Writing testable code

Rodrigo Boniatti

Developer at Codeminer 42

@boniattirodrigo

rodrigoboniatti.com

Conditionals

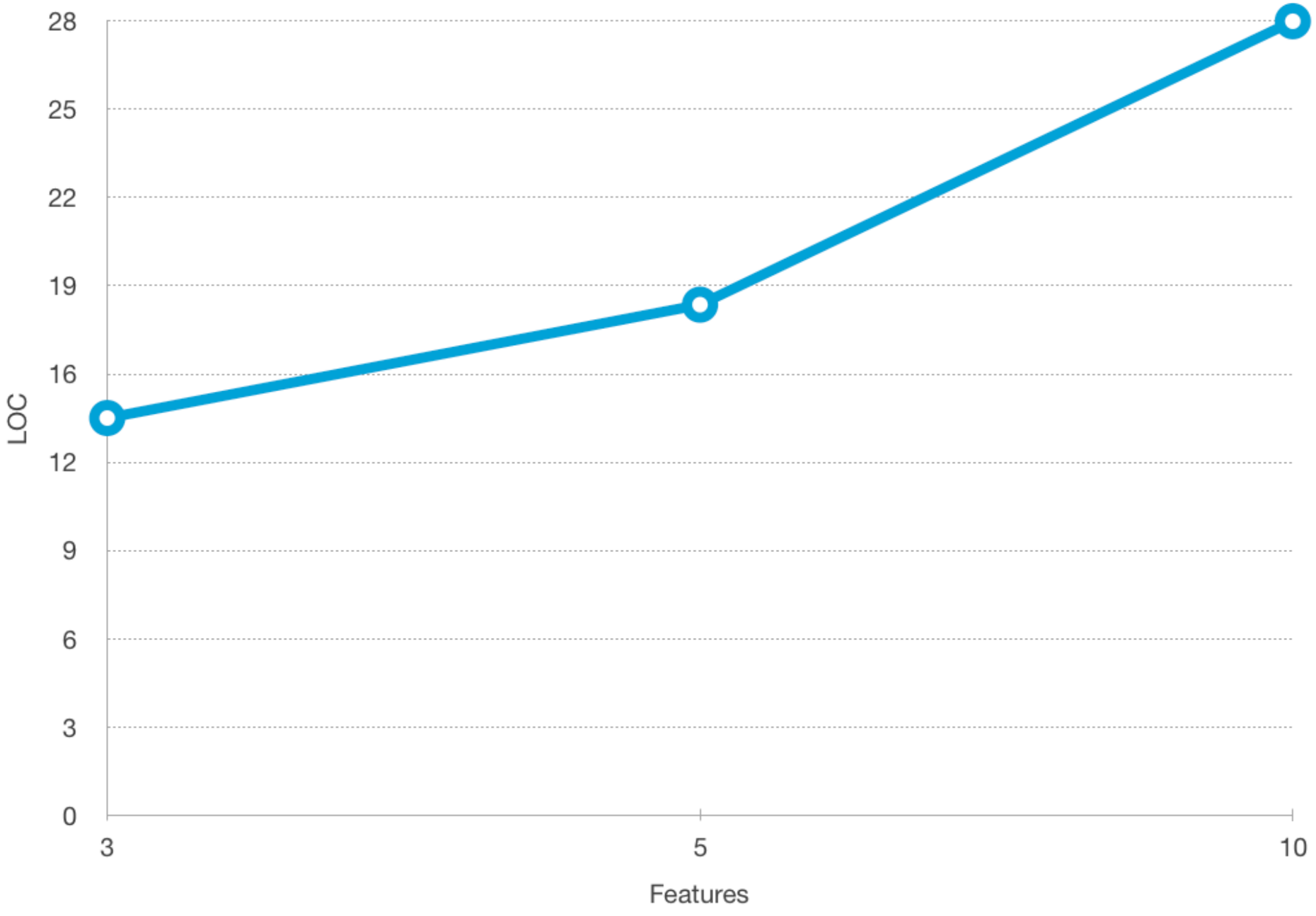
```ruby
class EcommerceA::TrackProductAccess
  def self.call(id)
  end
end

class EcommerceB::TrackProductAccess
  def self.call(id)
  end
end

class EcommerceC::TrackProductAccess
  def self.call(id)
  end
end
```

Smell

```ruby
class TrackProductAccess
  def self.call(product)
    case product.partner
    when 'EcommerceA'
      EcommerceA::TrackProductAccess.call(product.id)
    when 'EcommerceB'
      EcommerceB::TrackProductAccess.call(product.id)
    when 'EcommerceC'
      EcommerceC::TrackProductAccess.call(product.id)
    else
      puts 'Partner not found'
    end
  end
end
```


28
25
22
19
16
12
9
6
3
0
LOC
3
5
10
Features

Refactored

```
class TrackProductAccess
  def self.call(product)
    partner_module = product.partner.constantize
    partner_module::TrackProductAccess.call(product.id)
  end
end
```


20
18
16
13
11
9
7
4
2
0
LOC
3
5
10
Features

🚧 Conditionals

- **Smell:**
    - High cyclomatic complexity score;
- **How to solve?**
    - Replace Conditional with Polymorphism;

Global state

Smell

```vue
<template>
  <div class="container">
    <ul v-for="user in $store.state.users" v-bind:key="user" data-test="users-list">
      <li>{{ user }}</li>
    </ul>
  </div>
</template>

<script>
export default {
  name: 'UsersList',
  mounted() {
    this.$store.dispatch('fetchUsers')
  }
}
</script>
```

Refactored

```vue
<template>
  <UsersList :users="users" :fetchUsers="fetchUsers" />
</template>

<script>
import { mapActions, mapState } from 'vuex';
import UsersList from './UsersList.vue'

export default {
  name: 'UsersListContainer',
  computed: mapState(['users']),
  methods: mapActions(['fetchUsers']),
  components: {
    UsersList
  }
}
</script>
```

 Global state

- **Smell:**
    - Coupling;
- **How to solve?**
    - Dependency injection;

🔄 Life cycle events

Smell

```ruby
class User < ApplicationRecord
  after_create :send_welcome_email

  validates :name, :email, :age, presence: true

  private

  def send_welcome_email
    UserMailer.with(user: self).welcome_email.deliver_now
  end
end
```

Refactored

```ruby
class User < ApplicationRecord
  validates :name, :email, :age, presence: true
end


class CreateUserService
  def self.call(params)
    user = User.new(params)

    if user.save
      UserMailer.with(user: user).welcome_email.deliver_now
    end
  end
end
```

🔄 Life cycle events

- **Smell:**
    - Hide code behavior;
    - No execution order;
- **How to solve?**
    - Single flow;

Principles of testing

 Principles of testing

- Test in isolation;
- A lot of setup == smell;
- Hard to test == smell;

🤔 Questions?

 Thank you

Code examples: https://github.com/boniattirodrigo/writing-testable-code