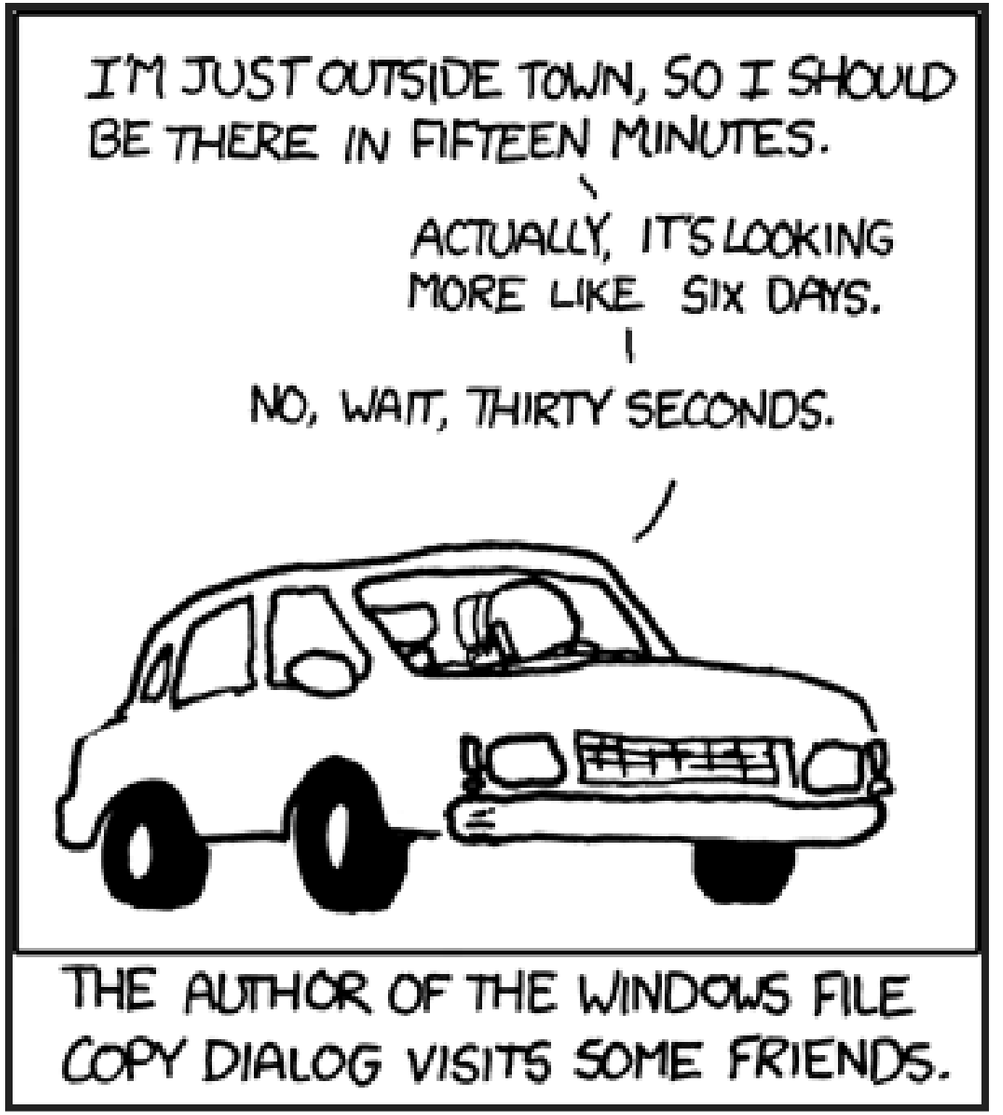
Recompose

Recompose your React.js components embracing a functional style

Connect.tech 2017

Jeff Barczewski

- @jeffbski
- jeff@codewinds.com
- https://codewinds.com/connect2017



Estimation by xkcd Licensed CC BY 2.5

Incident by xkcd Licensed CC BY 2.5

Jeff Barczewski

- Married, Father, Catholic
- 27 yrs (be nice to the old guy :-)
- JS (since 95, exclusive last 5 years)
- Open Source: redux-logic, pkglink, …
- Founded CodeWinds, live/online training (React, Redux, Immutable, RxJS) – I love teaching, contact me

CodeWinds Training

- Live training (in-person or webinar)
- Self-paced video training classes (codewinds.com)
- Need training for your team on any of these?
  - React
  - Redux
  - RxJS
  - JavaScript
  - Node.js
  - Functional approaches
- I'd love to work with you and your team
- I appreciate any help in spreading the word.

My early career

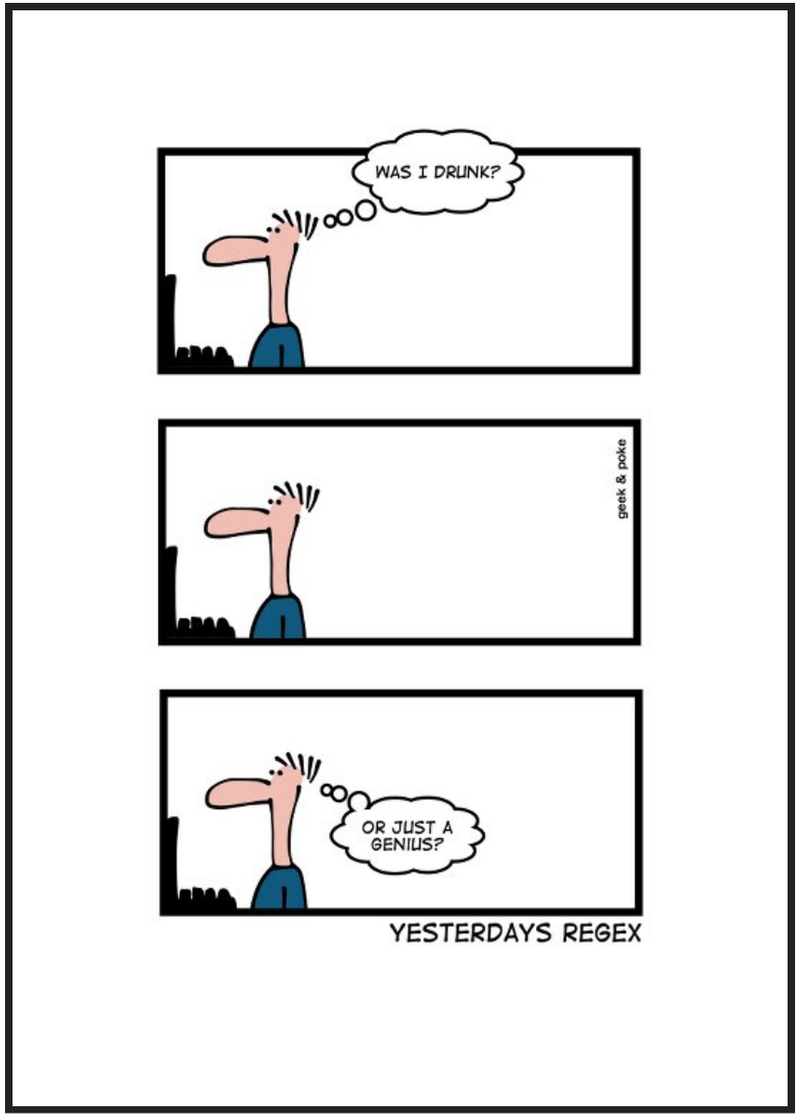
- Aerospace Engineer – US Air Force, ASD/XR Wright Patterson AFB, Ohio
  - Fortran 77 with extensions
  - C
  - C++ / Interviews
- Consulting
  - C++
  - Java

OO JavaScript

```
class App extends Component {
  constructor(props) {
    super(props)
    this.state = { todos: [] };
  }
  addTodo = (text) => {
    const todos = [
      {
        id: createUniqueID(),
        completed: false,
        text: text
      },
      ...this.state.todos
    ]
    this.setState({todos});
  }
  render() {
    return (
      <div>
        <Header addTodo={this.actions.addTodo} />
        <MainSection todos={this.state.todos} actions={this.actions} />
      </div>
```

OOP Advantages

- Methods live close to the data they work with
- Reuse through inheritance
- Encapsulation - protect data – changes performed by methods
- Flexibility through Polymorphism – obj.draw()



Yesterday's Regex by Geek & Poke Licensed CC BY 3.0

Object Oriented Programming

OOP Shortcomings

- Clutter – classes quickly grow in any real usage, everything is commingled - Hard to reason about
- Inheritance spreads implementation over many classes/files
- OOP favors mutation
- Cross domain sharing is difficult

Functional Programming

Functional JS Advantages

- Easy to reason about especially for pure functions
- Composition is simple
- Separation of concerns – do one thing well
- Testing is easy for stateless functions
- Treating data immutably reduces surprises and enables features like undo and auditing
- Event driven system like Redux creates predictable one directional data flow and makes it easy to react in multiple domains

Functional JS

```
// simple, testable, functional component
export const App = ({todos, actions}) => (
  <div>
    <Header addTodo={actions.addTodo} />
    <MainSection todos={todos} actions={actions} />
  </div>
)

// connect returns a HOC wrapped component
export default connect(
  state => ({       // maps redux state to props
    todos: state.todos
  }),
  dispatch => ({    // binds action creators to dispatch
    actions: bindActionCreators(TodoActions, dispatch)
  })
)(App);
```

Higher Order

- **Higher Order Functions** – function that takes a function and returns a function

```
const twice = (fn, x) => fn(fn(x));
const fn = x => x + 3;
const result = twice(fn, 7);  // (7+3)+3 = 13
```

- **Higher Order Components** (HOC) – function that takes a component and returns a component

```
const Profile = ({ name }) => <div>{ name }</div>;
const PureProfile = pure(Profile);
ReactDOM.render(<PureProfile name={myName} />, div);
```

Functional Composition

```
const getCurrentDate = () => new Date();
const format = date => date.toLocaleString();
const createTodayMessage = formattedDate => `Today is ${formattedDate}`;

const str = createTodayMessage(format(getCurrentDate()));

const createFormattedTodayMessage = compose(
  createTodayMessage,
  format
);

const str2 = createFormattedTodayMessage(getCurrentDate());
```

Stateless Function Components

```
function Orders({ orders }) {
  return (
    <ul>
      { orders.map(x => (
        <li key={ x.id }>{ x.name } - { x.date }</li>
      ))}
    </ul>
  );
}
```

Stateless function components

(ES6 arrow functions)

```
const Orders = ({ orders }) => (
  <ul>
    { orders.map(x => (
      <li key={x.id}>{ x.name } - { x.date }</li>
    ))}
  </ul>
);
```

Stateless Function Components

Advantages

- Extremely simple, easy to reason about
- Easy to test, no state, pass props to test modes
- Pure functions – input + output, nothing else
- No need to use “this”, props are local variables
  - Modular and reusable
- FaceBook team will continue to create optimizations. Favors functional style.

How do I handle X?

- state
- handlers
- shouldComponentUpdate optimizations
- lifecycle methods
  - dynamic loading

Redux / react-redux connect

```
// simple, testable, functional component
export const App = ({todos, actions}) => (
  <div>
    <Header addTodo={actions.addTodo} />
    <MainSection todos={todos} actions={actions} />
  </div>
)

// connect returns a HOC wrapped component
export default connect(
  state => ({       // maps redux state to props
    todos: state.todos
  }),
  dispatch => ({    // binds action creators to dispatch
    actions: bindActionCreators(TodoActions, dispatch)
  })
)(App);
```

react-redux connect

- locates just the data needed from redux store
- binds action creators to dispatch
- rerenders whenever our props change
- implements shouldComponentUpdate - optimized renders

```
// returns a HOC wrapped component
export default connect(
  state => ({       // maps redux state to props
    todos: state.todos
  }),
  dispatch => ({    // binds action creators to dispatch
    actions: bindActionCreators(TodoActions, dispatch)
  })
)(App);
```

Still need more?

- need to have state outside of Redux
- handlers to operate on that state
- specialized shouldComponentUpdate optimizations
- lifecycle methods
  - dynamic loading
- adapting a component for use in different ways
  - different defaults
  - remapping data to the proper props

Recompose

- Utility library for creating HOC's
  - "The lodash for React"
- npm install recompose
- Compose in common functionality using parameterized HOC functions
- Allows you to stay in the functional world
  - Stateless function components

https://github.com/acdlite/recompose

Solving problems with Recompose

```
const PureProfile = pure(Profile);  // optimized

const OptimProfile = onlyUpdateForKeys(['name', 'age'])(Profile);

const DefaultedComp = defaultProps({
  greeting: 'Hello'
})(Comp);

const Comp2 = renameProp('first', 'firstName')(Profile);

const Comp3 = renameProps({
  first: 'firstName',
  last: 'lastName'
})(Profile);
```

withProps

```
const Comp = withProps(props => ({
  fullName: `${props.first} ${props.last}`,
  mode: 1
}))(EchoProps);
ReactDOM.render(<Comp first="John" last="Smith" />, div);
```

withPropsOnChange - arr style

```
const Comp = withPropsOnChange(
  ['first', 'last'],
  props => {
    console.log('recalculating fullName');
    return {
      fullName: `${props.first} ${props.last}`
    };
  }
)(EchoProps);
ReactDOM.render(<Comp first="John" last="Smith" />, div);
ReactDOM.render(<Comp first="John" last="Smith" />, div);
ReactDOM.render(<Comp first="Jenny" last="Smith" />, div);
```

withPropsOnChange - fn style

```
const Comp = withPropsOnChange(
  (props, nextProps) =>
    props.first !== nextProps.first || props.last !== nextProps.last,
  props => {
    console.log('recalculating fullName');
    return {
      fullName: `${props.first} ${props.last}`
    };
  }
)(EchoProps);
ReactDOM.render(<Comp first="Bill" last="Wonder" />, div);
ReactDOM.render(<Comp first="Bill" last="Wonder" />, div);
ReactDOM.render(<Comp first="Mary" last="Wonder" />, div);
```

mapProps

```
const Comp = mapProps(props => ({
  firstName: props.first,
  lastName: props.last
}))(EchoProps);
ReactDOM.render(<Comp first="John" last="Smith" />, div);
```

composing

```
const Comp = compose(
  pure,
  withProps(props => ({
    fullName: `${props.first} ${props.last}`,
    mode: 1
  }))
)(EchoProps);
ReactDOM.render(<Comp first="John" last="Smith" />, div);
ReactDOM.render(<Comp first="John" last="Smith" />, div);
ReactDOM.render(<Comp first="Amanda" last="Green" />, div);
```

example form

```
const Form1 = ({ values, onChange, onSubmit }) => {
  return (
    <form onSubmit={onSubmit}>
    <input name="first" value={values.first} onChange={onChange} />
    <input name="last" value={values.last} onChange={onChange} />
    <button>Submit</button>
    </form>
  );
};
```

composing to use with our form

```
import { set } from 'lodash/fp';

const Comp = compose(
  withState('values', 'updateValues', { first: '', last: ''}),
  withHandlers({
    onChange: ({ updateValues }) => ev => {
      const {name, value} = ev.target;
      updateValues(x => set(name, value, x));
    },
    onSubmit: ({ values, updateValues }) => ev => {
      ev.preventDefault();
      console.log('submit', values); // do something with data
      updateValues(x => ({ first: '', last: '' }));
    }
  })
)(Form1);
```

withStateHandlers

```
const Comp = withStateHandlers(
  props => ({   // create initial state
    first: '',
    last: ''
  }),
  {  // state handlers
    onChange: props => ev => {
      const {name, value} = ev.target;
      return {  // return the state changes to merge
        [name]: value
      };
    },
    onSubmit: props => ev => {
      ev.preventDefault();
      console.log('submit', props)); // do something with data
      // return the state changes to merge
      return { first: '', last: '' };
    }
  }
)(Form1);
```

branch, renderComponent

```
const Loading = props => <div>Loading...</div>;
const Loaded = ({ content }) => <div>Loaded { content }</div>;

const Comp = branch(
  props => !props.content,
  renderComponent(Loading)
)(Loaded);

ReactDOM.render(<Comp />, div, () => {
  console.log(div.innerHTML);
  ReactDOM.render(<Comp content="hello" />, div, () => {
    console.log(div.innerHTML);

  });
});
```

branch, renderNothing

```
const Loading = props => <div>Loading...</div>;
const Loaded = ({ content }) => <div>Loaded { content }</div>;

const Comp = branch(
  props => !props.content,
  renderNothing
)(Loaded);

ReactDOM.render(<Comp />, div, () => {
  console.log(div.innerHTML);
  ReactDOM.render(<Comp content="hello" />, div, () => {
    console.log(div.innerHTML);

  });
});
```

composing branch, renderComponent, and lifecycle

```
const Loading = props => <div>Loading...</div>;
const MainContent = ({ items }) => <ul>
  { items.map(x => <li key={x.id}>{x.name}</li> )}
</ul>;

const DynamicContent = compose(
  lifecycle({
    componentDidMount() {
      axios.get('http://yourserver.com')
        .then(res => { this.setState({ items: res.data.result}); });
    }
  }),
  branch(props => !props.content, renderComponent(Loading))
)(MainContent);
```

Summary

- Functional JavaScript is fun!
- Functional JavaScript is easy to reason about
- Stateless Function Components are awesome!
  - easy to reason about
  - easy to test
- Recompose - the lodash of React HOC's
  - Wraps our component with additional functionality
  - Compose and layer on new features
  - Adapt or remap data to our functional components
  - Add state, handlers, lifecycle methods, pure optimizations

Thanks

- https://codewinds.com/connect2017 (slides, resources)
- https://codewinds.com/ (newsletter tips/training)
- jeff@codewinds.com
- @jeffbski