CLEAN CODE for the GREAT UNWASHED

Logic Room

Matthew Wasbrough

Developer @

Logic Room

Thanks to Pete

- Business side of things
- Why we do what we do

I'm going to talk about

- Bringing Pete's vision to life
- What we aim to do
- How we are doing it

What I'm going to talk about

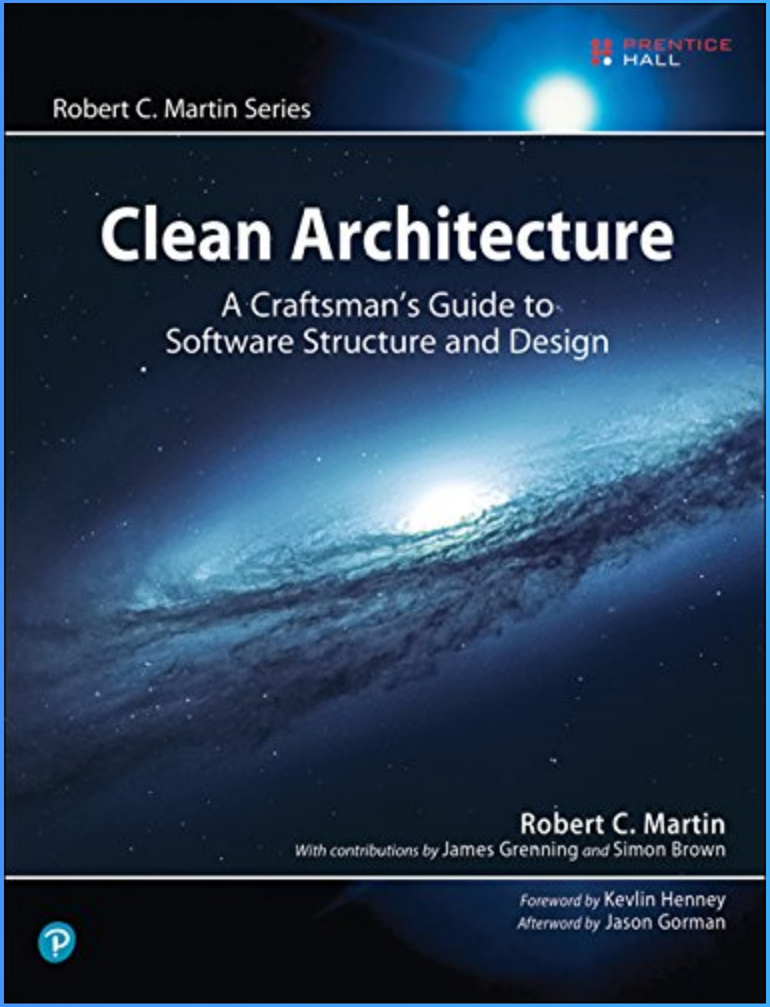
- What I/We see as problems currently in JavaScript Development
- Lessons we can learn from other languages
- The SOLID Principles
- In practice what we are doing with code examples
- What this architechture gives us as a development team

- How many JavaScript/TypeScript Developers are in the room?

The strength of JavaScript is that you can do anything.
The weakness is that you will.

- Reg Braithwaite

- How many people know about Clean Architecture?
- How many people have heard of Robert C Martin (Uncle Bob)?
- How many people have heard of the Solid Principles?


PRENTICE HALL
Robert C. Martin Series
Clean Architecture
A Craftsman's Guide to Software Structure and Design
Robert C. Martin
With contributions by James Grenning and Simon Brown
Foreword by Kevlin Henney
Afterword by Jason Gorman

Clean Architecture - Robert (Uncle Bob) Martin

SOLID Principles

- S - The Single Responsibilty Principle
- O - The Open-Closed Principle
- L - The Liskov Substitution Principle
- I - The Interface Segregation Principle
- D - The Dependency Inversion Principle

SOLID Principles

- S - The Single Responsibilty Principle
- O - The Open-Closed Principle
- L - The Liskov Substitution Principle
- I - The Interface Segregation Principle
- D - The Dependency Inversion Principle

- S - The Single Responsibilty Principle
Each Module has one and only one reason to change.
- O - The Open-Closed Principle
Systems should consist of modules which can be extended instead of being changed

- This is still a work in progress
- Even when complete this will always be our interpretation

> **This is likely the reason why so many systems lack good architecture: They begun with none, becuase the team was small and did not want the impediment of a superstructure.**

- Robert C Martin - Clean Architecture

SINGLE FILE REACT APP

```
import * as React from 'react'
import styled from '@emotion/styled'
import request from 'superagent'
import moment from 'moment'

const url = 'http://localhost:3001'

const Wrapper = styled.div`
  background-color: #f5f5f5;
  padding: 40px;
`


const Heading = styled.h1`
  color: pink;
`


const TodosList = styled.ul`
  list-style-type: none;
  margin: 0 auto;
  width: 400px;
```

Look through the code

```
import * as React from 'react'
import styled from
'@emotion/styled'
import request from 'superagent'
import moment from 'moment'

const url =
'http://localhost:3001'

const Wrapper = styled.div`
  background-color: #f5f5f5;
  padding: 40px;
`
```

```
const Wrapper = styled.div`
  background-color: #f5f5f5;
  padding: 40px;
`

const Heading = styled.h1`
  color: pink;
`

const TodosList = styled.ul`
  list-style-type: none;
  margin: 0 auto;
  width: 400px;
  padding: 20px 0 0 0;
`

const TodosListItem = styled.li`
```

Styles

```
const DateSpan = styled.span`
  color: darkgrey;
  font-size: 0.8em;
`

interface ITodo {
  title: string
  complete: boolean
  id: number
  created: string
}

const Input = (props: {
  value: string
  changeVal: (value: string) => void
  submitTodo: () => void
}) => {
```

TypeScript Interface

```
const Input = (props: {
  value: string
  changeVal: (value: string) => void
  submitTodo: () => void
}) => {
  return (
    <form>
      <input
        type="text"
        placeholder="todo"
        onChange={e => {
          return props.changeVal(e.target.value)
        }}
        value={props.value}
      />
```

Functional Component (Input)

```
const List = (props: {
  todos: ITodo[]
  toggleComplete: (id: number) => void
  deleteTodo: (id: number) => void
}) => {
  return props.todos.length > 0 ? (
    <TodosList>
      {props.todos.map(el => {
        return (
          <TodosListItem key={el.title + '-' + el.id}>
            <TodoItem complete={el.complete}>
              {el.title}
              <DateSpan> - {el.created}</DateSpan>
```

Functional Component (List)

```
    </TodosList>
  ) : null
}

interface ITodoListProps {}

interface ITodoListState {
  todos: ITodo[]
  value: string
  loadingData: boolean
}

class TodoList extends React.Component {
  constructor(props: ITodoListProps) {
    super(props)
    this.state = {
```

Main TS interface

```
  loadingData: boolean
}

class TodoList extends React.Component {
  constructor(props: ITodoListProps) {
    super(props)
    this.state = {
      todos: [],
      value: '',
      loadingData: true
    }
  }

  state: ITodoListState

  componentDidMount() {
    this.getTodos()
```

The main React Class Componenet

```
componentDidMount() {
 this.getTodos()
}

getTodos = async () => {
 try {
  const { body } = await request.get(url)
  if (body.success && body.todos) {
   const newTodos: ITodo = body.todos.map((todo: ITodo) => {
    return {
     ...todo,
     created: moment(todo.created).format('HH:mm:ss DD/MM/YYYY')
    }
```

Lifecyle Methods and fetches - Get Todos

```
  submitTodo = async () => {
    try {
      const todo = {
        title: this.state.value,
        complete: false
      }
      await request
        .post(url)
        .set('Content-Type', 'application/json')
        .send(todo)
      this.setState({ value: '' })
      await this.getTodos()
      return
    } catch (e) {
```

Further requests - Save Todo

```
toggleTodo = async (index: number) => {
 try {
  const oldTodo = this.state.todos.find(todo => todo.id === index)
  let newTodo = {}
  if (oldTodo && oldTodo.hasOwnProperty('complete')) {
   newTodo = { ...oldTodo, complete: !oldTodo.complete }
  } else {
   return
  }
  await request
   .put(url)
   .set('Content-Type', 'application/json')
   .send(newTodo)
```

Update Todo

```
    }
  }

  deleteTodo = async (index: number) => {
    try {
      await request.del(`${url}/${index}`)
      await this.getTodos()
      return
    } catch (e) {
      return
    }
  }

  render() {
    return (
      <Wrapper>
        <Heading>Todo
```

Delete Todo

```
  }

  render() {
    return (
      <Wrapper>
        <Heading>Todo List</Heading>
        <Input
          value={this.state.value}
          changeVal={this.changeVal}
          submitTodo={this.submitTodo}
        />
        <List
          todos={this.state.todos.filter(todo =>
```

Main Render

```
          todos={this.state.todos.filter(todo => todo.complete)}
          toggleComplete={this.toggleTodo}
          deleteTodo={this.deleteTodo}
        />
        <List
          todos={this.state.todos.filter(todo => !todo.complete)}
          toggleComplete={this.toggleTodo}
          deleteTodo={this.deleteTodo}
        />
      </Wrapper>
```

Render two lists

- Styles
- Input Functional Component
- List Functional Component
- Main React Class Component
- Get Todos
- Save Todo
- Update Todo
- Delete Todos
- Render App

- S - The Single Responsibilty Principle
Each Module has one and only one reason to change.
- O - The Open-Closed Principle
Systems should consist of modules which can be extended instead of being changed

Now lets break this code down into smaller pieces to make it easier to work with.

MobX

https://mobx.js.org/

mobx-react

https://github.com/mobxjs/mobx-react

SINGLE FILE REACT APP (ONLY REACT CODE)

```
import * as React from 'react'
import styled from '@emotion/styled'
import request from 'superagent'
import moment from 'moment'
import { observer } from 'mobx-react'
import { TodoPresenter } from './presenter'

const Wrapper = styled.div`
  background-color: #f5f5f5;
  padding: 40px;
`


const Heading = styled.h1`
  color: pink;
`


const TodosList = styled.ul`
  list-style-type: none;
```

SINGLE FILE REACT APP (MOBX STATE)

```
import { observable, action } from 'mobx'
import request from 'superagent'
import moment from 'moment'

interface ITodo {
  title: string
  complete: boolean
  id: number
  created: string
}

const url = 'http://localhost:3001'

export class TodoPresenter {
  constructor() {
    this.getTodos()
  }

  @observable
  public value: string = ''
```

Does this code obey the SO principles?

- S - The Single Responsibilty Principle
Each Module has one and only one reason to change.
- O - The Open-Closed Principle
Systems should consist of modules which can be extended instead of being changed

Still need to pull the code apart a little more

SINGLE FILE REACT APP (MOBX STATE WITHOUT FETCHES)

```
import { observable, action } from 'mobx'
import moment from 'moment'

import { todoGateway } from './gateway'

interface ITodo {
  title: string
  complete: boolean
  id: number
  created: string
}

export class TodoPresenter {
  constructor() {
    this.getTodos()
  }

  @observable
```

SINGLE FILE REACT APP (MOBX STATE ONLY FETCHES)

```
import request from 'superagent'

interface IServerTodo {
  title: string
  complete: boolean
  id: number
  created: string
}

const url = 'http://localhost:3001'

export class TodoGateway {
  public get = async (): Promise<{
    success: boolean
    todos: IServerTodo[]
  }> => {
    try {
      const response: {
```

Does this code now obey the SO principles?

Just looking at the imports from files

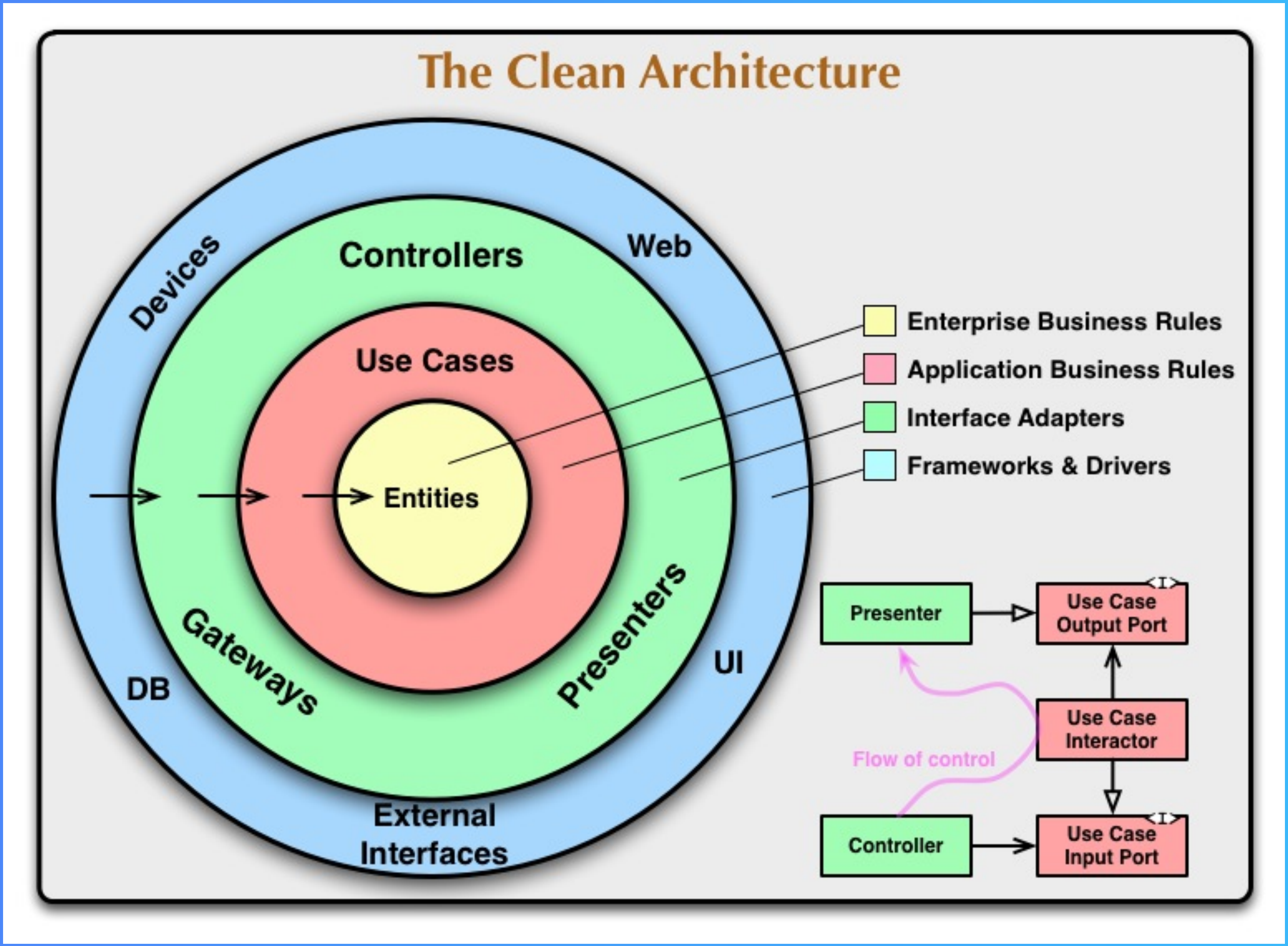
Before the refactor	**Refactor 1**	**Refactor 2**
React, Emotion, Moment, Superagent	React, Emotion	React, Emotion
	Moment, Superagent	Moment
		Superagent

A Backend Service

Express

Business Logic

Database Calls



http://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

GATEWAY -> REPOSITORY -> PRESENTER -> VIEW

Gateway -> Repository

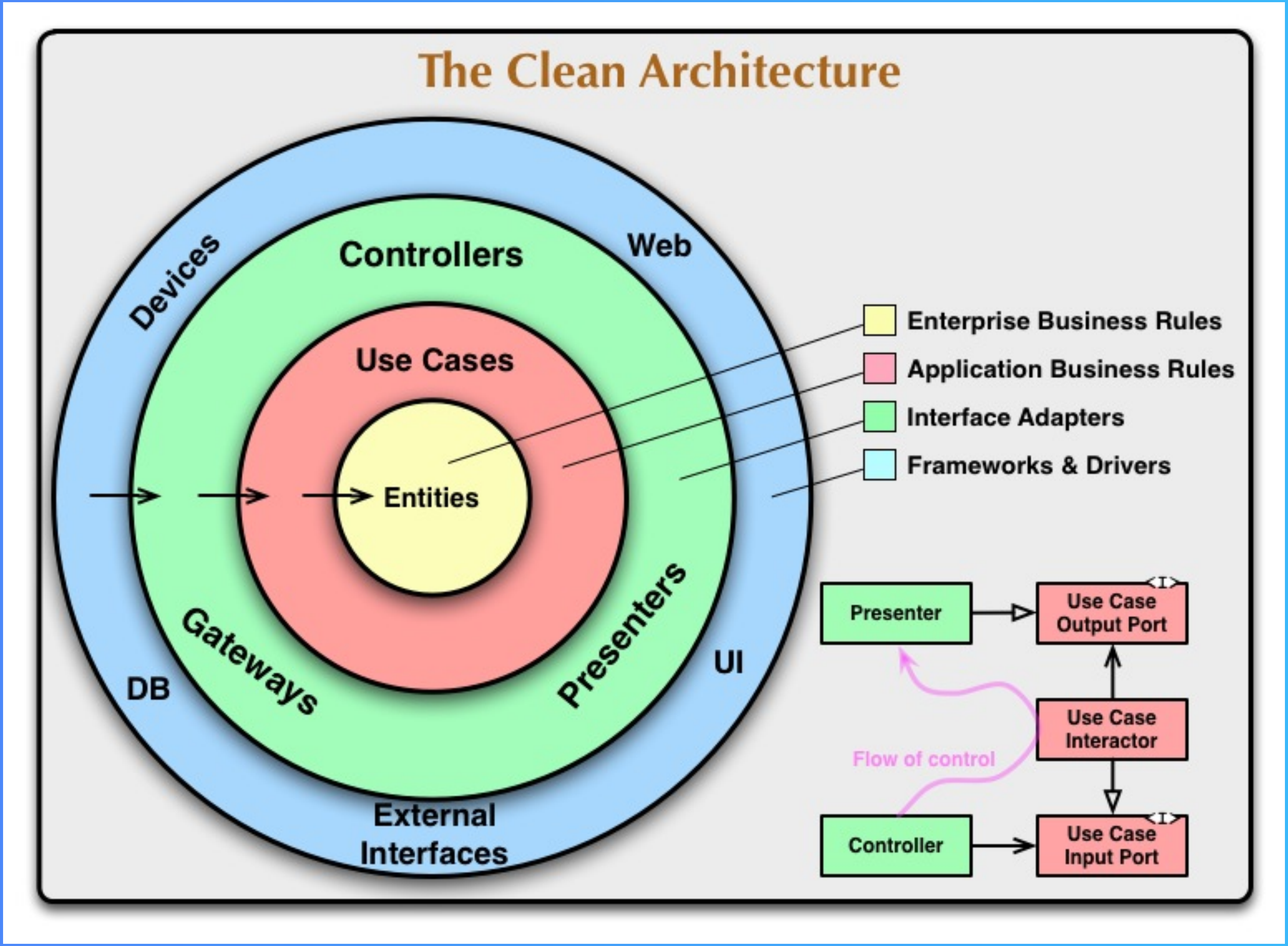
```
{
  ...,
  createdDate: moment(DateFromGateway),
  ...
}
```

Repository -> Presenter 1

```
{
  ...,
  createDate: repository.createdDate.format('DD/MM/YYYY'),
  ...
}
```

Repository -> Presenter 2

```
{
  ...,
  createdDate: repository.createdDate.format('DD/MM/YYYY HH:mm:ss'),
  ...
}
```



http://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

DOES THIS APP BENEFIT FROM THE CLEAN ARCHITECTURE?

For larger apps, what does this give us?

1. Maintainability
2. Reusability
3. Testability
4. Team work

- This is still a work in progress
- Even when complete this will always be our interpretation

Thank you for your attention.
Any questions?
https://github.com/matthew-was/ca_ts_todolist