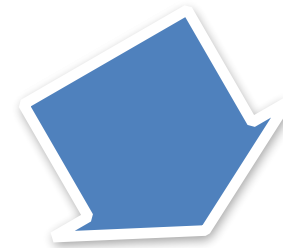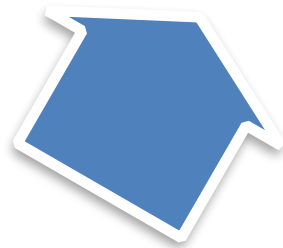# Refactoring

Graeme Ludwig

The process of changing a software system in such a way that it does not alter the **external behaviour** of the code, yet improves its **internal structure**.

Fowler, et al., Refactoring, 1999.

Test Driven Development

GREEN
Make the test work quickly.
Commit whatever sins are necessary.

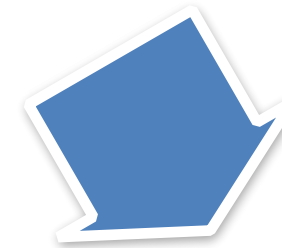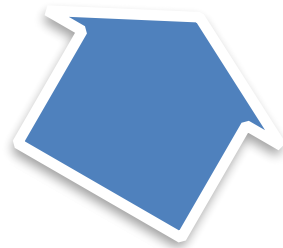**RED**
Write a failing test.

REFACTOR
Improve the design.
Fix some/all of the sins.

Tests & Refactoring

Tests:

- Reduce fear; give us the courage to refactor.

- Prove everything still works after refactoring.

- Help us write code that is better designed.

- Help us iterate the design to keep it clean.

- Help to document features IF they are well named and well written.

Refactoring from "A" to "B"
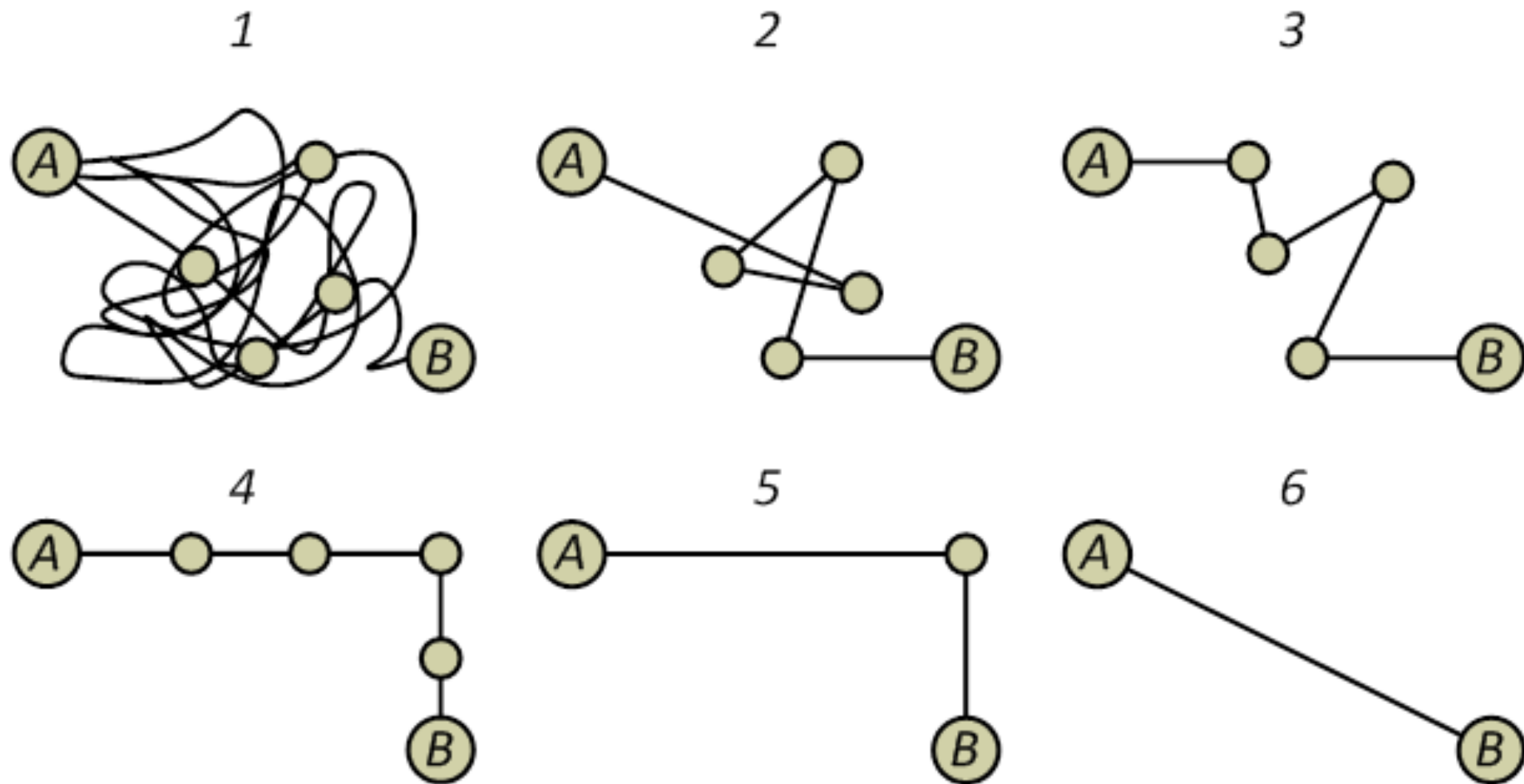
Test Driven Development

GREEN
Make the test work quickly.
Commit whatever sins are necessary.

Good tests ✓

Automated tests ✓

REFACTOR
Improve the design.
Fix some/all of the aforementioned sins.

RED
Write a failing test.

Why Refactor?

Because it's hard to be "Right First Time"

- Understanding of domain improves

- User requirements change

- Original design inadequate

- System becomes brittle over time

- Saves £££'s

Who is your customer?

Refactoring: Before & After

Before refactoring...

Unreadable code ✖
Duplicated code ✖
Complex code ✖
Hard to change ✖

After refactoring...

Clean code ✔
Better code ✔
Easier to understand ✔
Cheaper to modify ✔

What does good code look like?

Any fool can write code that a computer can understand.

Good programmers write code that humans can understand.

Refactoring Movies
(A Worked Example)

https://github.com/ludwiggj/refactoringMovies
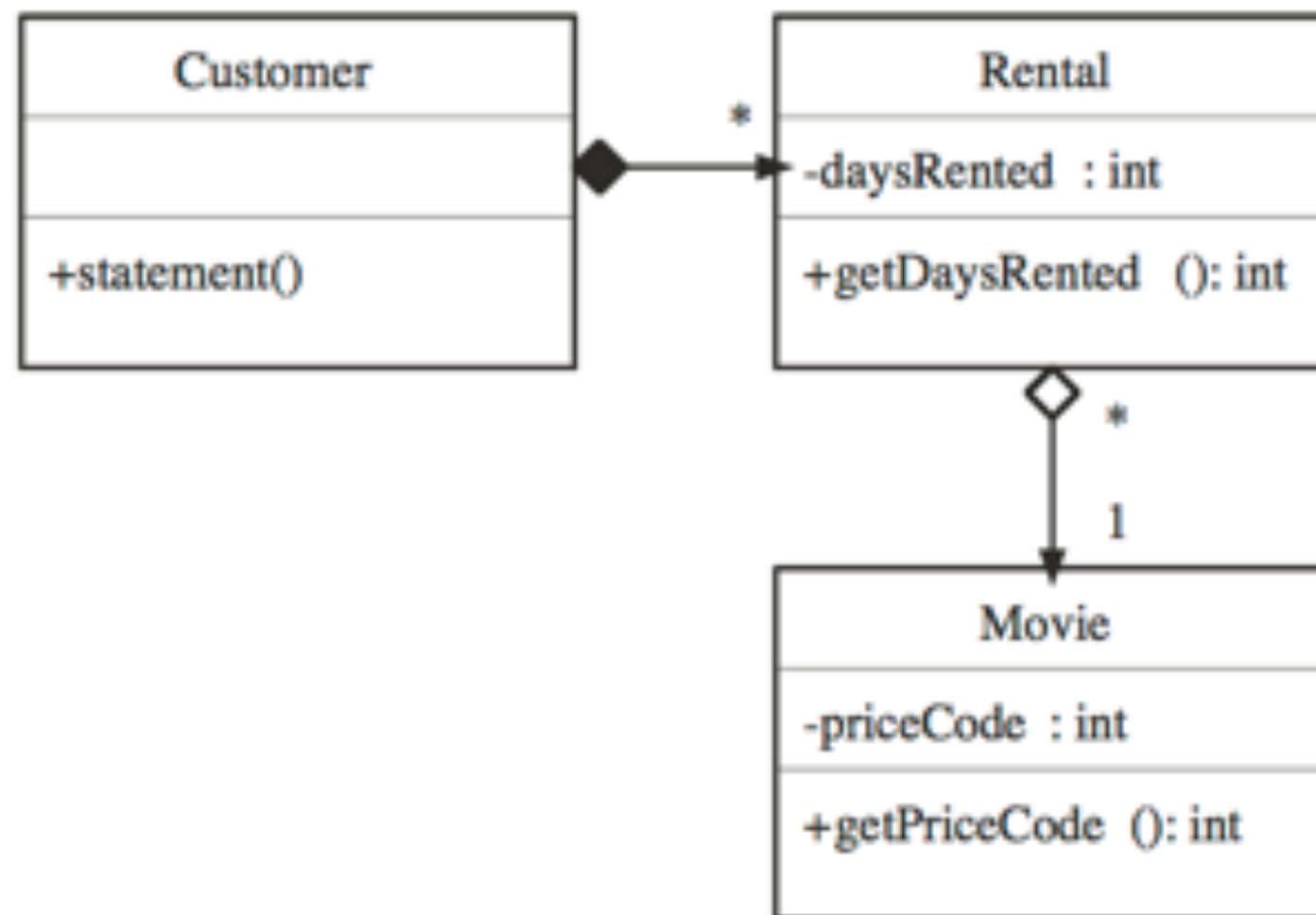
Comments on the Starting Program

- When you find you have to add a feature to a program, and the program's code is not structured in a convenient way to add the feature, first refactor the program to make it easy to add the feature, then add the feature.