The Four Horsemen of the ~~Apocalypse~~ Code

State

State

```
class Stable

  attr_reader :horses

  def initialize
    @horses = Horse.all
  end

  def feed_horses
    kittens = Kitten.all.to_a
    per_horse = kittens.length / @horses.length
    @horses.each do |horse|
      if horse.hungry?
        num_eaten = horse.eat(kittens.pop(per_horse))
        KittenOrder.new(quantity: num_eaten).save
      end
    end
  end

end
```

~~State~~

```ruby
class StableStateless

  attr_reader :horses

  def initialize(horses, kitten_source, order_sink)
    @horses = horses
    @kitten_source = kitten_source
    @order_sink = order_sink
  end

  def feed_horses()
    @horses.each do |horse|
      if horse.hungry?
        kittens = @kitten_source.(horse)
        horse.eat(kittens)
        @order_sink.(kittens.count)
      end
    end
  end

end
```

~~State~~

```
describe 'Stable' do

  it 'feeds horses' do
    orders = 0
    source = lambda { |_| [Kitten.new] * 2 }
    sink = lambda { |num_eaten| orders += num_eaten }
    horses = [
      Horse.new(hungry: true),
      Horse.new(hungry: true),
      Horse.new(hungry: false)
    ]

    stable = Stable.new(horses, source, sink)
    stable.feed_horses()
    expect(orders).to eq(4)
  end

end
```

~~State~~

- Sources and sinks
- Ports and adapters
- Funtional core, imperative shell

Coupling

Coupling

```
class Stable
  ...

  def scrub_horses
    @horses.each do |horse|
      @stable_boy.scrub(horse)
    end
  end
```

```
class StableBoy

  def scrub(horse)
    brush = Brush.new
    brush.rinse()
    brush.apply(horse)
  end

end
```

~~Coupling~~

```ruby
class Stable
  ...

  def scrub_horses
    @horses.each do |horse|
      brush = Brush.for(horse)
      @stable_boy.scrub(brush, horse)
    end
  end
```

```ruby
class StableBoy

  def scrub(brush, horse)
    brush.prepare()
    brush.apply(horse)
  end

end
```

~~Coupling~~

```
class Stable
  ...

  def scrub_horses
    @horses.each do |horse|
      brush = Brush.for(horse)
      @stable_boy.scrub(brush, horse)
    end
  end
```

```
class StableBoy

  def scrub(brush, horse)
    brush.prepare()
    brush.apply(horse)
  end

end
```

~~Coupling~~

```
class Stable
  ...

  def scrub_horses
    @horses.each do |horse|
      brush = Brush.for(horse)
      @stable_boy.scrub(brush, horse)
    end
  end
```

```
class StableBoy

  def scrub(brush, horse)
    brush.prepare()
    brush.apply(horse)
  end

end
```

~~Coupling~~

- One place, one change
- Tell, don't ask
- Law of Demeter ("no dot-chaining")

```
def scrub_horses
  @horses.each do |horse|
    brush = Brush.for(horse)
    @stable_boy.right_hand.grab(brush)
    @stable_boy.scrub(brush, horse)
  end
end
```

Complexity

Complexity

I conclude that there are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies and the other way is to make it so complicated that there are no obvious deficiencies. — Tony Hoare

Complexity

Sandi Metz: All the Little Things

Complexity

```
if (tempToken != null) {
    loginToken = parseToken(tempToken);
    if (loginToken != null) {
        // This creates our login information
        List<NameValuePair> parameters = createParams();
        // Here we send the information to the server and login
        String postResults = conn.postData("https://mytlc.bestbuy.com/etm/login.jsp", parameters);
        // If we logged in properly, then we download the schedule
        if (postResults.contains("etmMenu.jsp")) {
            // Here is the actual call for the schedule
            updateStatus("Retrieving schedule...");
            String schedule = conn.getData("https://mytlc.bestbuy.com/etm/time/timesheet/etmTnsMonth.jsp");
            // If we successfully got the information, then parse out the schedule to read it properly
            if (schedule != null) {
                updateStatus("Parsing schedule...");
                workDays = parseSchedule(schedule);
                String secToken = parseSecureToken(schedule);
                if (secToken != null) {
                    parameters = createSecondParams(secToken);
                    schedule = conn.postData("https://mytlc.bestbuy.com/etm/time/timesheet/etmTnsMonth.jsp", parameters);
                    if (schedule != null) {
                        List<String[]> secondMonth = parseSchedule(schedule);
                        if (secondMonth != null) {
                            workDays.addAll(secondMonth);
                            finalDays = workDays;
                            updateStatus("Adding shifts to calendar...");
                            // Add our shifts to the calendar
                            if (addDays()) {
                                // Report back that we're successful!
                                Message msg = Message.obtain();
                                Bundle b = new Bundle();
                                b.putString("status", "DONE");
                                b.putInt("count", workDays.size());
                                msg.setData(b);
                                try {
                                    messenger.send(msg);
                                } catch (Exception e) {
                                    // Nothing
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

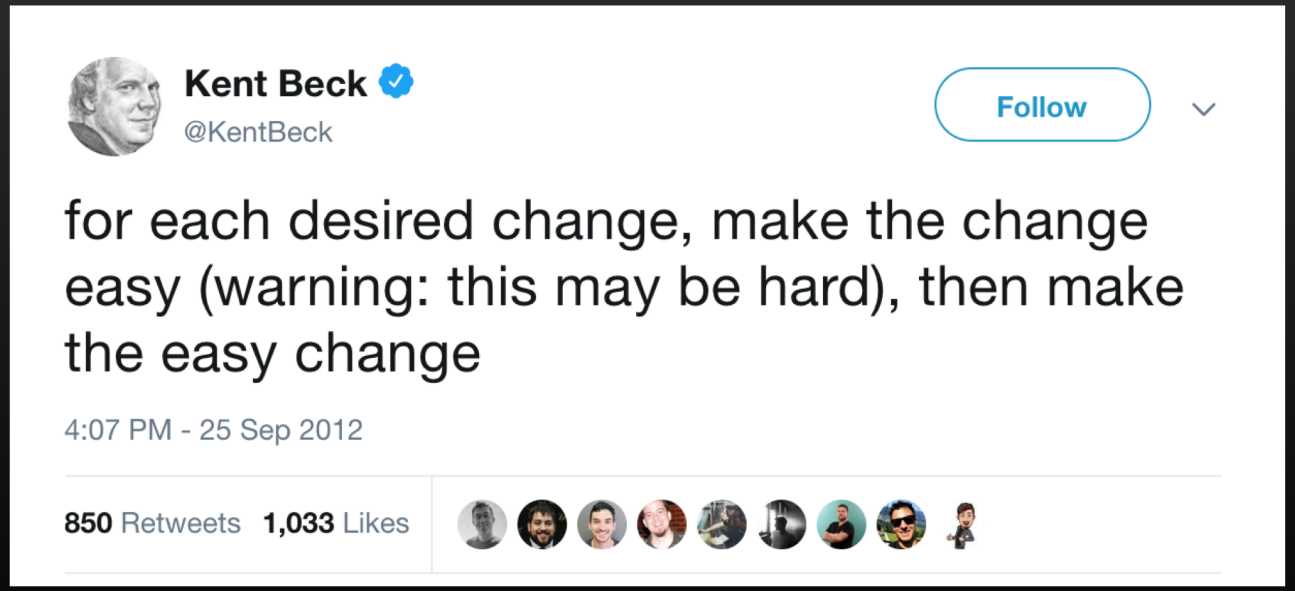
Complexity

Squint test

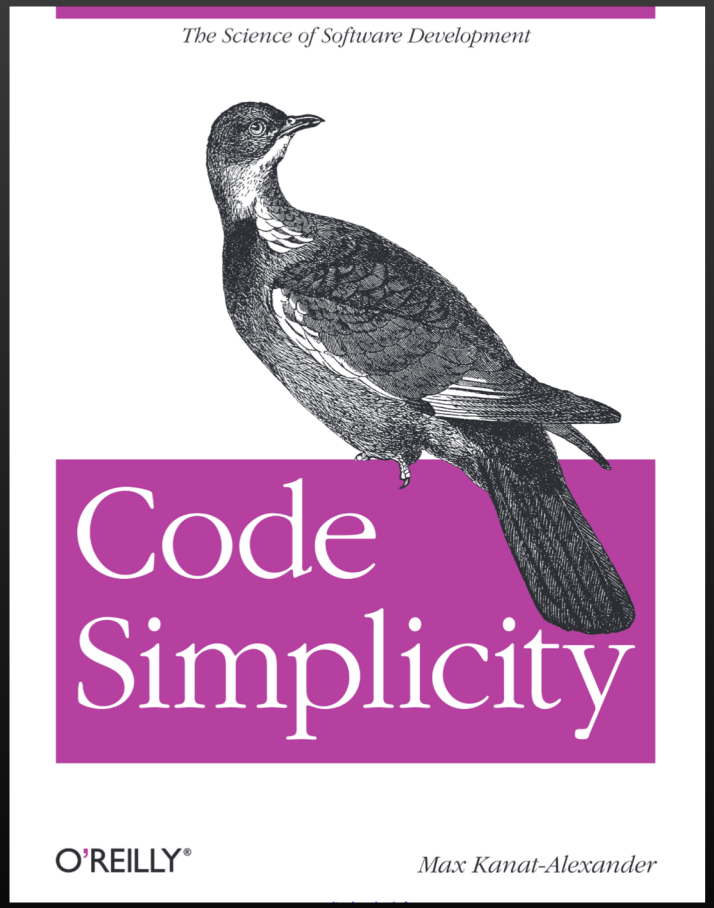
- shape
- colors

Complexity

- We tend to do more of what's already there
- Over-eagerness to DRY



Complexity



Kent's tweet in 70 pages – something to hit new developers over the head with

Duplication

Duplication

- Low-hanging fruit – for good and bad
- Be wary of wrong abstractions

Duplication < Complexity < Coupling < State