

# Object Engineering

---

Designing Large-Scale  
Object-Oriented  
Systems

Gary C. Sullo

---

---

# Object Engineering

Designing Large-Scale, Object-Oriented Systems

Gary C. Sullo

A Wiley–QED Publication



WILEY

**JOHN WILEY & SONS, INC.**

New York • Chichester • Brisbane • Toronto • Singapore

Designations used by companies to distinguish their products are often claimed as trademarks. In all instances where John Wiley & Sons, Inc. is aware of a claim, the product names appear in initial capital or all capital letters. Readers, however, should contact the appropriate companies for more complete information regarding trademarks and registration.

This text is printed on acid-free paper.

Copyright © 1994 by John Wiley & Sons, Inc.

All rights reserved. Published simultaneously in Canada.

This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. It is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional service. If legal advice or other assistance is required, the services of a competent professional person should be sought.

Reproduction or translation of any part of this work beyond that permitted by sections 107 or 108 of the 1976 United States Copyright Act without the permission of the copyright owner is unlawful. Requests for permission or further information should be addressed to the Permission Department, John Wiley & Sons, Inc.

### **Library of Congress Cataloging-in-Publication Data:**

Sullo, Gary, 1953-

Object engineering: designing large-scale, object-oriented systems/Gary Sullo.

p. cm.

Includes index.

ISBN 0-471-62369-5

1. Object-oriented programming (Computer science)

2. System design. I. Title.

QA76.64.S83

1994

005.1'2—dc20

94-1968

CIP

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

---

# Object Engineering

---

# Preface

As a software engineer on real-world projects, like many of you, it seems that I am continually faced with learning new ways of looking at my craft and new ways of applying my skills to build systems and to advise others in doing the same. Structured design was good, and it helped to build better systems (sometimes). Then information engineering was introduced, and it was sometimes better for building other kinds of systems. A lot to learn, but it seemed like this covered it all, and in many ways, it did. What, then, is object-oriented design all about? An object is just another way of looking at a component of a system, like the process modules in structured design or the data entities in information engineering. This is good news. It means that your prior knowledge is useful after all.

Object-oriented programs are said to be easy to use. Very easy. Why then, must their design be so complicated? The answer, of course, is that it doesn't have to be. Surely, something as popular as object-oriented design can be explained without requiring that we totally retrain our professional work force. And that is the purpose of this book—to explain object-oriented design in the context of the things that you already know (which is how we all learn it, anyway) and, as a result, to help you to upgrade your skills with the least amount of pain this time. This shouldn't be the only book that you read on object-oriented design, but it should be one of them.

This book is for those professionals and students of computer science who are familiar with software engineering methodologies and who have been exposed to objects, object languages, and object tools, but who need

to have the smoke cleared away. This will include software engineers, programmers, analysts, and systems integrators. The software engineers and systems integrators who will benefit from reading this book are those who are charged with designing, managing, or integrating large scale industrial strength systems using object-oriented technology. The programmers and analysts who will benefit are those charged with carrying out those projects.

Object engineering doesn't rely on any one particular technology, but rather gives you the understanding of how that technology fits into your project. If you are interested in keeping your skills up to date and need to have all of the new terminology put into a more meaningful perspective, then this book is for you. It will :

- Define the terms used in object-oriented design in a clear and concise manner, so as to bring them together in a meaningful way.
- Relate object-oriented design to concepts borrowed from conventional design, which are more familiar to you anyway.
- Lay out an organized model and a corresponding methodology for object-oriented design which is both comprehensive and yet flexible.
- Provide cross references to the many other object-oriented notations, diagrams, and techniques commonly used in the industry.

Object engineering is a methodology for designing large-scale, object-oriented systems. This book explains *the methodology* in a way that eases your transition into the object-oriented world. The book builds on what you already know about system development. Part I reviews the principles of conventional software design as the context for designing objects. Part II then defines the specific components of an object-oriented design within that context. Finally, Part III provides a layered model for developing that design. The book tells a building story from front to back, but you can also skip around from topic to topic without any loss of continuity. It is, therefore, both a training guide and a reference book.

By approaching the topic in this way, the book helps you to understand that object-oriented design is an evolution, rather than a revolution, in software engineering. This is a unique approach to teaching object-oriented design techniques, and yet, results in a model which is still consistent with the many other diagram notations and design methods that you may encounter elsewhere. Whether you are using Booch, Coad/Yourdon, or some other specific set of diagrams and techniques, and whether you favor CORBA, IEEE, Microsoft, or some other set of object standards, the principles of object engineering should help you to move more quickly and to be more effective in the new world of object-oriented design.

Gary C. Sullo

---

# Object Engineering

---

# Contents

## Preface

**xix**

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	The Purpose of the Book	1
	Making Sense of Objects	1
1.2	The Object-Oriented World	2
1.2.1	What Is Object Engineering?	2
	A Methodology for Object-Oriented Design	2
1.2.2	What Is Object-Oriented Design?	2
	Looking at Data and Processes	2
1.2.3	When Do You Need Object Engineering?	3
	Developing Client/Server Systems	3
1.2.4	What Is a Client/Server System?	3
	Distributed Modules	3
	Large-Scale Systems	4
1.2.5	What Is the Payoff?	4
	Reusable Software Objects	4
1.3	Software Design as a Discipline	5
	Large-Scale Software Design	5
1.4	The Object-Oriented Design Model	5
	Object-Oriented Terminology	5

1.5	The Object-Engineering Methodology	6
	Object-Oriented Methodology	6
<b>PART I</b>	<b>LARGE-SCALE SOFTWARE DESIGN</b>	<b>7</b>
<b>2</b>	<b>Software Engineering</b>	<b>8</b>
2.1	The Software Engineering Model	8
2.1.1	The Concepts of Software Engineering	8
2.2	Development Approaches	10
2.2.1	What Is Life-Cycle Development?	10
	Software-Engineering Principles	10
	Conventional Models	13
	A Framework for Development	14
2.2.2	Why Are Life-Cycle Phases Used?	15
	The Transformation of Requirements	15
2.2.3	How Does This Apply to Objects?	16
	A Similar Framework	16
	Similarities in Approach	17
	Similarities in Techniques	17
2.3	Design Techniques	18
2.3.1	What Is Process-Driven Design?	18
	The Concept of Processes	18
	Designing a Program around Processes	19
2.3.2	What Is Data-Driven Design?	20
	The Concept of Entities	20
	Designing a Program around Entities	21
2.3.3	What Is Object-Oriented Design?	22
	The Concept of Objects	22
	Designing a Program around Objects	26
	Using Processes and Entities	26
<b>3</b>	<b>Conventional Design</b>	<b>29</b>
3.1	Conventional Models	29
3.1.1	What Is the Process-Driven Approach?	29
	The Concept of Process Requirements	29
3.1.2	What Is the Data-Driven Approach?	30
	The Concept of Data Requirements	30
3.1.3	When Is Conventional Design Useful?	32
	Process-Intensive Applications	32
	Data-Intensive Applications	32

3.2	A Process-Driven Approach	33
3.2.1	How Do You Develop a Process-Driven Design?	33
	The Original Approach	33
	The Revised Approach	34
	Process Decomposition	35
3.2.2	How Do You Decompose Processes?	36
	Identifying Process Functions	36
	Allocating the Processes into Groups	38
	Implementing the Process Groups	39
3.2.3	When Is an Object Design Process Driven?	40
	The Concept of Encapsulated Objects	40
	The Concept of Operation Decomposition	41
3.3	A Data-Driven Approach	41
3.3.1	How Do You Develop a Data-Driven Design?	41
	The Business Enterprise	41
	Entity Analysis	43
3.3.2	How Do You Analyze Data Entities?	44
	Identifying Data Entities	44
	Allocating the Entities into Groups	46
	Implementing the Entity Groups	48
3.3.3	When Is an Object Design Data Driven?	50
	The Concept of Classified Object	50
	The Concept of Attribute Analysis	50
<b>4</b>	<b>Object-Oriented Design</b>	<b>54</b>
4.1	Object Requirements	54
	The Concept of Object Requirements	54
4.2	The Client/Server Model	57
4.2.1	What Is a Client/Server Domain?	57
	The Concept of Client Objects	57
	The Concept of Server Objects	59
	The Concept of a Domain	60
4.2.2	What Are Reusable Components?	61
	A Matter of Perspective	61
	The Concept of Abstraction	62
	The Concept of Inheritance	64
	The Concept of Collaboration	67
4.3	Distributed Applications	69
4.3.1	What Is Context-Sensitive Referencing?	69

	The Concept of Polymorphism	69
	The Concept of Visibility	70
	Implied Function Calling	71
	Implied Data Referencing	71
4.3.2	What Is Event-Driven Operation?	72
	The Concept of an Event	72
	Triggering Object Operations	73
	Accessing Object Attributes	74
	Programmed Events	74
	User Events	75
4.3.3	When Is an Application Distributed?	75
	Client/Server Applications	75
	The Concept of Extensibility	76
<b>5</b>	<b>Object Engineering</b>	<b>81</b>
5.1	An Object-Oriented Model	81
	Context-Sensitive Referencing	81
5.2	An Object-Oriented Approach	82
5.2.1	How Do You Develop an Object-Oriented Design?	82
	The Transformation of Requirements	82
	Objects as Operations and Attributes	84
5.2.2	When is Object-Oriented Design Recursive?	84
	The Object-Engineering Approach	84
	Reusing Classes in a Domain	85
	Redesigning Objects in a Class	86
5.3	Object-Oriented Techniques	86
5.3.1	How Do You Identify Object Requirements?	86
	The Class Hierarchies of a Domain	86
	Identifying Inheritance Hierarchies	88
	Identifying Collaboration Hierarchies	88
5.3.2	How Do You Allocate Object Requirements?	89
	Objects Associated with Classes	89
	Allocating Requirements by Classification	91
	Allocating Requirements by Encapsulation	91
5.3.3	How Do You Implement Object Requirements?	92
	Individual Object Design	92



	Objects Linked to Hierarchies	119
	A Group of Related Objects	119
	Abstract Classes of Objects	122
	Concrete Classes of Objects	123
7.1.2	What Is an Instance of a Class?	123
	A Composite Instance of an Object	123
7.1.3	What Is a Member of a Class?	124
	Subclasses of an Abstract Class	124
	Instances of a Concrete Class	124
7.2	Abstract Classes	124
7.2.1	What Is an Abstract Class?	124
	The Result of Classification	124
	A Catalog of Common Parts	125
7.2.2	How Are Abstract Classes Used?	127
	Object-Oriented Representation	127
	The Inheritance Hierarchy	127
7.3	Concrete Classes	128
7.3.1	What Is a Concrete Class?	128
	The Result of Classification	128
	The Result of Encapsulation	128
	Modules of the Operational Design	129
7.3.2	How Are Concrete Classes Used?	131
	Conventional Representation	131
	The Collaboration Hierarchy	132
<b>8</b>	<b>The Definition of Inheritance</b>	<b>134</b>
8.1	The Inheritance Hierarchy	134
8.1.1	What Is Class Inheritance?	134
	An Object-Oriented Hierarchy	134
	Diagrammed Classification	135
	Abstract-Class Interaction	136
	Factoring and Prototyping	137
	Single-Inheritance Hierarchies	138
8.1.2	What Is Multiple Inheritance?	138
	Multiple-Inheritance Hierarchies	138
	Metaclass Inheritance Hierarchies	140
8.1.3	What Does Inheritance Represent?	141
	An <i>Is-a-Kind-of</i> Relationship	141
	A Pattern of Common Objects	142
	Class Categories in a Domain	144
8.2	Object Visibility	145
8.2.1	What Is Object Visibility?	145

	Access to Characteristics	145
	Public Characteristics	146
	Private Characteristics	148
8.2.2	How Is Visibility Used?	148
	The Scope of a Member	148
8.3	Object Types	149
8.3.1	What Is Object Typing?	149
	Class Consistency	149
	Data-Attribute Types	150
	Process-Operation Types	150
8.3.2	How Is Typing Used?	151
	Language Extensions	151
	Strong and Weak Typing	152
	Early and Late Binding	152
<b>9</b>	<b>The Definition of Collaboration</b>	<b>155</b>
9.1	The Collaboration Hierarchy	155
9.1.1	What Is Class Collaboration?	155
	A Conventional Hierarchy	155
	Diagrammed Encapsulation	156
	Concrete Class Interaction	157
9.1.2	What Does Collaboration Represent?	158
	A <i>Makes-Use-of</i> Relationship	158
	A Pattern of Object Operation	159
	Module Assemblies in a Domain	160
	Schema Assemblies in a Domain	161
9.2	Object Requests	162
9.2.1	What Is a Collaboration Contract?	162
	A Package of Requests	162
	Interobject Relationship	164
9.2.2	What Is a Collaboration Request?	166
	Interobject Communication	166
	The Request Stimulus	167
	The Request Response	168
9.2.3	How Are Requests Used?	168
	Hierarchical Encapsulation	168
	Lateral Encapsulation	170
<b>10</b>	<b>The Definition of a Domain</b>	<b>173</b>
10.1	Object Hierarchies	173
10.1.1	What Is a Domain?	173

	A Large-Scale Design	173
	Organized Requirements	174
	Inheritance and Collaboration	175
	Classes and Objects	176
	Operations and Attributes	176
10.1.2	What Is Class Aggregation?	178
	Organization of a Large Domain	178
	An <i>Is-a-Part-of</i> Relationship	178
10.1.3	How Are the Hierarchies Used?	179
	Coding the Inheritance Hierarchy	179
	Coding the Collaboration Hierarchy	181
	Coding the Object Internal Designs	181
	Operational Logic of the Program	182
10.2	The Application Program	182
10.2.1	How Are Operations Triggered?	182
	Common Subroutine Inheritance	182
	Local Subroutine Collaboration	183
	The Concept of Concurrency	184
10.2.2	How Are Attributes Accessed?	186
	Common Data Inheritance	186
	Local Data Collaboration	187
	The Concept of Persistence	188

## **PART III OBJECT-ORIENTED METHODOLOGY 193**

<b>11</b>	<b>The Object-Engineering Model</b>	<b>194</b>
11.1	The Model Composition	194
	An Object-Oriented Approach	194
11.2	The Domain Model	196
11.2.1	What Is a Domain Model?	196
	A Domain Perspective	196
	The Inheritance Diagram	198
	The Collaboration Diagram	198
11.2.2	How Do You Design a Domain?	199
	Identify Objects in a Domain	199
	The Domain Provides	
	Polymorphism	201
11.2.3	How Do You Use the Domain?	201
	Inheritance Polymorphism	201
	Collaboration Polymorphism	202
11.3	The Interface Model	202
11.3.1	What Is an Interface Model?	202

	A Class Perspective	202
	The Class Descriptions	203
	The Object Descriptions	203
	The Request Descriptions	204
11.3.2	How Do You Design Classes?	205
	Allocate Object Requirements	205
	The Classes Provide Extensibility	206
11.3.3	How Do You Use the Classes?	207
	Static Source Libraries	207
	Dynamic Link Libraries	208
11.4	The Implementation Model	208
11.4.1	What Is an Implementation Model?	208
	An Object Perspective	208
	The Operation Descriptions	209
	The Attribute Descriptions	210
11.4.2	How Do You Design Objects?	210
	Implement Individual Objects	210
	The Objects Provide Source Code	213
11.4.3	How Do You Use the Objects?	213
	An Object's Range Is Determined by Its Type	213
	An Object's Scope Is Called Its Visibility	214
	An Object's Extent Is Called Its Persistence	214
	Object Instances Depend on Concurrency	215
<b>12</b>	<b>Design at the Domain Layer</b>	<b>217</b>
12.1	Identifying Objects in a Domain	217
12.1.1	What Does a Design Represent?	217
	The Domain Model	217
12.1.2	How Do You Start Your Design?	219
	Designing a New Domain	219
	Modifying an Existing Domain	220
12.2	The Inheritance Diagram	221
12.2.1	How Do You Determine Inheritance?	221
	A Hierarchy of Common Requirements	221
	The Classification of Requirements	223
	Diagram the Inheritance Relationships	224
12.2.2	How Do You Diagram Inheritance?	225
	Draw the Abstract Classes	225

	Draw the Concrete Classes	226
	Draw the Inheritance Lines	226
12.2.3	How Do You Organize the Diagrams?	227
	Define Class Categories	227
	Define Utility Classes	227
	Define Metaclasses	228
12.3	The Collaboration Diagram	228
12.3.1	How Do You Determine Collaboration?	228
	A Hierarchy of Operational Requirements	228
	The Encapsulation of Requirements	230
	Diagram the Collaboration	
	Relationships	232
12.3.2	How Do You Diagram Collaboration?	233
	Draw the Concrete Classes	233
	Draw the Collaboration Lines	234
12.3.3	How Do You Nest the Diagrams?	235
	Draw Separate Hardware Diagrams	235
	Draw Separate Software Diagrams	235
<b>13</b>	<b>Design at the Class Layer</b>	<b>237</b>
13.1	Allocating Object Requirements	237
	The Interface Model	237
13.2	The Class Descriptions	239
13.2.1	How Do You Determine Classes?	239
	The Interaction between Objects	239
	Inheritance Defines Abstract Classes	241
	Encapsulation Defines Collaboration	243
13.2.2	How Do You Define Classes?	243
	Name the Classes	243
	Identify the Superior Classes	244
	Identify the Contracts	245
	Signify the Concurrency	246
	Signify the Persistence	247
13.3	The Object Descriptions	248
13.3.1	How Do You Determine Objects?	248
	The Requirements of Objects	248
13.3.2	How Do You Define Objects?	249
	Name the Objects	249
	Identify the Operations	251
	Identify the Attributes	252
13.4	The Request Descriptions	252