WRITE A FAILING TEST

MAKE IT GREEN

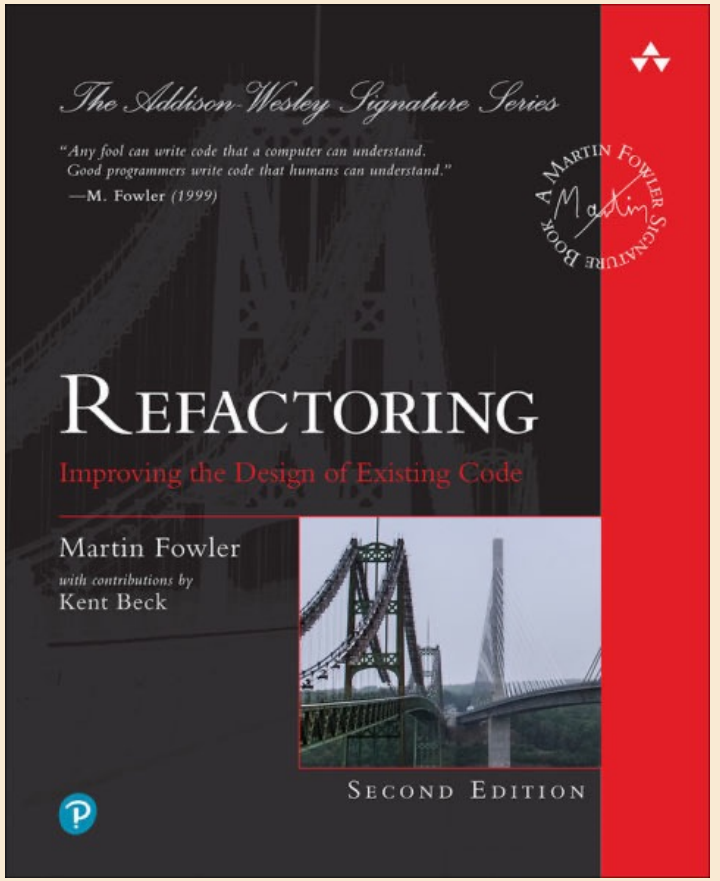
REFACTOR

Refactoring:

The what, why, when and how

Haakon Hafsahl Svane

The bible


The Addison-Wesley Signature Series
"Any fool can write code that a computer can understand.
Good programmers write code that humans can understand."
—M. Fowler (1999)
A Martin Fowler Signature Book
Martin
REFACTORING
Improving the Design of Existing Code
Martin Fowler
with contributions by
Kent Beck
SECOND EDITION

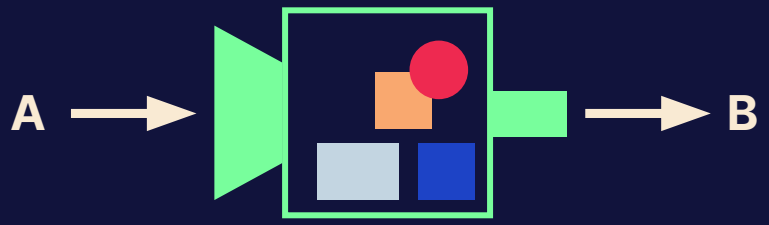
```
Surface* Spritesheet::create_strip(const Vector2D<int> start_stop, Uint8 dim) const
{
    const int start_index = start_stop.x-1;
    const int stop_index = start_stop.y-1;
    const int l = stop_index - start_index + 1;
    const int cd = CONSTANTS::CELL_DIM;
    const int c = width_ / cd;
    const float fac = static_cast<float>(cd)/dim;
    const int c_fac = static_cast<int>(c*fac);

    const int n = (static_cast<int>((l/c)*fac)*(width_/dim)) + (l%c)*fac;

    if (stop_index < start_index) throw std::invalid_argument("Indexes for ()-operator are wrong!");
    if (surf_ == nullptr) throw std::invalid_argument("Surface has not been set for the sheet!");

    SDL_Surface* new_surf = SDL_CreateRGBSurface(0, dim*n, dim, CONSTANTS::BIT_DEPTH, 0xff000000, 0x00ff0000, 0x0000ff00, 0x000000ff);
    if (new_surf == nullptr) std::fprintf(stderr, "Could not create surface from spritesheet. Error: %s", SDL_GetError());
    else {
        const int start_x = static_cast<int>(((start_index)%c)*fac);
        const int dy = (start_x + n - 1) / c_fac + 1;

        const int f_x_s = start_x*dim;
        const int f_y_s = static_cast<int>((1.f*start_index)/c)*cd;

        const int l_x = (c_fac-start_index%c_fac+(dy-2)*c)*dim;
        const int f_w = (dy > 1)? ((c_fac - start_x)*dim) : (n*dim);
        const int l_w = (n-((dy-1)*c-start_index))*dim;

        for (int i = 0; i < dy; ++i) {
            SDL_Rect source;
            SDL_Rect dest;

            if (i == dy-1 && i != 0) {
                source.x = 0; source.y = f_y_s + i*dim; source.w = l_w; source.h = dim;
                dest.x = l_x; dest.y = 0; dest.w = l_w; dest.h = dim;
            }
            else if (i == 0) {
                source.x = f_x_s; source.y = f_y_s+i*dim; source.w = f_w; source.h = dim;
                dest.x = 0; dest.y = 0; dest.w = f_w; dest.h = dim;
            }
            else {
                source.x = 0; source.y = f_y_s +i*dim; source.w = width_; source.h = dim;
                dest.x = f_w+(i-1)*c*dim; dest.y = 0; dest.w = width_; dest.h = dim;
            }
```

What is refactoring?

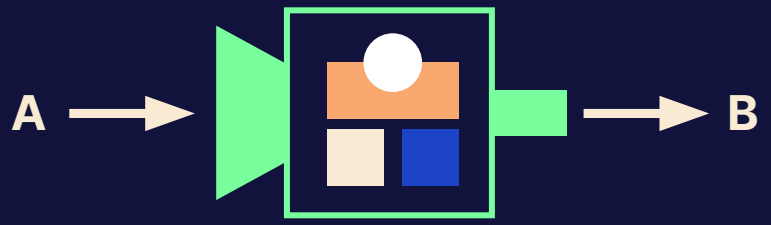
(noun):

«a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior»

- Martin Fowler



Before refactoring



After refactoring

Why should I refactor?

- Improving design
- Improving readability
- Finding bugs
- Faster development

When do I refactor?

TLDR: always

- Before you write any code
- `while` you are writing code (remember *the rule of three)*
- After writing code

bouvet

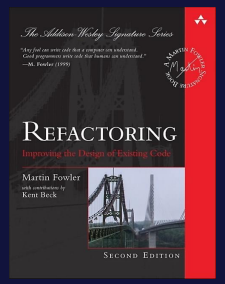
How do I refactor?

- Code smells
- Atomic refactorings -> Test -> Commit
- Check out Fowler’s refactoring catalog!

Thanks

The bible:

Refactoring – Martin Fowler



Contact info:

Haakon Hafsahl Svane

haakon.svane@bouvet.no