HOW TO

REFACTORING

WHAT IS REFACTORING ?

- Improve code without adding new functionality

WHAT IS

CLEAN CODE

CLEAN CODE

- Easy to understand
- Doesn’t contain duplication
- Contain minimal number of parts
- Pass all test
- Easier to maintain

WHAT IS

TECHNICAL DEBT

MEANING

- **Technical debt** is a concept in programming that reflects the extra development work that arises when code that is easy to implement in the short run is used instead of applying the best overall solution.

CAUSE OF TECHNICAL DEBT

- Lack of test
- Lack of document
- Lack of communication
- Lack of Knowledge

WHEN TO REFACTOR

WHEN TO REFACTOR

- When adding a feature
- When fixing a bug
- During code review

HOW TO REFACTOR

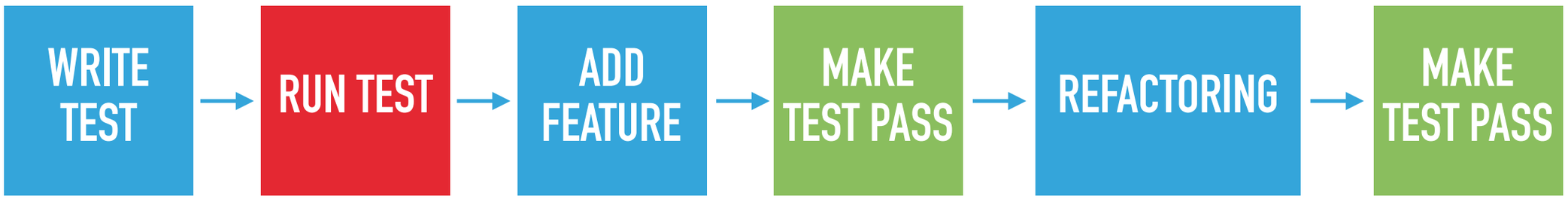
HOW TO REFACTOR

- Code should become cleaner
- No functionally add during refactoring
- All test must pass after refactoring

REFACTORING STEP

TRADITIONAL WAY

TDD [TEST DRIVEN DEVELOPMENT]



CODE SMELL

CODE SMELL

COMMENT CODE

PROBLEM

- Unnecessary part

HOW TO REFACTOR

- Make a good name for filename, variable, method or class

BENEFIT

- Understand code without comment

EXAMPLE

```
// get single product by send id

const getProduct = (id) => {

  ...

}
```

```
const getProductById = (id) => {

  ...

}
```

CODE SMELL

DEAD CODE

PROBLEM

- Fear to remove

HOW TO REFACTOR

- Write test !!

BENEFIT

- Small code size
- Easy to maintain
- Reduce communication

CODE SMELL

CODE FOR FUTURE

PROBLEM

- Create unused code
- More unnecessary complex code

HOW TO REFACTOR

- Write only use code
- Code coverage

BENEFIT

- Small code size
- Future code for future technique

CODE SMELL

DUPLICATE CODE

PROBLEM

- Hard to maintain if needs code change
- Duplicate test code

HOW TO REFACTOR

- Extract method
- Pull up field or method

EXTRACT METHOD

About.js

```
const About = (props) => (
  if (user.login !== undefined) {
    ...
  }
)
```

Product.js

```
const Product = (props) => (
  if (user.login !== undefined) {
    ...
  }
)
```

EXTRACT METHOD

About.js

```
const About = (props) => (
  if (user.isLogin()) {
    ...
  }
)
```

Product.js

```
const Product = (props) => (
  if (user.isLogin()) {
    ...
  }
)
```

User.js

```
const User = {
  isLogin() {
    if (user.login !== undefined) {
      ...
    }
  }
}
```

PULL UP FIELD OR METHOD

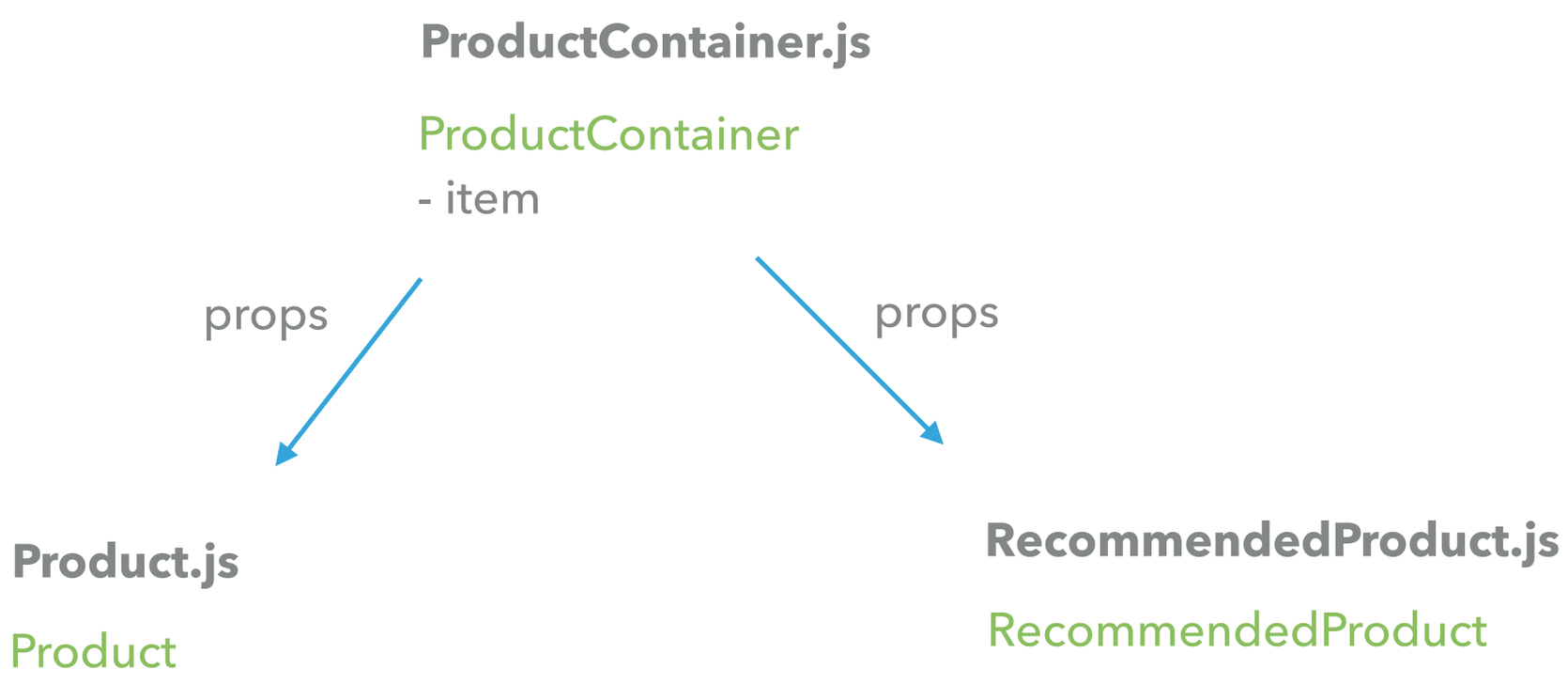
Product.js

Product
- item

RecommendedProduct.js

RecommendedProduct
- item

PULL UP FIELD OR METHOD



CODE SMELL

LONG METHOD

PROBLEM

- Hard to read
- Multiple responsibility

HOW TO REFACTOR

- Extract method

EXTRACT METHOD

```
const About extends Component {
  render() {
    return (
      do something
      do something
      do something
      do something
      do something
      do something
      do something
      do something
      do something
      do something
      ...
    )
  }
}
```

```
const About extends Component {
  doFirstThing() {
    return ...
  }

  doSecondThing() {
    return ...
  }

  render() {
    return (
      { this.doFirstThing() }
      { this.doSecondThing() }    )
  }
}
```

BENEFIT

- More readability
- Single responsibility
- Easy to test & debug

CODE SMELL

LOT OF TEMP VARIABLE

EXAMPLE

```
class Product extends Component {
 render() {
   const capitalizeName = this.props.product.name.charAt(0).toUpperCase() + this.props.product.name.slice(1)
   const shortDescription = this.props.product.description.substring(0, 50)
   const priceInUSD = this.props.price / 35
   return (
     <div>
       <h1>{capitalizeName}</h1>
       <p>{shortDescription}</p>
       <span>{priceInUSD}</span>
     </div>
   )
 }
}
```

PROBLEM

- Lack of readability
- Create unnecessary variable

HOW TO REFACTOR

- Replace temp variable with function call

BENEFIT

- More readability
- Improve performance
- Testability

EXAMPLE

```
class Product extends Component {
  render() {
    return (
      <div>
        <h1>{ CapitalizeName(this.props.product.name) }</h1>
        <p>{ getShortDescription(this.props.product.description) }</p>
        <span>{ convertPriceToUSD(this.props.product.price) }</span>
      </div>
    )
  }
}
```

CODE SMELL

LONG CONDITIONAL

PROBLEM

- Lack of readability
- Lack of reusability

HOW TO REFACTOR

- Decompose conditional to function

BENEFIT

- More readability
- Reusable condition
- More testability

EXAMPLE

```
class Product extends Component {
  render() {
    return (
      <div>
       {  this.props.product.isAvaliable === true && this.props.product.quantity !== 0 &&
          <AddToCart />
      </div>
    )
  }
}
```

EXAMPLE

```
class Product extends Component {
  render() {
    const isProductAvaiable = this.props.product.isAvaliable === true && this.props.product.quantity !== 0
    return (
      <div>
       {  isProductAvaliable &&
          <AddToCart />
       </div>
    )
  }
}
```

EXAMPLE

```
class Product extends Component {
  isProductAvaliable() {
    return this.props.product.isAvaliable === true && this.props.product.quantity !== 0
  }

  render() {
    return (
      <div>
        {  this.isProductAvaliable() &&
          <AddToCart />
        </div>
    )
  }
}
```

CODE SMELL

LONG PARAMETER LIST

EXAMPLE

```
class Product {
  addToCart(productId, productName, isProductAvaliable, Quantity) {
    return ...
  }
}
```

REASON

- Add feature without refactoring

HOW TO REFACTOR

- Use object instead
- Replace parameter with method call

EXAMPLE

```
class Product {
  addToCart(product, Quantity) {
    return ...
  }
}
```

EXAMPLE

```
class Product {
  addToCart(getProduct(), Quantity) {
    return ...
  }
}
```