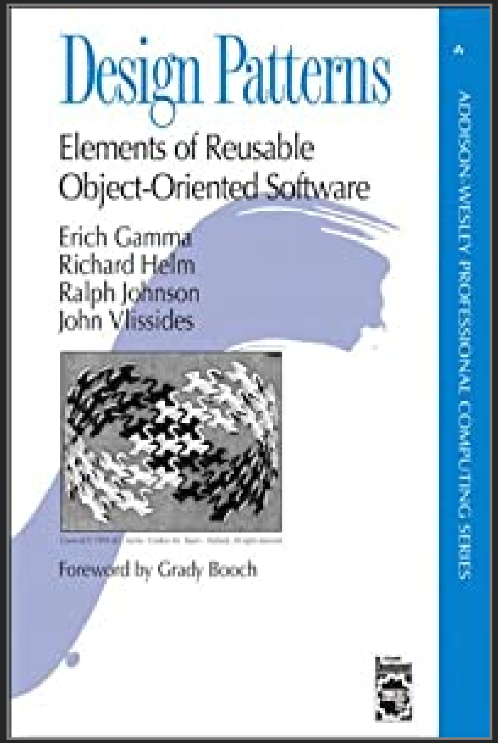
Patterns

Anti-Patterns and Refactoring

Yegor Bugayenko

Lecture #6 out of 16
90 minutes


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

"Experienced designers evidently know something inexperienced ones don't. What is it? One thing expert designers know not to do is solve every problem from first principles. Rather, they reuse solutions that have worked for them in the past. When they find a good solution, they use it again and again. Such experience is part of what makes them experts."

— *Design Patterns: Elements of Reusable Object-Oriented Software*, Erich Gamma et al.

"When I see patterns in my programs, I consider it a sign of trouble. The shape of a program should reflect only the problem it needs to solve. Any other regularity in the code is a sign, to me at least, that I'm using abstractions that aren't powerful enough—often that I'm generating by hand the expansions of some macro that I need to write."

— *Revenge of the Nerds*, Paul Graham

Chapter #1:

Some Patterns

Design Patterns and Anti-Patterns, Love and Hate (2016)

36 patterns (22 anti-patterns)

Adapter, Facade, Proxy, Decorator, Bridge

```
class Database {%
  String sql(String q);
}
void echo(Book b) {%
  print(b.title());
  print(b.author());
}
class BookInDatabase implements Book {%
  private Database d;
  private int id;
  String title() {%
    return d.sql("SELECT title FROM book WHERE id=%1", id);
  }
}
```

https://www.yegor256.com/2015/02/26/composable-decorators.html →

Resource Acquisition Is Initialization (RAII)

```
class File {%
  std::FILE* h;
public:
  File(const char* name) {%
    h = std::fopen(name, "w+")
  }
  ~File() {%
    std::fclose(h);
  }
}
void foo() {%
  f File("foo.txt");
  // write to f
}
```

https://www.yegor256.com/2017/08/08/raii-in-java.html →

Chapter #2:

Some Anti-Patterns

GOTO

```
void foo(int a) {%
  if (a % 2 == 0) {%
    printf("Even!");
    goto exit;
  }
  printf("Odd!");
  exit:
}
void foo(int a) {%
  if (a % 2 == 0) {%
    printf("Even!");
  } else {%
    printf("Odd!");
  }
}
```

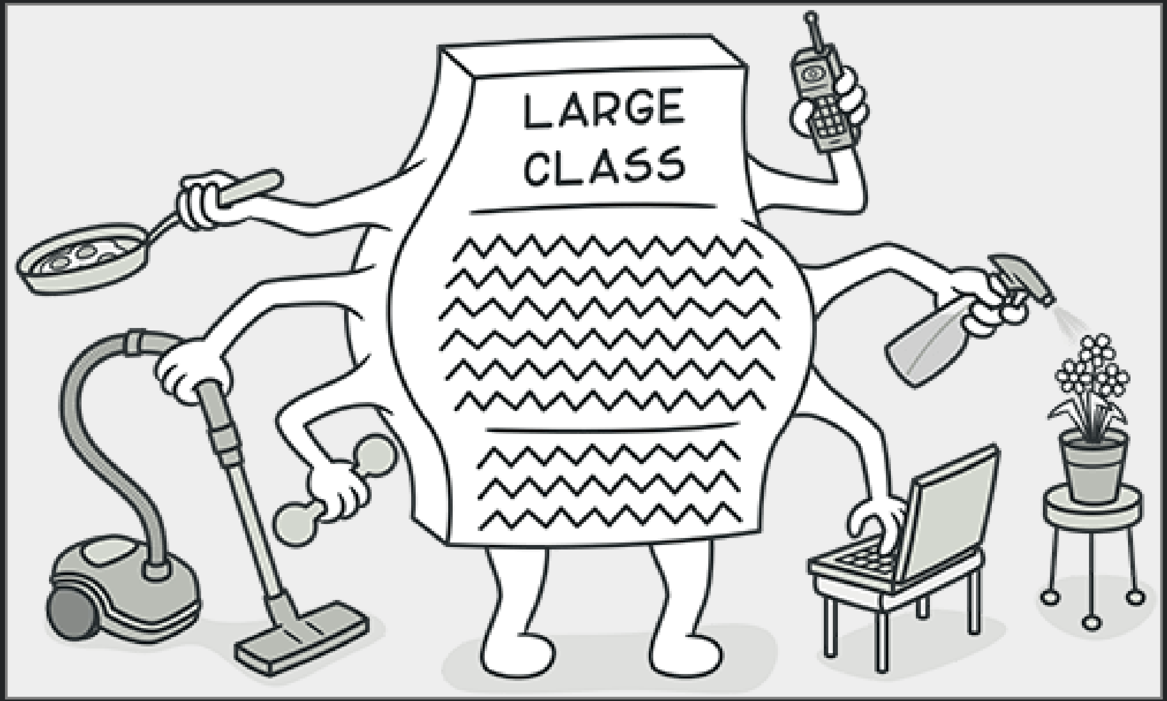
Magic Numbers

```
def points
  File.readlines("/data/users.csv") # why here?
    .map { |t| t.split(',', 11) } # what is 11?
    .map { a[7].to_i } # why 7?
    .inject(&:+)
end
```

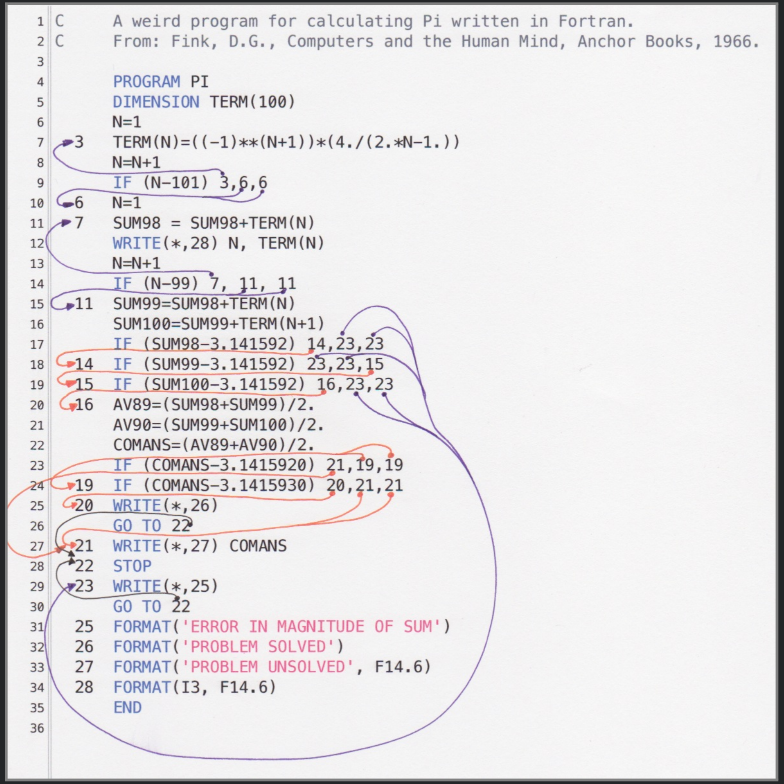
Magic Numbers ... Not!

```
def h2sec(h)
  return h * 60 * 60
end

def (h)
  seconds_in_minutes = 60
  minutes_in_hours = 60
  return h * seconds_in_minutes * minutes_in_hours
end
```

God Class

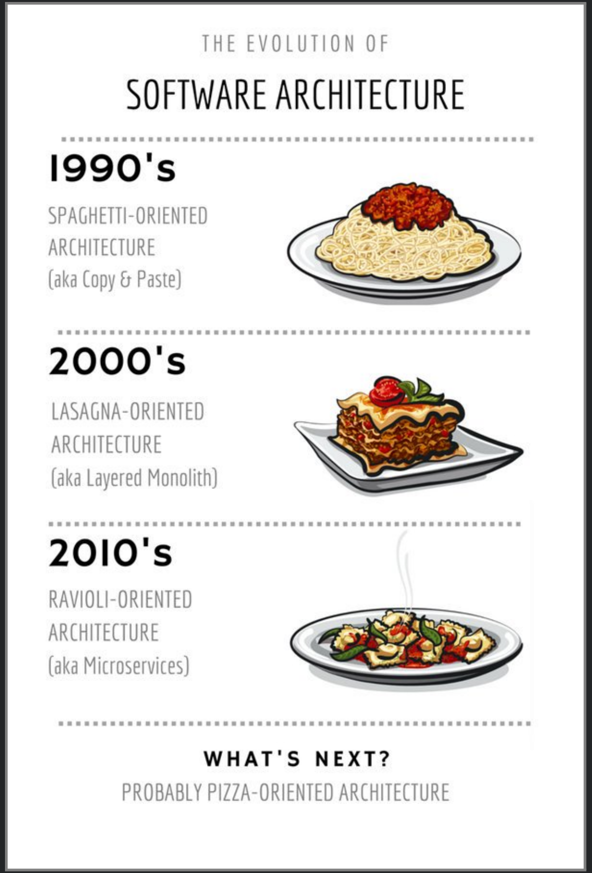


Spaghetti Code



```
C     A weird program for calculating Pi written in Fortran.
C     From: Fink, D.G., Computers and the Human Mind, Anchor Books, 1966.

      PROGRAM PI
      DIMENSION TERM(100)
      N=1
 3    TERM(N)=((-1)**(N+1))*(4./(2.*N-1.))
      N=N+1
      IF (N-101) 3,6,6
 6    N=1
 7    SUM98 = SUM98+TERM(N)
      WRITE(*,28) N, TERM(N)
      N=N+1
      IF (N-99) 7, 11, 11
 11   SUM99=SUM98+TERM(N)
      SUM100=SUM99+TERM(N+1)
      IF (SUM98-3.141592) 14,23,23
 14   IF (SUM99-3.141592) 23,23,15
 15   IF (SUM100-3.141592) 16,23,23
 16   AV89=(SUM98+SUM99)/2.
      AV90=(SUM99+SUM100)/2.
      COMANS=(AV89+AV90)/2.
      IF (COMANS-3.1415920) 21,19,19
 19   IF (COMANS-3.1415930) 20,21,21
 20   WRITE(*,26)
      GO TO 22
 21   WRITE(*,27) COMANS
 22   STOP
 23   WRITE(*,25)
      GO TO 22
 25   FORMAT('ERROR IN MAGNITUDE OF SUM')
 26   FORMAT('PROBLEM SOLVED')
 27   FORMAT('PROBLEM UNSOLVED', F14.6)
 28   FORMAT(I3, F14.6)
      END
```

Lasagna and Ravioli


THE EVOLUTION OF
SOFTWARE ARCHITECTURE

1990's
SPAGHETTI-ORIENTED ARCHITECTURE (aka Copy & Paste)

2000's
LASAGNA-ORIENTED ARCHITECTURE (aka Layered Monolith)

2010's
RAVIOLI-ORIENTED ARCHITECTURE (aka Microservices)

WHAT'S NEXT?
PROBABLY PIZZA-ORIENTED ARCHITECTURE

Chapter #3:

Anti-OOP Patterns

Anti-Patterns in OOP (2014)

Eleven: NULL, Utility Classes, Mutable Objects, Getters and Setters, Data Transfer Object (DTO), Object-Relational Mapping (ORM), Singletons, Controllers/Managers/Validators, Public Static Methods, Class Casting, Traits and Mixins.

Data Transfer Object (DTO)

Getters and Setters

```
// Getters and Setters: WRONG!
Dog dog = new Dog();
dog.setWeight("23kg");
w = dog.getWeight();

// Smart objects: RIGHT!
Dog dog = new Dog("23kg");
int w = dog.weight();
```

https://www.yegor256.com/2014/09/16/getters-and-setters-are-evil.html
→

Utility Class

```
public class NumberUtils {%
  public static int max(int a, int b) {%
    return a > b ? a : b;
  }
}
public class Max implements Number {%
  private final int a;
  private final int b;
  public Max(int x, int y) { this.a = x; this.b = y; }
  public int intValue() {%
    return this.a > this.b ? this.a : this.b;
  }
}
```

https://www.yegor256.com/2014/05/05/oop-alternative-to-utility-classes.html →

Singleton

```
class Database {%
  public static Database INSTANCE = new Database();
  private Database() {  /* start */ }
  public java.sql.Connection connect() { /* fetch */ }
}
c = Database.INSTANCE.connect();
class Foo {%
  private final Database d;
  void foo() {%
    this.d.connect();
  }
}
```

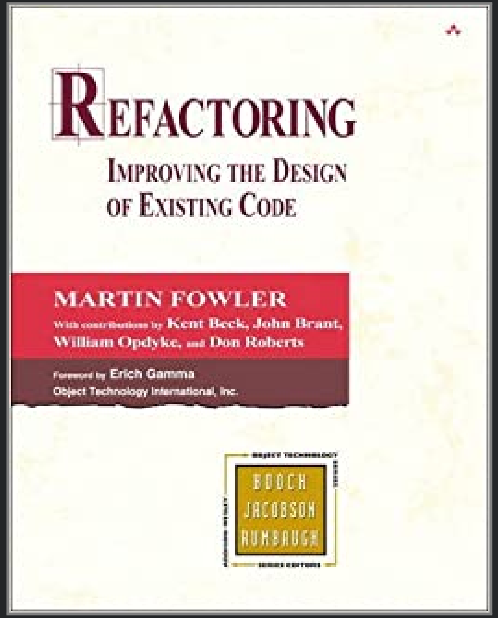
https://www.yegor256.com/2016/06/27/singletons-must-die.html →

Object-Relational Mapping (ORM)

```
// ORM: Wrong!
Post post = new Post();
post.setDate(new Date());
post.setTitle("How to cook an omelette");
session.save(post);

// Objects: RIGHT!
Post post = new Post();
post.setDate(new Date());
```

https://www.yegor256.com/2014/12/01/orm-offensive-anti-pattern.html →

Chapter #4: Some Refactorings


REFACTORING
IMPROVING THE DESIGN OF EXISTING CODE
MARTIN FOWLER
With contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts
Foreword by Erich Gamma
Object Technology International, Inc.
BOOCH JACOBSON RUMBAUGH

"Whenever I do refactoring, the first step is always the same. I need to build a solid set of tests for that section of code. The tests are essential because even though I follow refactorings structured to avoid most of the opportunities for introducing bugs, I'm still human and still make mistakes. Thus I need solid tests."

— *Refactoring: Improving the Design of Existing Code*, Martin Fowler

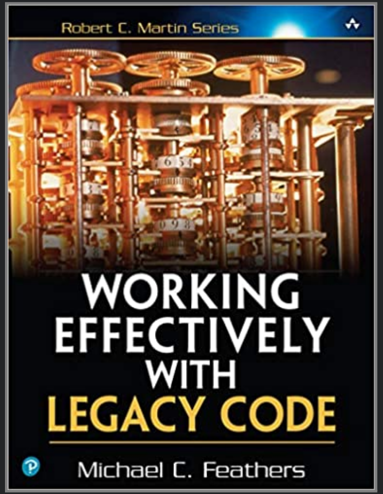
Extract Method

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

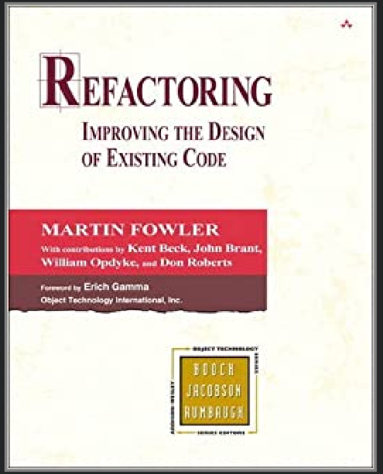
```
def root(a, b, c)
  d = Math.sqrt(b * b - 4 * a * c)
  r1 = (-b + d) / (2 * a)
  r2 = (-b - d) / (2 * a)
  [r1, r2]
end

def root(a, b, c)
  d = Math.sqrt(b * b - 4 * a * c)
  [r(a, b, d, 1), r(a, b, d, -1)]
end
def r(a, b, d, m)
  (-b + d * m) / (2 * a)
end
```

Chapter #5:

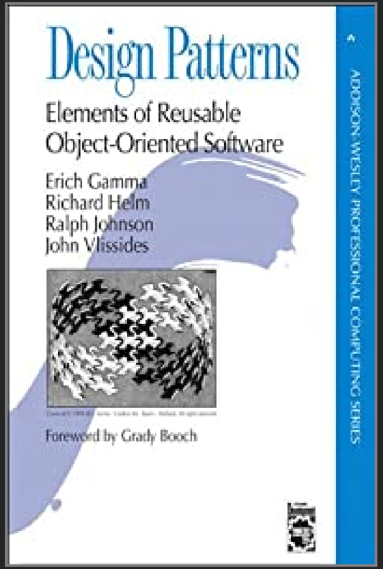
Books, Venues, Call-to-Action



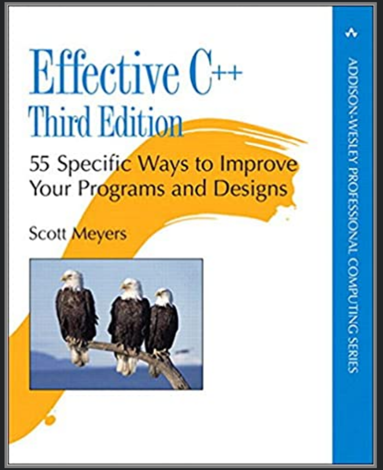
"Working Effectively with Legacy Code" by Michael Feathers


REFACTORING
IMPROVING THE DESIGN OF EXISTING CODE
MARTIN FOWLER
With contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts
Foreword by Erich Gamma
Object Technology International, Inc.
BOOCH JACOBSON RUMBAUGH

"Refactoring: Improving the Design of Existing Code" by Martin Fowler



"Design Patterns: Elements of Reusable Object-Oriented Software" by Erich Gamma et al.



"Effective C++: 55 Specific Ways to Improve Your Programs and Designs" by Scott Meyers

Where to publish:

SPLASH: ACM SIGPLAN conference on Systems, Programming, Languages, and Applications

International Conference on Code Quality (ICCQ), in cooperation with ACM SIGPLAN/SIGSOFT and IEEE

Call to Action:

In your application demonstrate the usage of 4+ design patterns. Also, perform 4+ refactorings, each one in its own pull request.

Still unresolved issues:

- How to prove certain patterns are anti-patterns?
- How to find methods for automated refactoring?
- How to guarantee validity during refactoring?
- How to mine patterns from code?