.1.1 Example 15
2.1.2 Messages 16
2.1.3 Interfaces 17
2.1.4 Classes 17
2.2 How does this relate to the aims of the previous chapter? 20
2.2.1 What have objects to do with components? 21
2.3 Inheritance 22
2.4 Polymorphism and dynamic binding 24

Chapter 3 **Introductory case study** 27
3.1 The problem 27
3.1.1 Clarifying the requirements 27

3.1.2 Use case model 29
3.2 Scope and iterations 31
3.3 Identifying classes 32
3.4 Relations between classes 34
3.5 The system in action 36
Panel 3.1 Design by Contract 1 39
3.5.1 Changes in the system: state diagrams 41
3.5.2 Further work 41
Panel 3.2 Persistence 42

Chapter 4 **The development process** 44
4.1 Defining terms 44
4.1.1 Models and modeling languages 45
4.1.2 Process and quality 46
4.2 The development process 47
4.2.1 A unified methodology? 49
4.2.2 Processes to use with UML 50
4.3 System, design, model, diagram 52

PART II The Unified Modeling Language 55

Chapter 5 **Essentials of class models** 56
5.1 Identifying objects and classes 56
5.1.1 What makes a class model good? 57
5.1.2 How to build a good class model 57
5.1.3 What kinds of things are classes? 59
5.1.4 Real world objects vs their system representation 60
5.2 Associations 60
5.2.1 Multiplicities 63
5.3 Attributes and operations 63
5.3.1 Operations 64
5.3.2 Attributes 64
5.4 Generalization 65
Panel 5.1 Design by Contract 2: substitutivity 66
5.4.1 Using English to check whether a generalization exists 67
5.4.2 Implementing generalization: inheritance 67
5.5 The class model during the development 69
5.6 CRC cards 69
5.6.1 Creating CRC cards 70
5.6.2 Using CRC cards in developing a design 70
5.6.3 CRC card example 71
5.6.4 Refactoring 72

Chapter 6	**More on class models**	**74**
6.1	More about associations	74
	6.1.1 Aggregation and composition	74
	6.1.2 Roles	76
	6.1.3 Navigability	77
	6.1.4 Qualified associations	78
	6.1.5 Derived associations	79
	6.1.6 Constraints	81
Panel 6.1	OCL, the Object Constraint Language	83
	6.1.7 Association classes	84
6.2	More about classes	85
Panel 6.2	Stereotypes	86
	6.2.1 Interfaces	86
	6.2.2 Abstract classes	88
Panel 6.3	Properties and tagged values	89
6.3	Parameterized classes	90
6.4	Dependency	91
6.5	Components and packages	91
6.6	Visibility, protection	92
Chapter 7	**Essentials of use case models**	**93**
7.1	Actors in detail	95
7.2	Use cases in detail	97
7.3	System boundary	98
7.4	Using use cases	99
	7.4.1 Use cases for requirements capture	99
	7.4.2 Use cases through the development	99
7.5	Possible problems with use cases	101
Panel 7.1	Use case driven development?	102
Chapter 8	**More on use case models**	**104**
8.1	Relationships between use cases	104
	8.1.1 Use cases for reuse: ≪include≫	104
	8.1.2 Components and use cases	107
	8.1.3 Separating variant behavior: ≪extend≫	108
8.2	Generalizations	109
8.3	Actors and classes	110
	8.3.1 Notation: actors as classes	111
Chapter 9	**Essentials of interaction diagrams**	**113**
9.1	Collaborations	114
9.2	Interactions on collaboration diagrams	115
9.3	Sequence diagrams	117
Panel 9.1	Where should messages go? Law of Demeter	118
9.4	More advanced features	120
	9.4.1 Messages from an object to itself	120

9.4.2 Suppressing detailed behavior 120
9.4.3 Returned values 122
9.4.4 Creation and deletion of objects 122
9.4.5 Timing 125
9.5 Interaction diagrams for other purposes 126
9.5.1 Show how a class provides an operation 126
9.5.2 Describe how a design pattern works 127
9.5.3 Describe how a component can be used 127

Chapter 10 **More on interaction diagrams** **128**
10.1 Generic interaction diagrams 128
10.1.1 Conditional behavior 129
10.1.2 Iteration 131
10.2 Concurrency 131
10.2.1 Modeling several threads of control 133

Chapter 11 **Essentials of state and activity diagrams** **138**
11.1 State diagrams 139
11.1.1 Unexpected messages 140
11.1.2 Level of abstraction 140
11.1.3 States, transitions, events 141
11.1.4 Actions 142
11.1.5 Guards 143
Panel 11.1 Designing classes with state diagrams 144
11.2 Activity diagrams 146

Chapter 12 **More on state and activity diagrams** **150**
12.1 Other kinds of events 150
12.2 Other kinds of actions 151
12.3 Looking inside states 152
12.4 Concurrency within states 153

Chapter 13 **Implementation diagrams** **155**
13.1 Component model 156
Panel 13.1 Summary: classifiers and instances 158
13.2 Deployment model 159
13.2.1 The physical layer 159
13.2.2 Deploying the software on the hardware 160
Panel 13.2 The deployment model in the project 160

Chapter 14 **Packages, subsystems, models** **162**
14.1 Packages 162
14.1.1 Namespace control 164
14.2 Subsystems 166
14.3 Models 167

PART III Case studies 169

Chapter 15 **CS4 administration** 170
15.1 The case study 170
15.1.1 Class model 174
15.1.2 Dynamics 174
15.1.3 State diagrams 175
15.1.4 Activity diagrams 175
15.2 Discussion 175

Chapter 16 **Board games** 178
16.1 Scope and preliminary analysis 179
16.1.1 Noughts and Crosses (Tic-Tac-Toe) 179
16.1.2 Chess 179
16.2 Interaction 183
16.3 Back to the framework 186
16.4 States 188

Chapter 17 **Discrete event simulation** 190
17.1 Requirements 191
17.1.1 More detailed description 191
17.2 Outline class model 192
17.3 Use cases 193
17.3.1 Summary of `create model` 193
17.3.2 Summary of `observe behavior` 194
17.3.3 Summary of `collect statistics` 194
17.3.4 Summary of `run a model` 196
17.4 Standard mechanism for process based simulation 197
17.5 Associations and navigability 198
17.6 Classes in detail 201
17.6.1 Class `Scheduler` 201
17.6.2 Class `ActiveEntity` 202
17.6.3 Class `PassiveEntity` 203
17.6.4 Class `Resource` 204
17.7 Class `Report` 206
17.8 Class `Statistic` 206
17.8.1 Class `Average` 207
17.9 Building a complete simulation model 207
17.10 The dining philosophers 208

PART IV Towards practice 211

Chapter 18 **Reuse: components, patterns** 212
18.1 Practicalities of reuse 212
18.1.1 What can be reused, and how? 212

Panel 18.1 What is a component really? Controversial! 213
18.1.2 Why reuse? 215
18.1.3 Why is reuse hard? 215
18.1.4 Which components are genuinely reusable? 217
18.1.5 What about building your own components? 217
18.1.6 What difference does object orientation make? 218
18.2 Design patterns 219
18.2.1 Example: Façade 221
18.2.2 UML and patterns 222
18.3 Frameworks 224

Chapter 19 **Product quality: verification, validation, testing** **225**
19.1 Quality review 225
19.2 How can high quality be achieved? 226
19.2.1 Focus on the product 226
19.2.2 Focus on the process 226
19.2.3 Further reading 226
19.3 Verification 227
19.4 Validation 228
19.4.1 Usability 228
19.5 Testing 229
19.5.1 Choosing and carrying out tests 230
19.5.2 Special problems of OO 232
19.5.3 Why is testing so often done badly? 234
19.6 Reviews and inspections 234
19.6.1 Problems of FTRs 235

Chapter 20 **Process quality: management, teams, QA** **237**
20.1 Management 237
20.1.1 Project management 238
20.1.2 Estimating an iterative project 239
20.1.3 Managing component based development 240
20.1.4 People management 241
20.2 Teams 241
20.3 Leadership 242
20.3.1 Reform of development process 244
20.4 Quality assurance 244
20.4.1 Quality assurance for iterative projects 245
20.4.2 Total Quality Management 246
Panel 20.1 Quality assurance: the case against 246
20.5 Further reading 247

Bibliography **249**

Index **252**