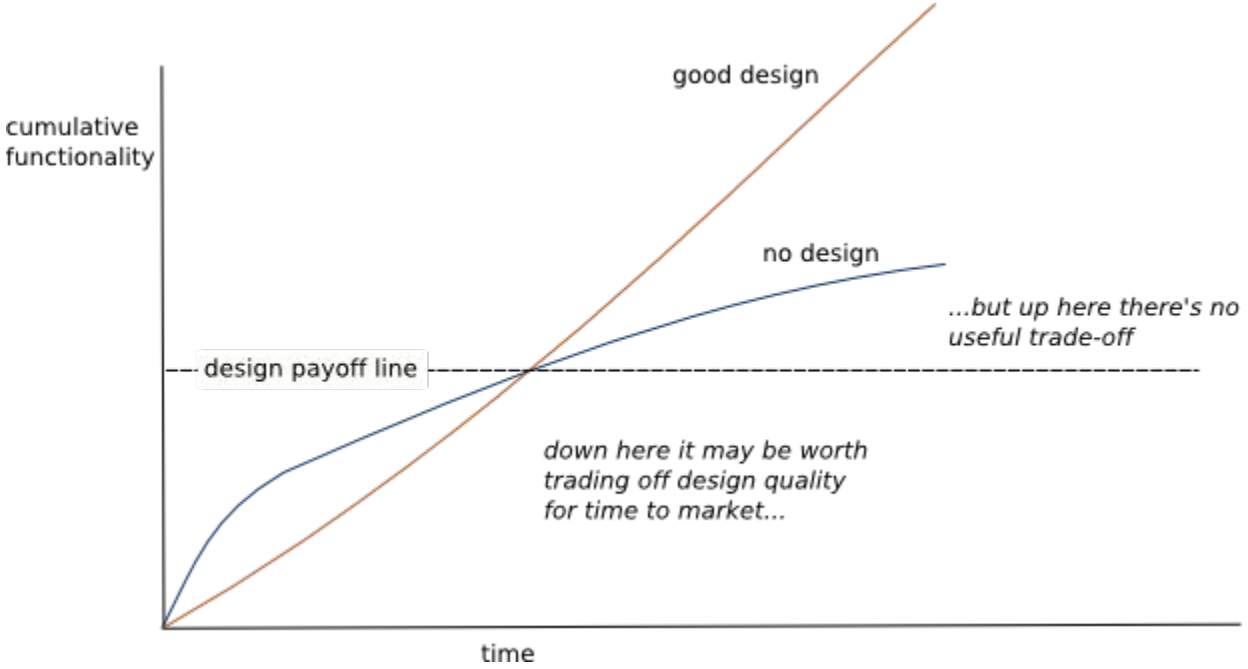
Design stamina hypothesis



Martin Fowler

Martin Fowler (born 1963) is a British software engineer, author and international public speaker on software development, specializing in object-oriented analysis and design, UML, patterns, and agile software development methodologies, including extreme programming.

ThoughtWorks®

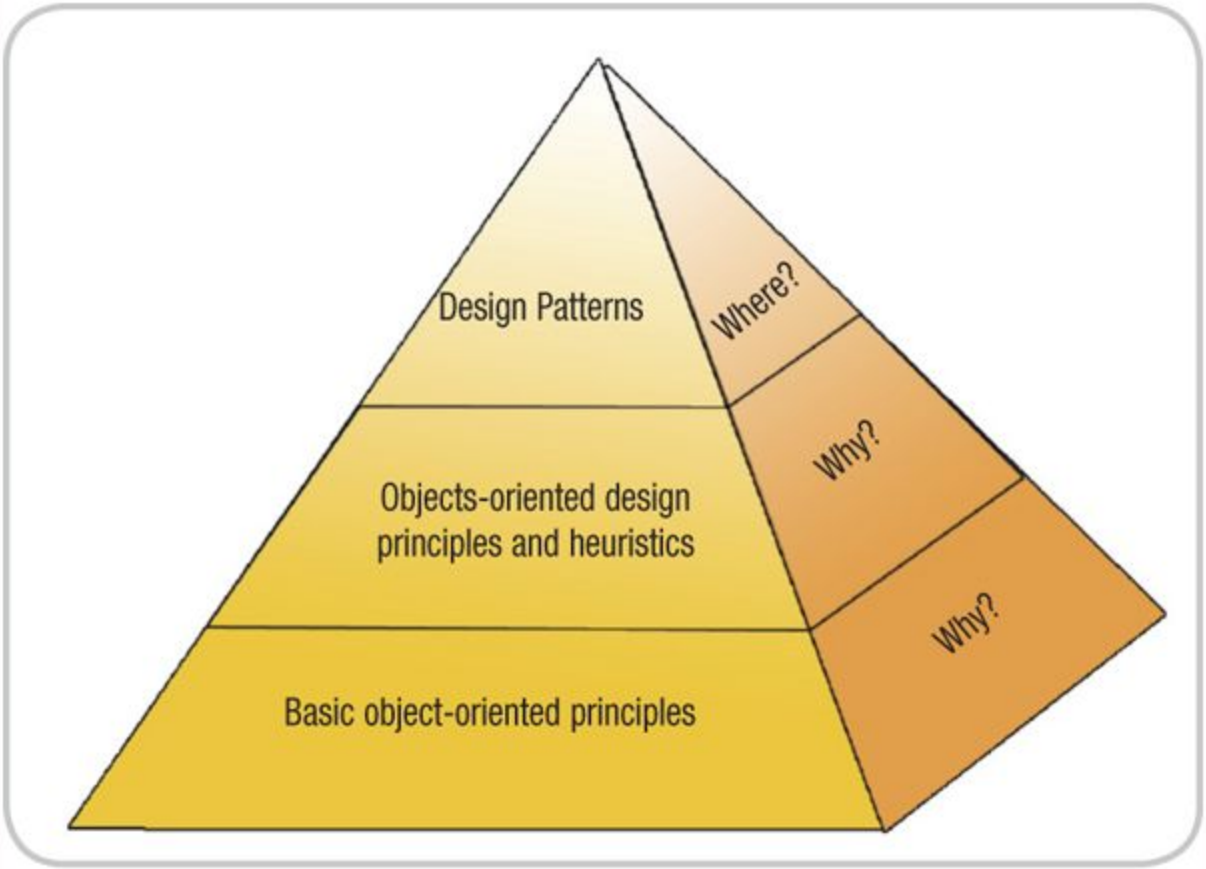


Figure 1 Design pyramid

Robert C. Martin

Robert C. Martin has been a software professional since 1970. In the last 35 years, he has worked in various capacities on literally hundreds of software projects. He has authored "landmark" books on Agile Programming, Extreme Programming, UML, Object-Oriented Programming, and C++ Programming. He has published dozens of articles in various trade journals. Today, He is one of the software industry's leading authorities on Agile software development and is a regular speaker at international conferences and trade shows. He is a former editor of the C++ Report and currently writes a monthly *Craftsman* column for *Software Development* magazine.

Mr. Martin is the founder, CEO, and president of Object Mentor Incorporated. Object Mentor is a sister company to Object Mentor International. Like OMI, Object Mentor is comprised of highly experienced software professionals who provide process improvement consulting, object-oriented software design consulting , training, and development services to major corporations around the world.

SOLID

Software Development is not a Jenga game

SOLID principles

Single responsibility principle
Open-closed principle
Liskov substitution principle
Interface segregation principle
Dependency inversion principle

Single responsibility principle

THERE SHOULD NEVER BE MORE THAN ONE REASON FOR A CLASS TO CHANGE.

Robert C. Martin “Uncle Bob”

Open/Closed principle

SOFTWARE ENTITIES (CLASSES, MODULES, FUNCTIONS, ETC.) SHOULD BE **OPEN** FOR EXTENSION, BUT **CLOSED** FOR MODIFICATION.

Robert C. Martin “Uncle Bob”

Liskov substitution principle

FUNCTIONS THAT USE POINTERS OR REFERENCES TO BASE CLASSES MUST BE ABLE TO USE OBJECTS OF DERIVED CLASSES WITHOUT KNOWING IT.

Robert C. Martin “Uncle Bob”

Interface segregation principle

CLIENTS SHOULD NOT BE FORCED TO DEPEND UPON INTERFACES THAT THEY DO NOT USE

Robert C. Martin “Uncle Bob”

Dependency inversion principle

A. HIGH LEVEL MODULES SHOULD NOT DEPEND UPON LOW LEVEL MODULES. BOTH SHOULD DEPEND UPON ABSTRACTIONS.

B. ABSTRACTIONS SHOULD NOT DEPEND UPON DETAILS. DETAILS SHOULD DEPEND UPON ABSTRACTIONS.

Robert C. Martin “Uncle Bob”

Cohesion and coupling

Cohesion: how strongly-related and focused are the various responsibilities of a module

Coupling: the degree to which each program module relies on on each one of the other modules

Design is all about dependencies

To avoid dependencies the code should be

loosely coupled
highly cohesive
easily composable
context independent

Responsibilities are axes of change

More responsibilities = more likelihood of change
Having multiple responsibilities within a class couples together these responsibilities

The more classes the change affects = the more likely the change will introduce errors

Single responsibility principle

THERE SHOULD NEVER BE MORE THAN ONE REASON FOR A CLASS TO CHANGE.

Robert C. Martin “Uncle Bob”

SINGLE RESPONSIBILITY PRINCIPLE

Just Because You Can, Doesn't Mean You Should

Sample application

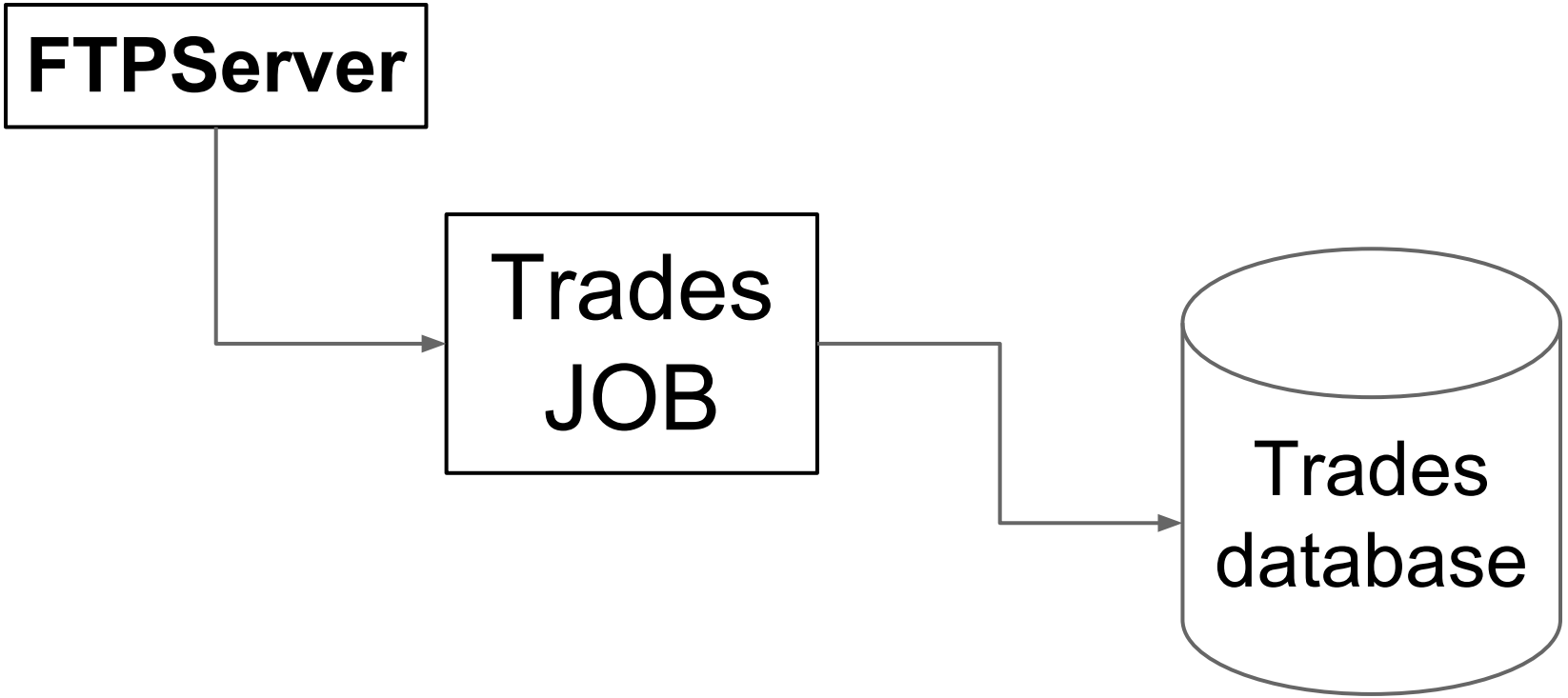
trades.csv

```
trade_id, instrument, price, quantity
234EQTYR23, ‘US5949181045’, 120, ‘USD’, 1000
23TTQWUY3, ‘NL0000729408’, 99, ‘CAD’, 100
54GHJGTU76, ‘RU1238763866’, 3000, ‘RUB’, 1000
```

FTP Server

host: localhost
login: foo
password: passw
path: public/prod
filename: trades.csv

Sample application



Concerns

What if ftp host/login/password changes?
What if I need another job like this?

What if I don’t want to ftp a file in every test?

Open/Closed principle

“Open For Extension”

This means that the behavior of the module can be extended. That we can make the module behave in new and different ways as the requirements of the application change, or to meet the needs of new applications.

“Closed for Modification”

The source code of such a module is inviolate. No one is allowed to make source code changes to it.

OPEN CLOSED PRINCIPLE

Open Chest Surgery Is Not Needed When Putting On A Coat

Open/Closed principle

Problems with sample application

Adding new rules requires changes to the calculator
Each change can introduce bug and requires re-testing

Open/Closed principle

Solution

Writing new classes is less likely to introduce problems

- Nothing depends on new classes
- New classes are easy to design and test

Open/Closed principle

Parameters

Inheritance / Template Method Pattern

child types override behavior of a base class

Composition / Strategy Pattern

client code depends on abstraction
provides a “plug in” model
implementations utilize inheritance

When do we apply OCP

Experience tells you

- you know in advance that the change is likely to happen

Otherwise

- don’t apply OCP at first
- if the module changes once - accept it
- second time - refactor to achieve OCP

There is no free lunch

OCP adds complexity

No design can be closed against all types of changes