
Table of Contents

Preface.....	ix
1. The Art of Software Design.....	1
Guideline 1: Understand the Importance of Software Design	2
Features Are Not Software Design	2
Software Design: The Art of Managing Dependencies and Abstractions	3
The Three Levels of Software Development	5
The Focus on Features	9
The Focus on Software Design and Design Principles	10
Guideline 2: Design for Change	11
Separation of Concerns	11
An Example of Artificial Coupling	12
Logical Versus Physical Coupling	15
Don't Repeat Yourself	18
Avoid Premature Separation of Concerns	23
Guideline 3: Separate Interfaces to Avoid Artificial Coupling	24
Segregate Interfaces to Separate Concerns	24
Minimizing Requirements of Template Arguments	27
Guideline 4: Design for Testability	28
How to Test a Private Member Function	28
The True Solution: Separate Concerns	32
Guideline 5: Design for Extension	35
The Open-Closed Principle	35
Compile-Time Extensibility	39
Avoid Premature Design for Extension	41
2. The Art of Building Abstractions.....	43
Guideline 6: Adhere to the Expected Behavior of Abstractions	44

An Example of Violating Expectations	44
The Liskov Substitution Principle	47
Criticism of the Liskov Substitution Principle	51
The Need for Good and Meaningful Abstractions	52
Guideline 7: Understand the Similarities Between Base Classes and Concepts	52
Guideline 8: Understand the Semantic Requirements of Overload Sets	56
The Power of Free Functions: A Compile-Time Abstraction Mechanism	56
The Problem of Free Functions: Expectations on the Behavior	59
Guideline 9: Pay Attention to the Ownership of Abstractions	62
The Dependency Inversion Principle	62
Dependency Inversion in a Plug-In Architecture	69
Dependency Inversion via Templates	72
Dependency Inversion via Overload Sets	72
Dependency Inversion Principle Versus Single-Responsibility Principle	74
Guideline 10: Consider Creating an Architectural Document	74
3. The Purpose of Design Patterns.....	79
Guideline 11: Understand the Purpose of Design Patterns	80
A Design Pattern Has a Name	80
A Design Pattern Carries an Intent	81
A Design Pattern Introduces an Abstraction	82
A Design Pattern Has Been Proven	84
Guideline 12: Beware of Design Pattern Misconceptions	85
Design Patterns Are Not a Goal	85
Design Patterns Are Not About Implementation Details	86
Design Patterns Are Not Limited to Object-Oriented Programming or Dynamic Polymorphism	89
Guideline 13: Design Patterns Are Everywhere	92
Guideline 14: Use a Design Pattern's Name to Communicate Intent	97
4. The Visitor Design Pattern.....	101
Guideline 15: Design for the Addition of Types or Operations	102
A Procedural Solution	102
An Object-Oriented Solution	108
Be Aware of the Design Choice in Dynamic Polymorphism	111
Guideline 16: Use Visitor to Extend Operations	112
Analyzing the Design Issues	113
The Visitor Design Pattern Explained	114
Analyzing the Shortcomings of the Visitor Design Pattern	118
Guideline 17: Consider <code>std::variant</code> for Implementing Visitor	122
Introduction to <code>std::variant</code>	122
Refactoring the Drawing of Shapes as a Value-Based, Nonintrusive Solution	125

Performance Benchmarks	130
Analyzing the Shortcomings of the <code>std::variant</code> Solution	132
Guideline 18: Beware the Performance of Acyclic Visitor	133
5. The Strategy and Command Design Patterns.....	139
Guideline 19: Use Strategy to Isolate How Things Are Done	140
Analyzing the Design Issues	142
The Strategy Design Pattern Explained	146
Analyzing the Shortcomings of the Naive Solution	150
Comparison Between Visitor and Strategy	156
Analyzing the Shortcomings of the Strategy Design Pattern	157
Policy-Based Design	159
Guideline 20: Favor Composition over Inheritance	162
Guideline 21: Use Command to Isolate What Things Are Done	165
The Command Design Pattern Explained	165
The Command Design Pattern Versus the Strategy Design Pattern	172
Analyzing the Shortcomings of the Command Design Pattern	175
Guideline 22: Prefer Value Semantics over Reference Semantics	176
The Shortcomings of the GoF Style: Reference Semantics	176
Reference Semantics: A Second Example	180
The Modern C++ Philosophy: Value Semantics	182
Value Semantics: A Second Example	184
Prefer to Use Value Semantics to Implement Design Patterns	186
Guideline 23: Prefer a Value-Based Implementation of Strategy and Command	186
Introduction to <code>std::function</code>	186
Refactoring the Drawing of Shapes	189
Performance Benchmarks	193
Analyzing the Shortcomings of the <code>std::function</code> Solution	194
6. The Adapter, Observer, and CRTP Design Patterns.....	197
Guideline 24: Use Adapters to Standardize Interfaces	198
The Adapter Design Pattern Explained	199
Object Adapters Versus Class Adapters	201
Examples from the Standard Library	202
Comparison Between Adapter and Strategy	204
Function Adapters	205
Analyzing the Shortcomings of the Adapter Design Pattern	206
Guideline 25: Apply Observers as an Abstract Notification Mechanism	209
The Observer Design Pattern Explained	210
A Classic Observer Implementation	211
An Observer Implementation Based on Value Semantics	221

Analyzing the Shortcomings of the Observer Design Pattern	223
Guideline 26: Use CRTP to Introduce Static Type Categories	225
A Motivation for CRTP	225
The CRTP Design Pattern Explained	230
Analyzing the Shortcomings of the CRTP Design Pattern	236
The Future of CRTP: A Comparison Between CRTP and C++20 Concepts	238
Guideline 27: Use CRTP for Static Mixin Classes	241
A Strong Type Motivation	241
Using CRTP as an Implementation Pattern	243
7. The Bridge, Prototype, and External Polymorphism Design Patterns.....	249
Guideline 28: Build Bridges to Remove Physical Dependencies	250
A Motivating Example	250
The Bridge Design Pattern Explained	254
The Pimpl Idiom	258
Comparison Between Bridge and Strategy	262
Analyzing the Shortcomings of the Bridge Design Pattern	264
Guideline 29: Be Aware of Bridge Performance Gains and Losses	266
The Performance Impact of Bridges	266
Improving Performance with Partial Bridges	269
Guideline 30: Apply Prototype for Abstract Copy Operations	272
A Sheep-ish Example: Copying Animals	272
The Prototype Design Pattern Explained	274
Comparison Between Prototype and <code>std::variant</code>	277
Analyzing the Shortcomings of the Prototype Design Pattern	278
Guideline 31: Use External Polymorphism for Nonintrusive Runtime Polymorphism	279
The External Polymorphism Design Pattern Explained	280
Drawing of Shapes Revisited	283
Comparison Between External Polymorphism and Adapter	291
Analyzing the Shortcomings of the External Polymorphism Design Pattern	291
8. The Type Erasure Design Pattern.....	297
Guideline 32: Consider Replacing Inheritance Hierarchies with Type Erasure	298
The History of Type Erasure	298
The Type Erasure Design Pattern Explained	301
An Owning Type Erasure Implementation	303
Analyzing the Shortcomings of the Type Erasure Design Pattern	311
Comparing Two Type Erasure Wrappers	312
Interface Segregation of Type Erasure Wrappers	315
Performance Benchmarks	316
A Word About Terminology	317

Guideline 33: Be Aware of the Optimization Potential of Type Erasure	318
Small Buffer Optimization	319
Manual Implementation of Function Dispatch	328
Guideline 34: Be Aware of the Setup Costs of Owning Type Erasure Wrappers	333
The Setup Costs of an Owning Type Erasure Wrapper	333
A Simple Nonowning Type Erasure Implementation	336
A More Powerful Nonowning Type Erasure Implementation	338
9. The Decorator Design Pattern.....	347
Guideline 35: Use Decorators to Add Customization Hierarchically	348
Your Coworkers' Design Issue	348
The Decorator Design Pattern Explained	353
A Classic Implementation of the Decorator Design Pattern	356
A Second Decorator Example	360
Comparison Between Decorator, Adapter, and Strategy	363
Analyzing the Shortcomings of the Decorator Design Pattern	364
Guideline 36: Understand the Trade-off Between Runtime and Compile Time	
Abstraction	367
A Value-Based Compile Time Decorator	367
A Value-Based Runtime Decorator	372
10. The Singleton Pattern.....	379
Guideline 37: Treat Singleton as an Implementation Pattern, Not a Design	
Pattern	380
The Singleton Pattern Explained	380
Singleton Does Not Manage or Reduce Dependencies	383
Guideline 38: Design Singletons for Change and Testability	385
Singletons Represent Global State	386
Singletons Impede Changeability and Testability	387
Inverting the Dependencies on a Singleton	391
Applying the Strategy Design Pattern	396
Moving Toward Local Dependency Injection	400
11. The Last Guideline.....	405
Guideline 39: Continue to Learn About Design Patterns	405
Index.....	409