

# React

- [React - Optimize Performance](#)
- [React - Testing](#)
- [React - HOC \(Higher Order Component\)](#)

# React - Optimize Performance

## Case 1 - Preventing Wasted Renders with Complex Props

Create new object reference before delete a key.

```
function handleDelete(card) {  
  const clonedCards = { ...cards };  
  delete clonedCards[card.id];  
  setCards(clonedCards);  
}
```

### Class

```
3  
4 export class Summary extends React.Component {  
5   shouldComponentUpdate(nextProps) {  
6     const oldKeys = Object.keys(this.props.cards);  
7     const newKeys = Object.keys(nextProps.cards);  
8  
9     console.log({  
10      oldLength: oldKeys.length,  
11      newLength: newKeys.length  
12    });  
13  
14    return oldKeys.length !== newKeys.length;  
15  }  
16 }
```

### Function

```

export const Summary = React.memo(function Summary(props) {
  const cards = Object.values(props.cards);

  const distances = { max: 0, min: 100000 };
  cards.forEach(currentCard => {
    cards.forEach(compareCard => {
      if (compareCard === currentCard) {
        return;
      }
      const distance = levenshtein(currentCard.label, compareCard.label);

      distances.max = Math.max(distances.max, distance);
      distances.min = Math.min(distances.min, distance);
    });
  });

  return (
    <div>

```

```

    </div>
  ), (p1, p2) => Object.keys(p1.cards).length === Object.keys(p2.cards).length);

```

## Case 2 - Preventing Wasted Renders in a Simple Component

### Class

```

import React from 'react';

export class AddButton extends React.PureComponent {
  render() {
    const { onClick } = this.props;

    return (
      <button
        onClick={onClick}

```

### Function

#### Before

```

<AddButton onClick={() => setIsAddOpen(true)} />

const showDialog = useCallback(() => setIsAddOpen(true), []);

```

#### After

```

<AddButton onClick={showDialog} />

```

## Caching Expensive Operation Results

```
useMemo(
```

## Lazy Loading Components

```
useCallback,  
  lazy,  
  Suspense  
} from "react";
```

Component:

```
const AddModal = lazy(() => import("./AppModal"));
```

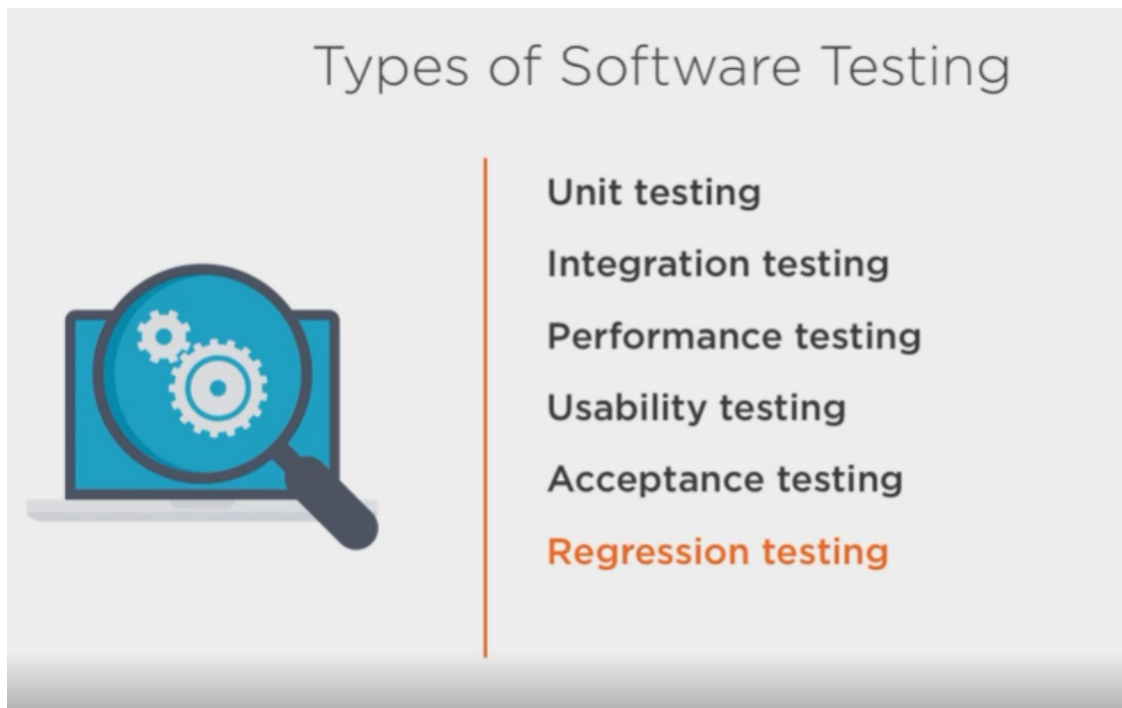
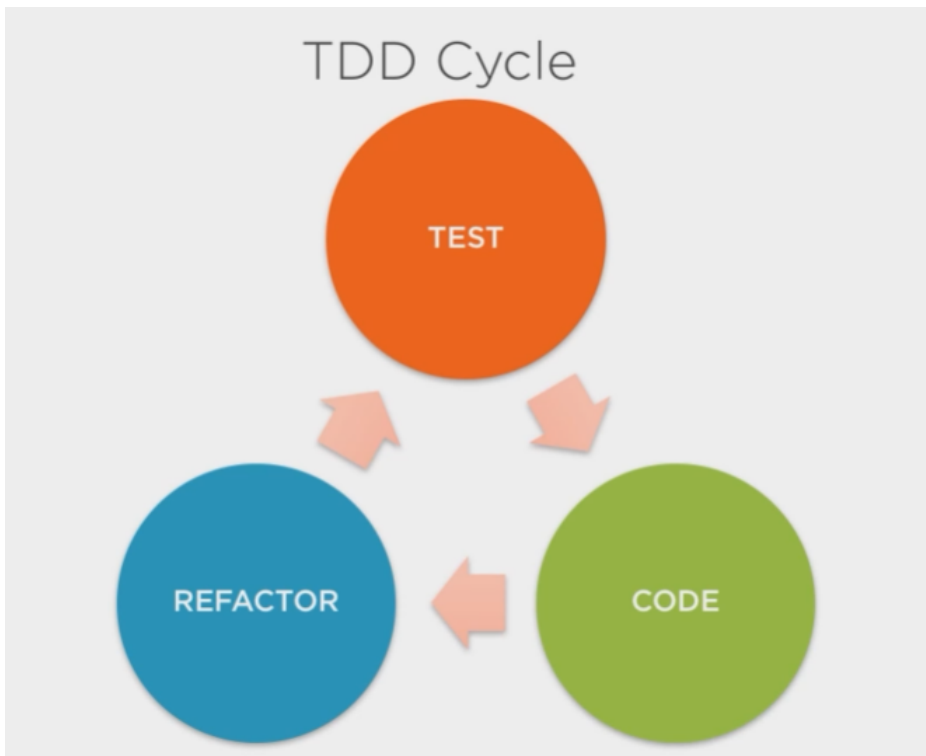
Fallback:

```
const ModalLoader = () => <div className="Modal-loader">Loading</div>;
```

```
{isAddOpen && (  
  <Suspense fallback={<ModalLoader>}>  
    <AddModal  
      isOpen={isAddOpen}  
      onClose={() => setIsAddOpen(false)}  
      onAdd={cardText => {  
        const updatedCards = positionCards(  
          addCard(cards, cardText),  
          width,  
          height  
        );  
        setCards(updatedCards);  
      }}  
    />  
  </Suspense>  
)}
```



# React - Testing



## What Makes a Good Test?



Each test should be independent of the others

Any behavior should be specified in only one test

No unnecessary assertions

Test only one code unit at a time

**Avoid unnecessary preconditions**

## How to Talk Testing



Suite

Spec

Assertion

Matcher

**Test runner**

# React - HOC (Higher Order Component)

```
import React, {ReactElement} from 'react';

const withAds = (Component: React.ComponentType) => {
  return (props): ReactElement => {

    // Write logic here.....

    return <Component {...props} />;
  };
};

export default withAds;
```