OBJECT ORIENTED DESIGN

SOLID

SINGLE RESPONSIBILITY PRINCIPLE

EVERY CLASS SHOULD HAVE A SINGLE RESPONSIBILITY. THERE SHOULD NEVER BE MORE THAN ONE REASON FOR A CLASS TO CHANGE.

- SMALLER CLASSES
- AVOID GOD CLASSES

SINGLE RESPONSIBILITY PRINCIPLE

Just Because You Can, Doesn't Mean You Should

OPEN/CLOSED PRINCIPLE

SOFTWARE ENTITIES SHOULD BE OPEN FOR EXTENSION, BUT CLOSED FOR MODIFICATION.

- AN ENTITY CAN ALLOW ITS BEHAVIOUR TO BE EXTENDED WITHOUT MODIFYING ITS SOURCE CODE

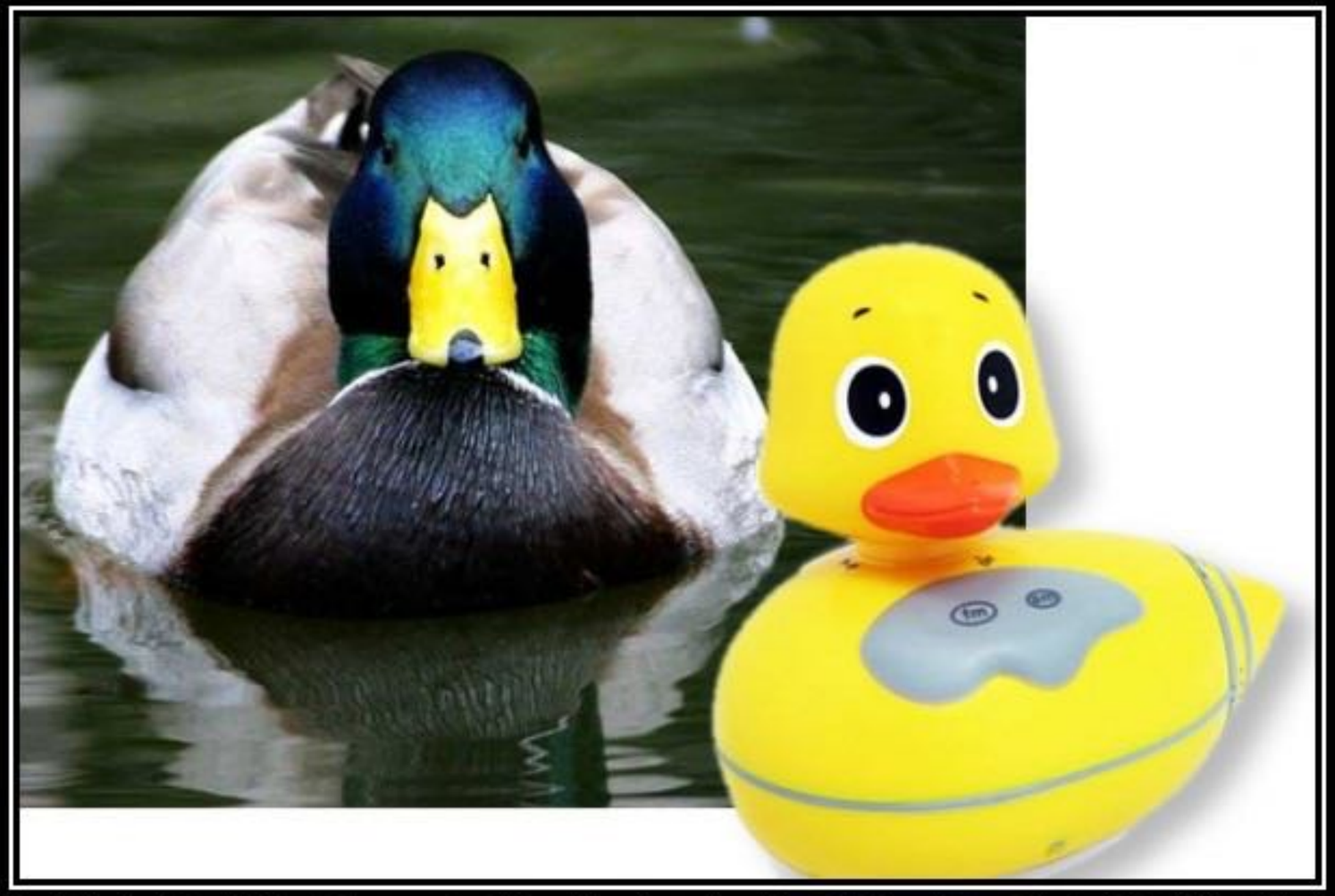
OPEN CLOSED PRINCIPLE

Open Chest Surgery Is Not Needed When Putting On A Coat

LISKOV SUBSTITUTION PRINCIPLE

OBJECTS IN A PROGRAM SHOULD BE REPLACEABLE WITH INSTANCES OF THEIR SUBTYPES WITHOUT ALTERING THE CORRECTNESS OF THE PROGRAM.

- VIOLATION: SQUARE IS A SUBCLASS OF RECTANGLE



LISKOV SUBSTITUTION PRINCIPLE

If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction

INTERFACE SEGREGATION PRINCIPLE

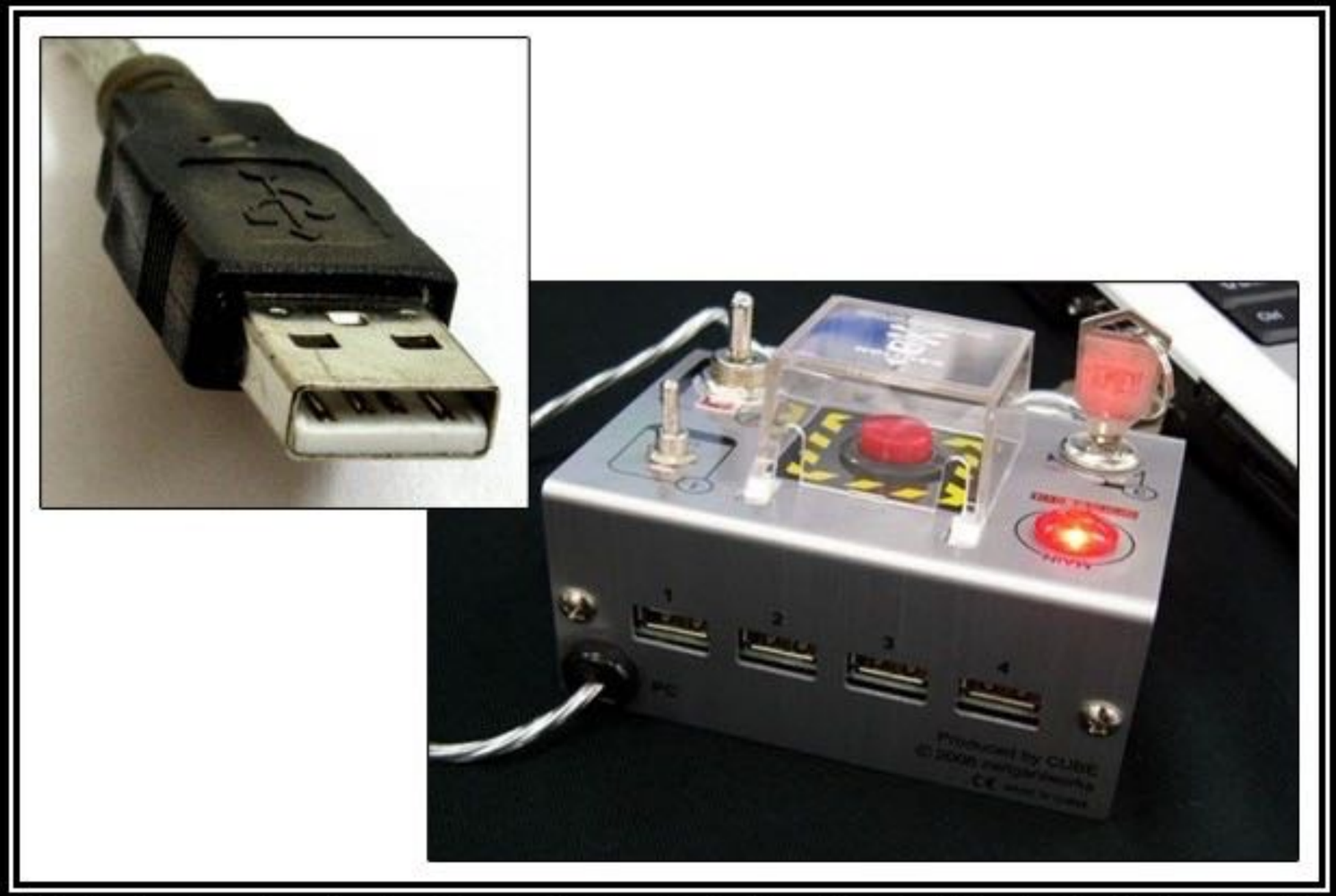
MANY CLIENT-SPECIFIC INTERFACES ARE BETTER THAN ONE GENERAL-PURPOSE INTERFACE.

- **LOW COUPLING**

DEGREE OF INTERDEPENDENCE BETWEEN SOFTWARE MODULES

- **HIGH COHESION**

DEGREE TO WHICH THE ELEMENTS OF A MODULE BELONG TOGETHER



INTERFACE SEGREGATION PRINCIPLE

You Want Me To Plug This In, Where?

DEPENDENCY INVERSION PRINCIPLE

A. HIGH-LEVEL MODULES SHOULD NOT DEPEND ON LOW-LEVEL MODULES. BOTH SHOULD DEPEND ON ABSTRACTIONS.
B. ABSTRACTIONS SHOULD NOT DEPEND ON DETAILS. DETAILS SHOULD DEPEND ON ABSTRACTIONS.

USE THE SAME LEVEL OF ABSTRACTION AT A GIVEN LEVEL

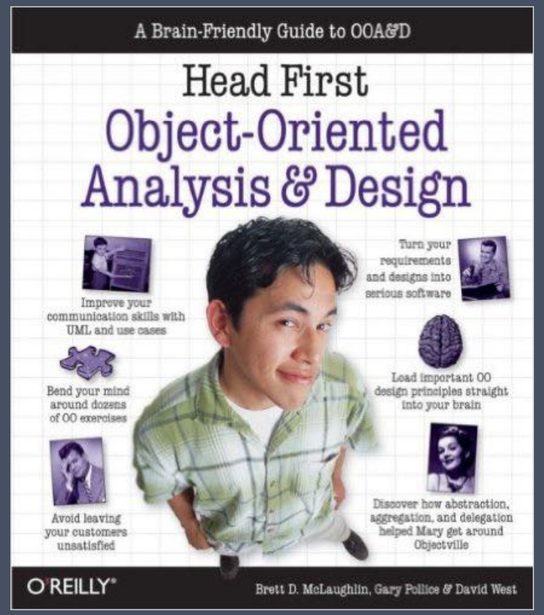
ALSO, RATHER THAN WORKING WITH CLASSES THAT ARE TIGHT COUPLED, USE INTERFACES. THIS IS CALLED PROGRAMMING TO THE INTERFACE. THIS REDUCES DEPENDENCY ON IMPLEMENTATION

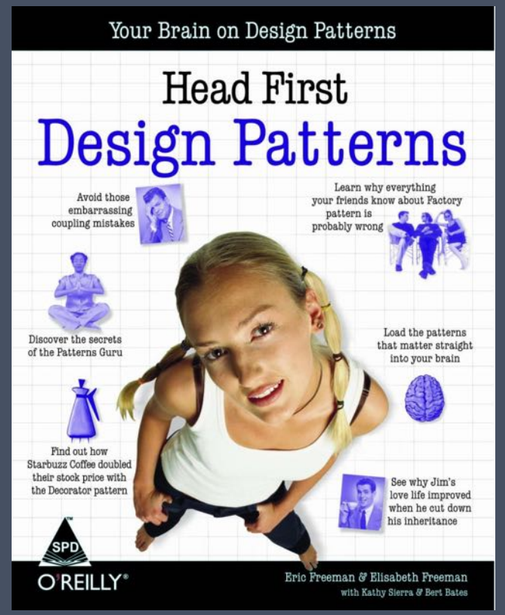
DEPENDENCY INVERSION PRINCIPLE

Would You Solder A Lamp Directly To The Electrical Wiring In A Wall?

RESOURCES

HEAD FIRTS OBJECT-ORIENTED ANALYSIS AND DESIGN



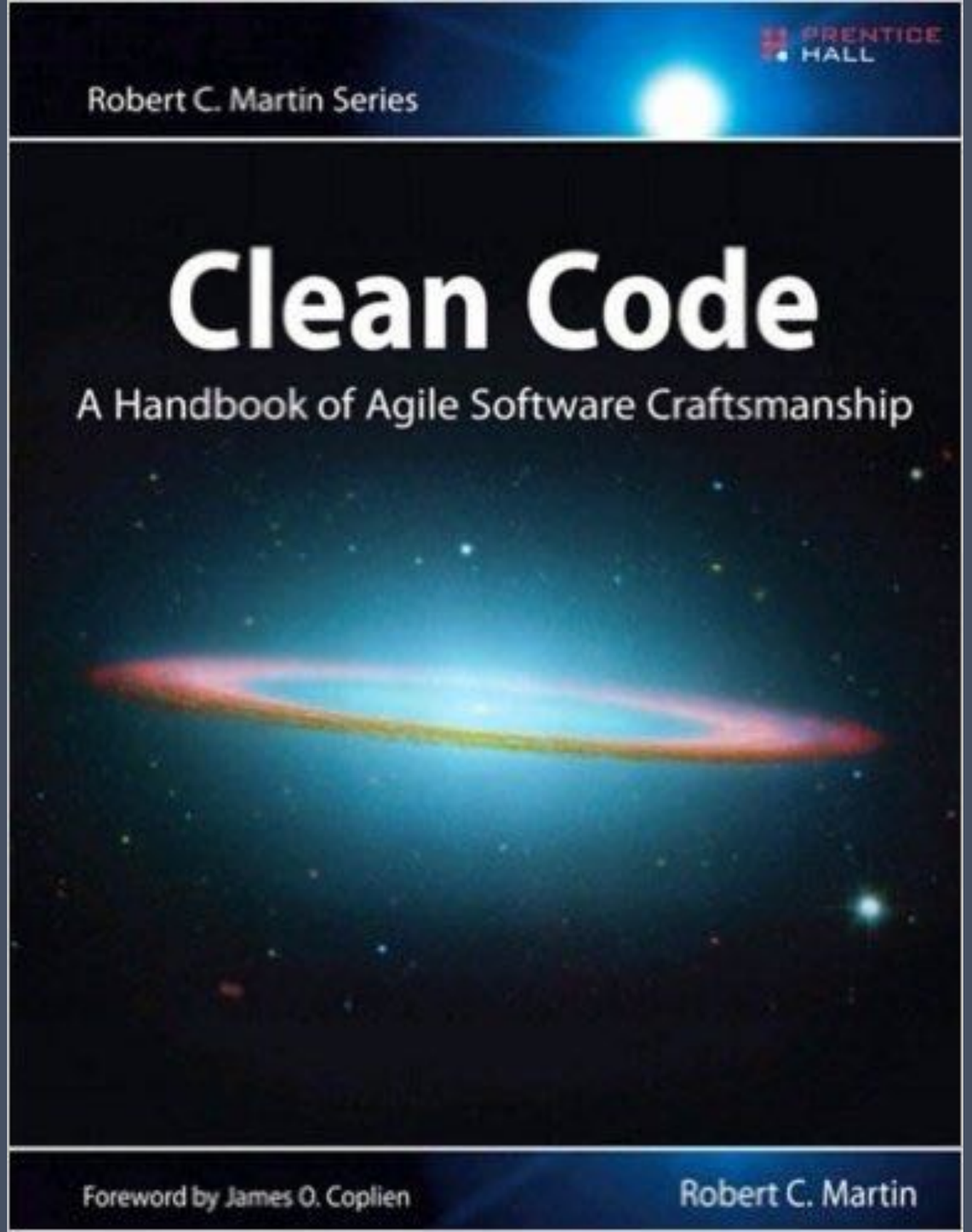


HEAD FIRST DESIGN PATTERNS

CLEAN CODE

AVOID UGLY CODE

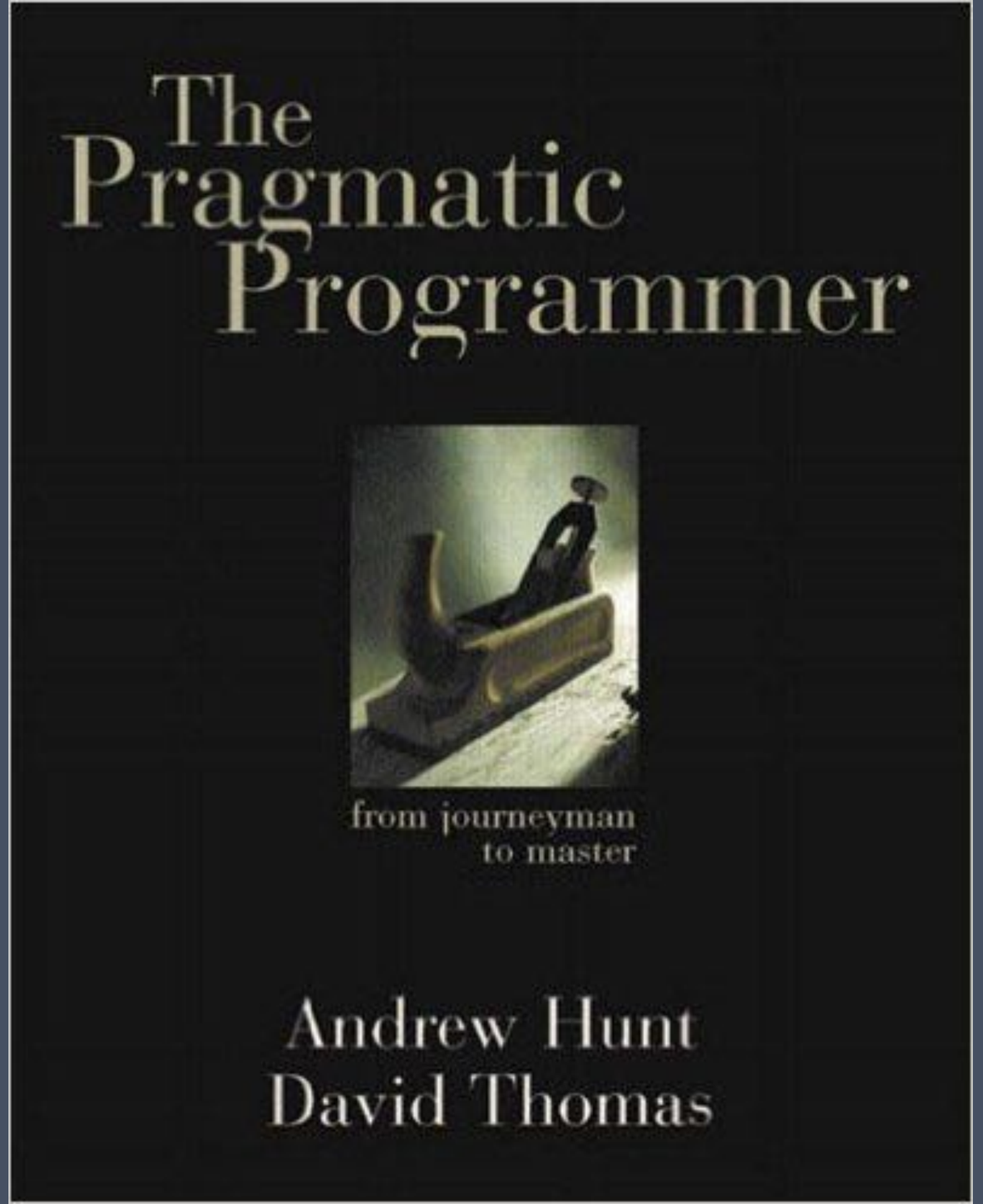
- NAMING
- FUNCTIONS
- COMMENTS
- FORMATING



PRAGMATIC PROGRAMMER

HOW TO WORK AS A PROGRAMMER.

- RESPONSIBILY
  - QUALITY
  - ESTIMATING

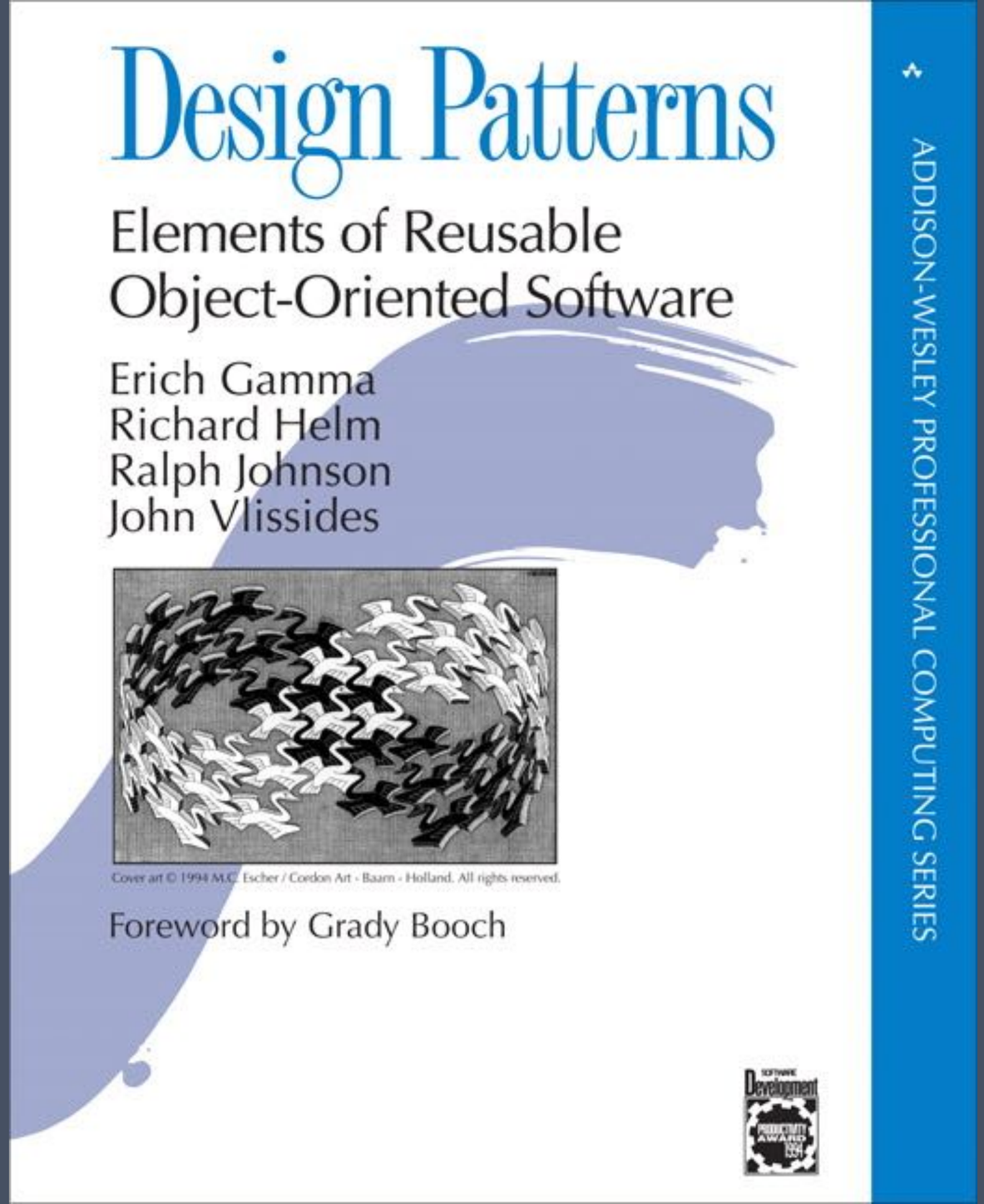


DESIGN PATTERNS

ELEMENTS OF REUSEABLE OBJECT-ORIENTED SOFTWARE

IMPORTANT AND RECURRING DESIGNS IN OO SYSTEMS

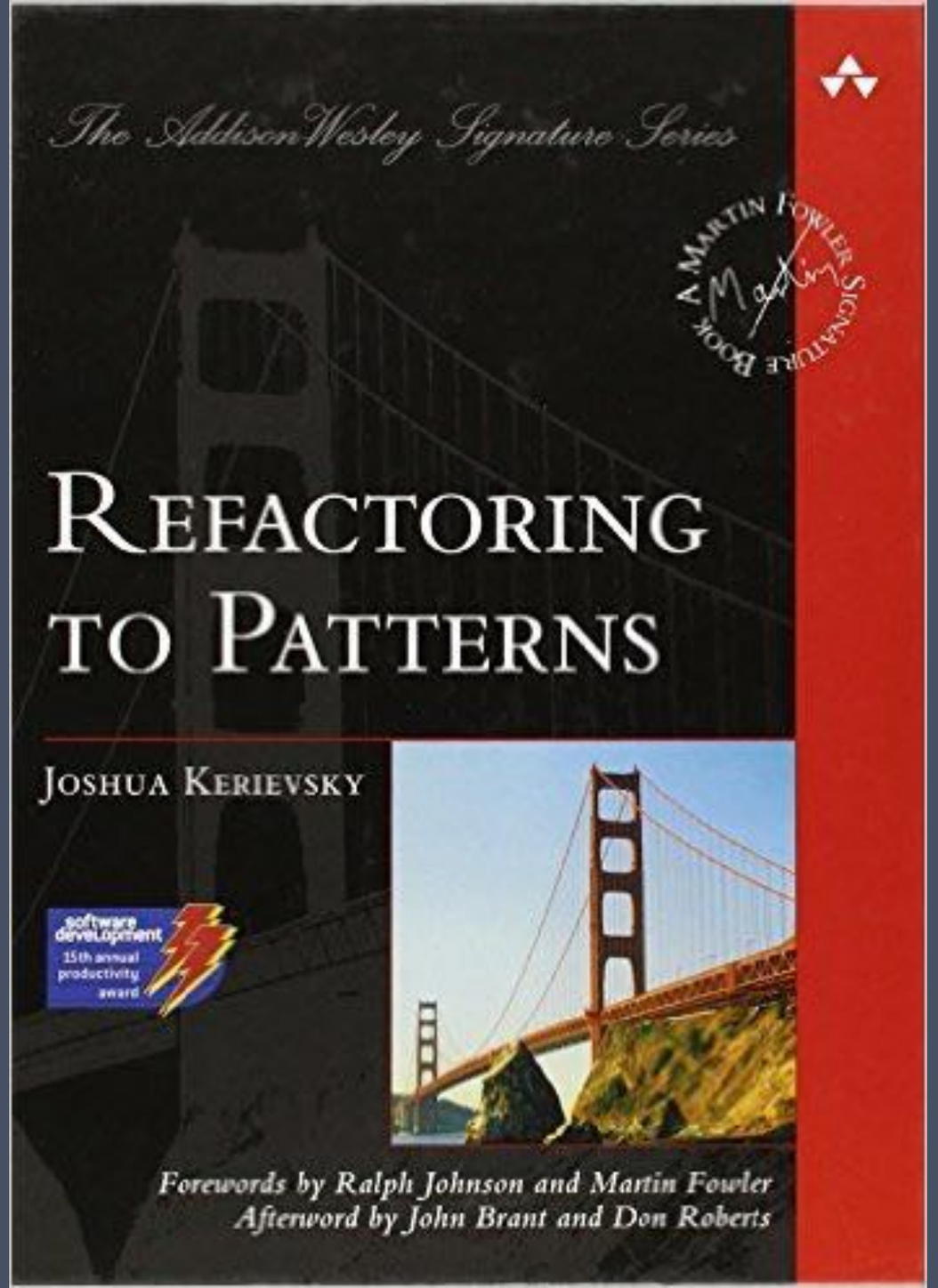
- CREATIONAL PATTERNS
- STRUCTURAL PATTERNS
- BEHAVIORAL PATTERNS



REFACTORING TO PATTERNS

IMPROVE YOUR CODE WITH THE CLASSIC SOLUTIONS TO RECURRING DESIGN PROBLEMS.

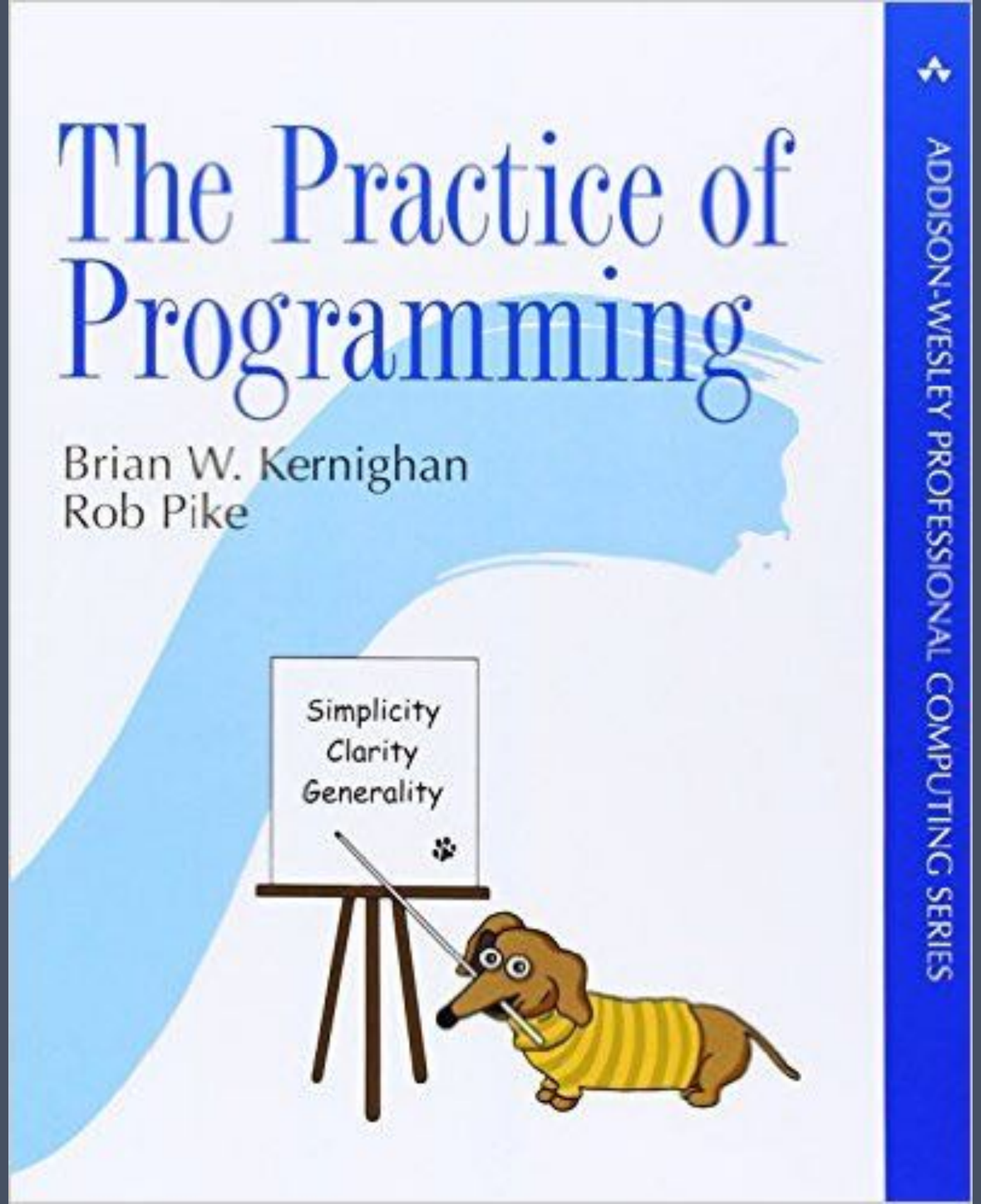
- CODE SMELLS
- MOVE CREATION KNOWLEDGE TO FACTORY



THE PRACICE OF PROGRAMMING

RECOMMENDATION BY JOAO

- DEBUGGING
- TESTING
- PERFORMANCE
- PORTABILITY
- DESIGN
- INTERFACES
- STYLE
- NOTATION

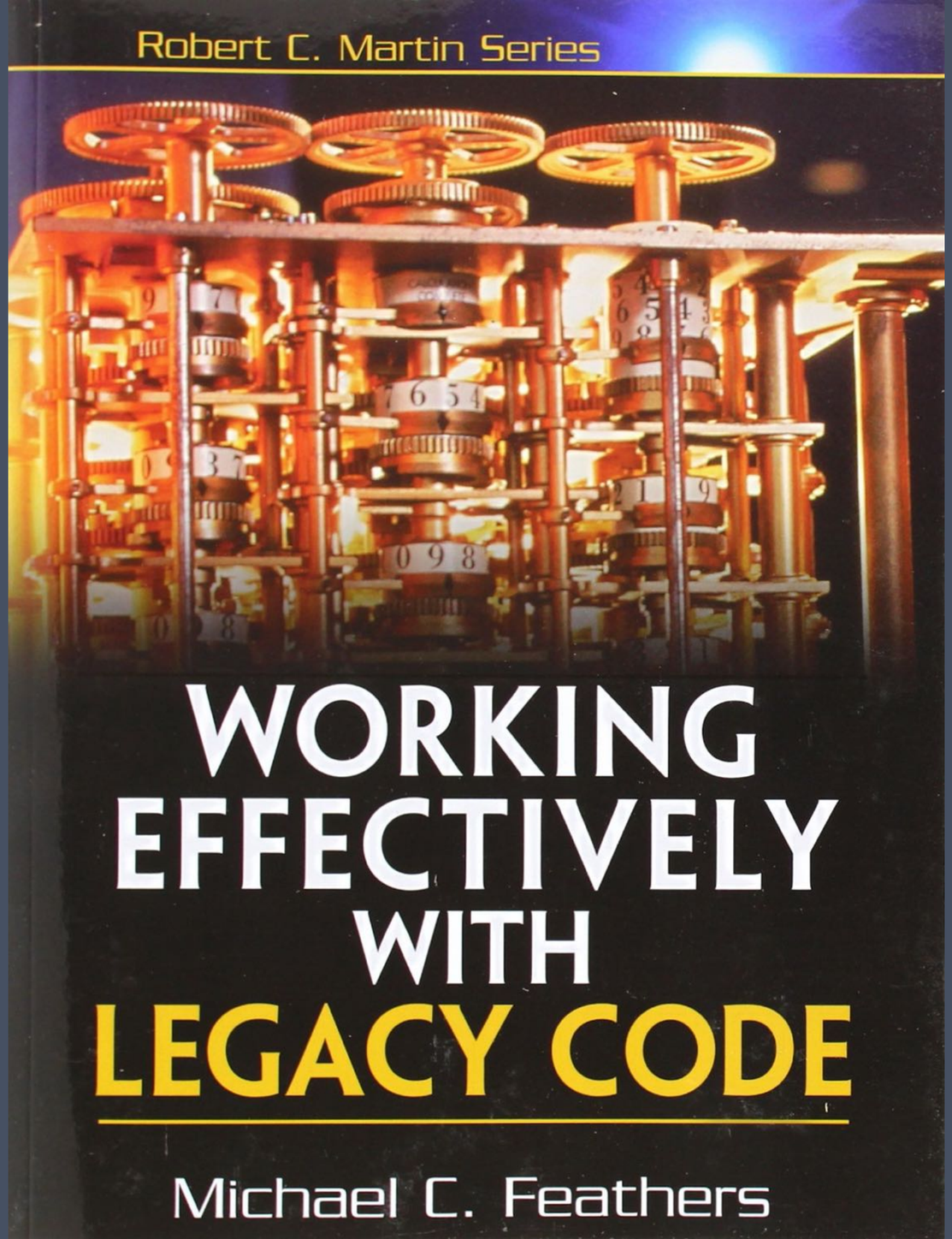


WORKING EFFECTIVELY WITH LEGACY CODE

RECOMMENDATION BY BEN

ACCURATELY IDENTIFYING WHERE CODE CHANGES NEED TO BE MADE COPING WITH LEGACY SYSTEMS THAT AREN T OBJECT-ORIENTED. HANDLING APPLICATIONS THAT DON T SEEM TO HAVE ANY STRUCTURE.

THIS BOOK ALSO INCLUDES A CATALOG OF TWENTY-FOUR DEPENDENCY-BREAKING TECHNIQUES THAT HELP YOU WORK WITH PROGRAM ELEMENTS IN ISOLATION AND MAKE SAFER CHANGES.



REFACTORING: IMPROVING THE DESIGN OF EXISTING CODE

RECOMMENDATION BY BEN

A COLLECTION OF TECHNIQUES TO IMPROVE THE STRUCTURAL INTEGRITY AND PERFORMANCE EXISTING SOFTWARE PROGRAMS