Refactor Like A Pro

Chris Freeman
@cwfreeman
chris.freeman@outpace.com
“A most important, but also most elusive, aspect of any tool is its influence on the habits of those who train themselves in its use.”

Edsger Dijkstra
public String statement() {
    double total_amount = 0;
    int frequent_renter_points = 0;
    String result = "Rental Record for " + get_name() + "\n";
    for (Rental element : rentals) {
        double this_amount = 0;
        // determines amount for each line
        switch(element.get_movie().get_price_code()) {
        case Movie.REGULAR:
            this_amount += 2;
            if(element.get_days_rented() > 2)
                this_amount += (element.get_days_rented() - 2) * 1.5;
            break;
        case Movie.NEW_RELEASE:
            this_amount += element.get_days_rented() * 3;
            break;
        case Movie.CHILDRENS:
            this_amount += 1.5;
            if(element.get_days_rented() > 3)
                this_amount += (element.get_days_rented() - 3) * 1.5;
            break;
        }
        // add frequent renter points
        frequent_renter_points += 1;
        if(element.get_movie().get_price_code() == Movie.NEW_RELEASE & element.get_days_rented() > 1)
            frequent_renter_points += 1;
        // show figures for this rental
        result += "\n" + element.get_movie().get_title() + "\t" + this_amount + "\n";
        total_amount += this_amount;
    }
    // add footer lines
    result += "\nAmount owed is " + total_amount + "\n";
    result += "You earned " + frequent_renter_points + " frequent renter points."
    return result;
}
Duplicated Code
Multi-hundred-line functions
Global variable abuse
Useless/misleading comments
Repurposed primitive variables
Duplicated Code
Nested (nested (nested loops) )
Inconsistent_sTyle
Poorly named stuff

This is what the worst code that I’ve worked on looked like.
This is why bad code is horrible.
This is a hard situation to make money in.
Commit to refactoring over rewriting

Every organization which moves from bad code to good code has made these realizations and commitments.
Commit to refactoring over rewriting
It takes time

Geo took several years.
Commit to refactoring over rewriting
It takes time
No code is sacred

Everyone owns the code. We’re all experts.
Commit to refactoring over rewriting

It takes time

No code is sacred

Refactoring is not a special activity

Do not ask for time aside to refactor your work. There are some exceptions, but in general, you want to make improvements every time you look at a file. Don’t expect a perfect file immediately; it will get there only if you work at it.
Commit to refactoring over rewriting
It takes time
No code is sacred
Refactoring is not a special activity
Portfolio really big efforts

Sometimes, it is necessary to put aside some effort. This is a business decision. Help the business make it.

You'll need to show the business success irregular refactoring efforts first.
Commit to refactoring over rewriting
It takes time
No code is sacred
Refactoring is not a special activity
Portfolio really big efforts
Green = Deploy!
Most development occurs like this.
Every other human activity is iterative.

Not Mozart.
If someone can refactor this building, you can refactor your code.
Refactoring Step #1:
Run The Tests
This has implications for your tests – short/fast/good/…

Like a nervous habit

If you’ve heard of the TDD cycle, you’re familiar with Red-Green-Refactor. Well, the refactoring cycle is Green-Change-Green.
An Atomic Refactoring: Rename
What this means in practice is that a refactoring should only take a few seconds to a minute to make. It means that you should be able to EASILY revert it. And it means that your tests pass before and after, and that you can trust them to fail when you break things.

Confidence leads to larger steps. Be careful!
A Refactoring Pro
Runs Tests
Constantly
oops, I broke it!
oops, I broke it!
oops, I broke it!

Revert

No, really. Revert
oops, I broke it!
oops, I broke it!

Think about what you did.

Revert

No, really. Revert

Try it differently.
Atomic Refactoring:
Extract Method
Method call overhead is nearly never a problem anymore.

If you think it is, you should take measurements. One or two methods could be inlined to get the performance you need.

By giving code blocks names, we make it easier to manipulate them in our head.
Atomic Refactoring:
Move Method
We’re improving design by ensuring SOLID (for example) principles. If this were functional programming, we’d be ensuring compositibility, etc.
It’s going to be much easier to test Rental now.
Story of reporting system.

This is central to the idea of refactoring. Most systems reach a point of infeasibility due to a high cost of change. We want to push that off as much as possible.
Editing is not writing!
One of the big reasons we want to separate these activities is that we want to focus on one at a time.
Atomic Refactoring:
Extract Variable
Continue up to the point we extract getRegularMoviePrice().
Tools
Enable Refactoring

Decision Fatigue
Tools reduce cognitive load!

Story of learning renaming in editor instead of Vim
Inlining multi-return fns is usually impossible. Inlining functions with side-effects can get ugly.

I like refactoring in part because it encourages me to use relatively pure code.
How do we know which actions to take? In large part, SMELLS
Smells are an INDICATION of design problems. They do not obligate you to take action, they merely suggest it.

If you need help figuring out what to do in the face of a smell, there are lists of smell→refactorings.
smells

duplicate code

extract method
Feature envy
smells
long parameter list
inappropriate intimacy
contrived complexity
large class
duplicate code
lazy class
conditional complexity
long method
refused bequest
dependent bequest
combinatorial explosion
divergent change
comment deodorant
duplicate text
lazy class
refused bequest
conditional complexity
long method
dependent bequest
combinatorial explosion
divergent change
comment deodorant
duplicate text
smells

- long method
- extract method
smells

- conditional complexity
- introduce null object
- move embellishment to decorator
- replace with strategy
feature envy
duplicate code
lazy class
long method
long parameter list
inappropriate intimacy
contrived complexity
large class
conditional complexity
combinatorial explosion
duplicate code
comment deodorant
refused bequest
divergent change
smells

feature envy
extract method
extract field
move method
feature envy
long parameter list
inappropriate intimacy
corrupted complexity
long method
duplicate code
large class
lazy class
feature envy
inappropriate intimacy
corrupted complexity
long parameter list
conditional complexity
combinatorial explosion
comment deodorant
duplicate code
refused bequest
You’ll want to become proficient at these building blocks.

By the way, reordering things is often necessary, but fraught with danger. Always, always, always run your tests after each reordering!
Feature Envy: Price & FrequentRenterPoints
Refactorings with Test Runs Throughout
smells

primitive obsession

replace type code with polymorphism
primitive obsession

replace type code with polymorphism

extract method  extract subclass  move method

Complete subclassing of Movie
Refactoring helps us create design and feature options.
Visitor pattern is not close to the state pattern. State is close to strategy. Designs exist in a landscape which you can move through.

Toward and Away create options and make them obvious.
This is a bit of a straw-man of big-up-front-design, but it is descriptive of many people's experience.

As a developer, it's your job to learn and exercise design.
Explore the for-loop; split it into three. Move those three onto the collection?
Learn the code
Clear the code of cruft
Clarify design ambiguity
Try out new designs cheaply
Create options

Talk about inside-out Clojure program?
Allows us to extract functions, and move fns to Rentals class.
When It’s Complicated: 
The Mikado Method
Try a refactoring naively
Write down blockers
Revert
Fix blockers (Recurse here)
Try again

Useful for large scale refactorings, especially.
<table>
<thead>
<tr>
<th>Defects</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defects are often made possible (or likely) due to design problems. Fix the design problem, and the defect becomes easy to fix.</td>
<td>Features can be difficult to add because the design does not account for them. Fix the design, and the feature is easy.</td>
</tr>
</tbody>
</table>

... Speculative

Refactor first is often speculative in nature. You should keep that in mind when you consider refactoring first.
Wrap price, which is a double, in a class called 'Money'.
Variables
‘Flow’ from Initialization to Consumption.
Scaling It
1. Code
2. Design
3. Architecture
4. Infrastructure
Extract interfaces
Take reversible baby steps
Verify Behavioral Equivalence

Interface, boundaries, seams, ...
monolithic -> services

- extract interface
- move implementation to library
- stand up service
- create endpoint for implementation
- change library call to service call
- hide service call behind interface

Example: static pages in GH, Moving away from Twin Towers, moving to DI,
refactor db

zero-downtime db migrations
refactoring is:

...taking small steps
...keeping the system working
...safe
...transformation (of code)
...transformative (for you)
VS

Atomic  Composite
Tactical  Strategic
Tiny     Bigger
Practicing
**Pair**
Pair with people who refactor constantly. Ask them to help you hold yourself accountable for practicing.

**Katas**
Do the Video Store several times. Strive to type as little as possible.

**Golf**
Play Refactoring Golf. See if you can achieve the same results with fewer, or better, refactorings.
<table>
<thead>
<tr>
<th>Books</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refactoring</td>
</tr>
<tr>
<td>Refactoring To Patterns</td>
</tr>
<tr>
<td>Working Effectively With Legacy Code</td>
</tr>
<tr>
<td>The Mikado Method</td>
</tr>
<tr>
<td>Fowler</td>
</tr>
<tr>
<td>Kerievsky</td>
</tr>
<tr>
<td>Feathers</td>
</tr>
<tr>
<td>Ellnestam &amp; Brolund</td>
</tr>
</tbody>
</table>
Design and code are malleable. Refactoring helps you manipulate them easily.
Thank you