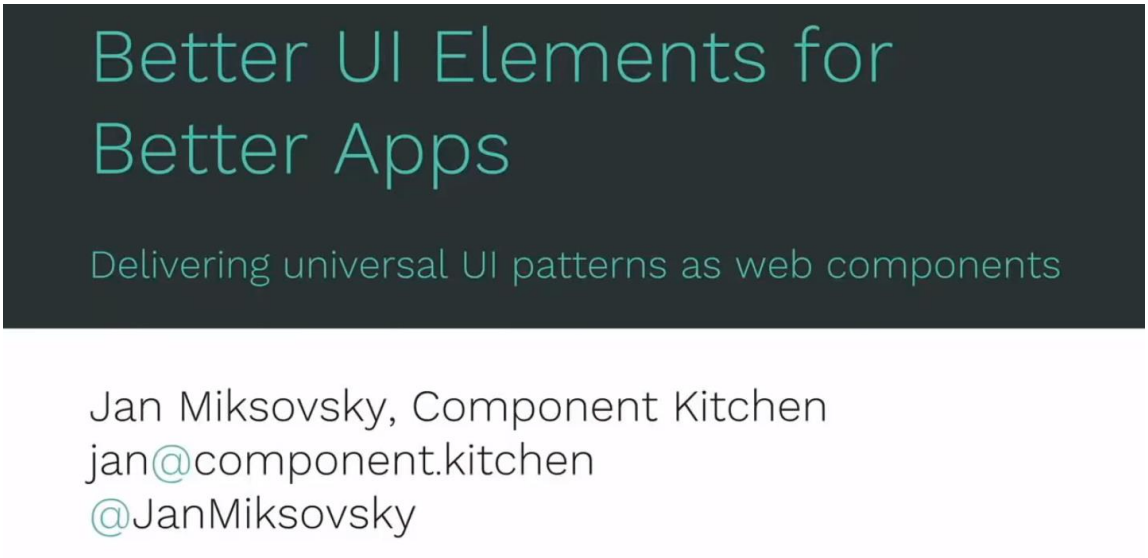



Why do we see so little reuse of web UI across organizations? Web projects freely use countless open libraries from npm and elsewhere, but it's rare to find high-quality web UI code which is flexible and reliable enough for immediate reuse. This makes creating great user experiences both difficult and expensive. This talk begins by looking at root causes of this problem in search of insights that might help address it.


Jan then discusses the strategies of his work on the Elix project, which seeks to make all common, general-purpose UI patterns available as high-quality, reusable web components. Implementing UI patterns as standard web components transforms them from a theoretical exercise into live, production-ready code that can quickly and directly express a designer's intention in any web framework.

Techniques

Strategy

Opportunity

Symptoms

Problem — Poor supply chain for UI parts prevents specialization

Causes

The talk concludes with practical techniques for deconstructing complex components into UI primitives that can be readily recombined to create UI elements tuned for an application's particular needs and aesthetics.

Wood
+ Glass
+ Nails

= Window

A modern window is an unbelievably complex machine

Modern windows are made in factories by specialists

A builder creates framing for the factory parts

Handmade windows
can stick or leak

Properly installed
factory windows
almost never leak

Builders assemble
modern buildings
from parts

We still build UI
ourselves
by hand
from raw materials

## Interactive HTML elements

| | |
|---|---|
| <input> | <select> |
| <textarea> | <option> |
| <button> | <audio> |
| <a> | <video> |

## Problem statement

A poor supply chain for UI parts prevents specialization, which leads to inefficiency and low quality.

Techniques

Strategy

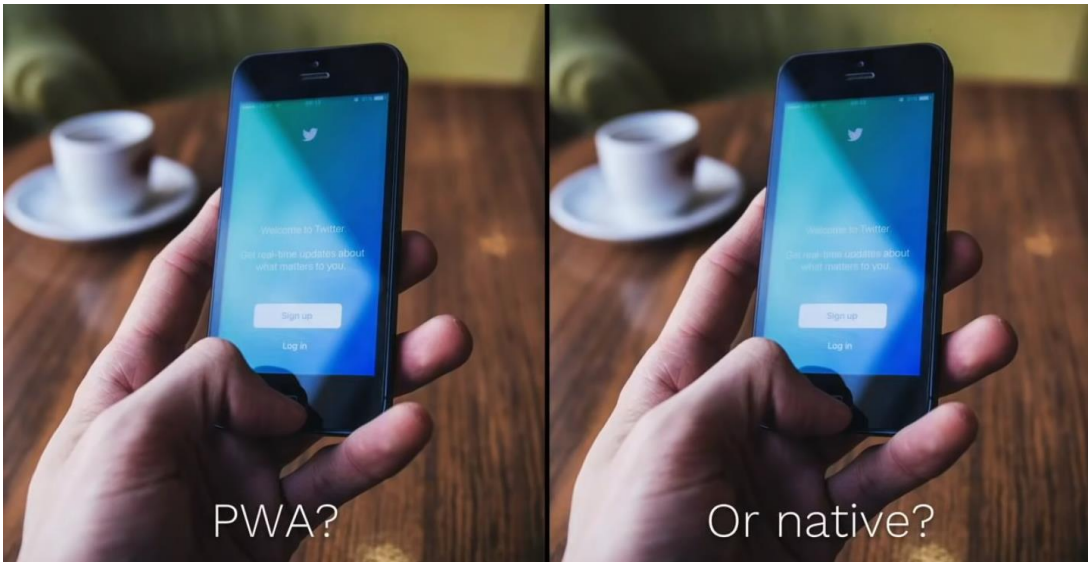
Opportunity

Symptoms

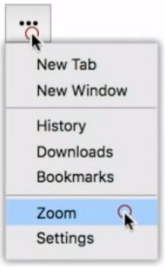
Problems

Causes

Poor supply chain for UI parts prevents specialization




Selecting a menu item with **two clicks** may feel easier:

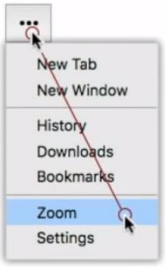



Mouse down/up (click) on menu button

Move over menu

Mouse down/up (click) on menu item

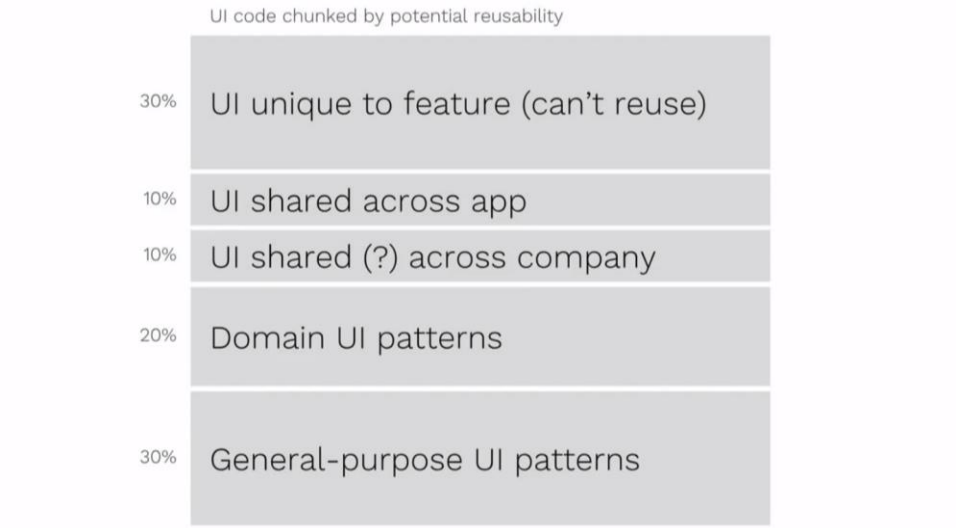
Selecting a menu item in a single **drag operation** may feel faster:




Mouse down on menu button

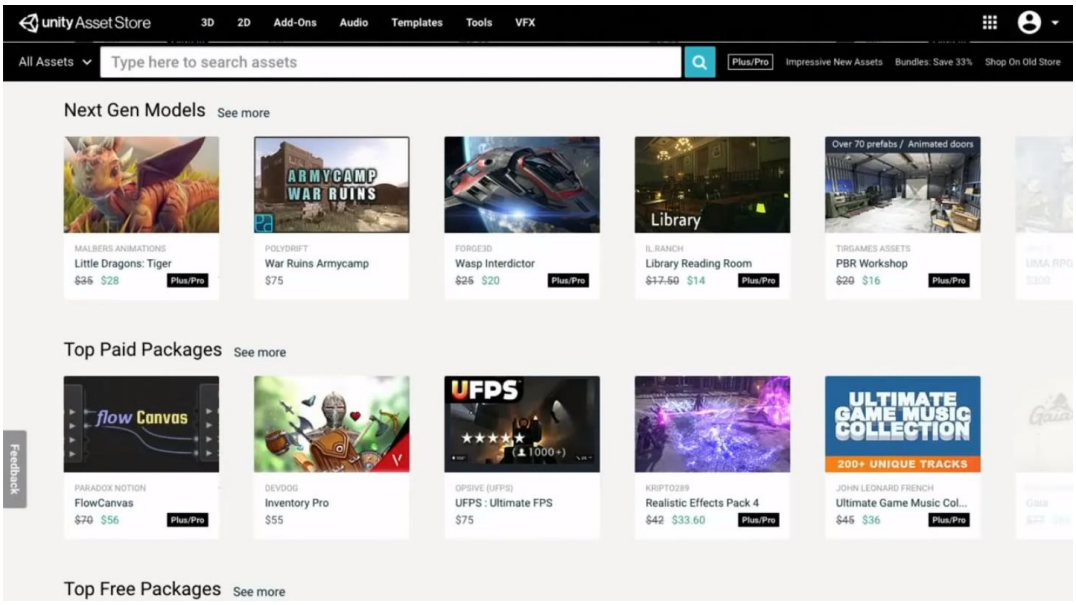
Drag into menu

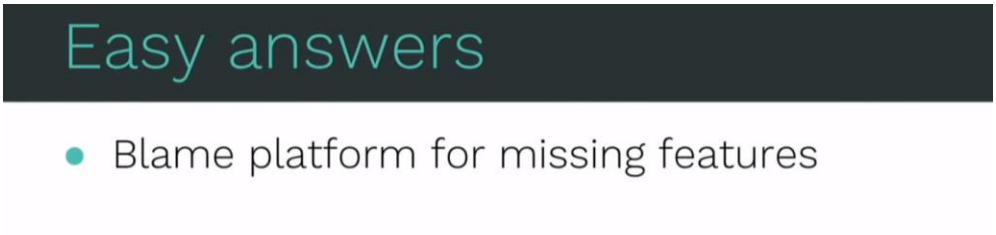
Mouse up over menu item



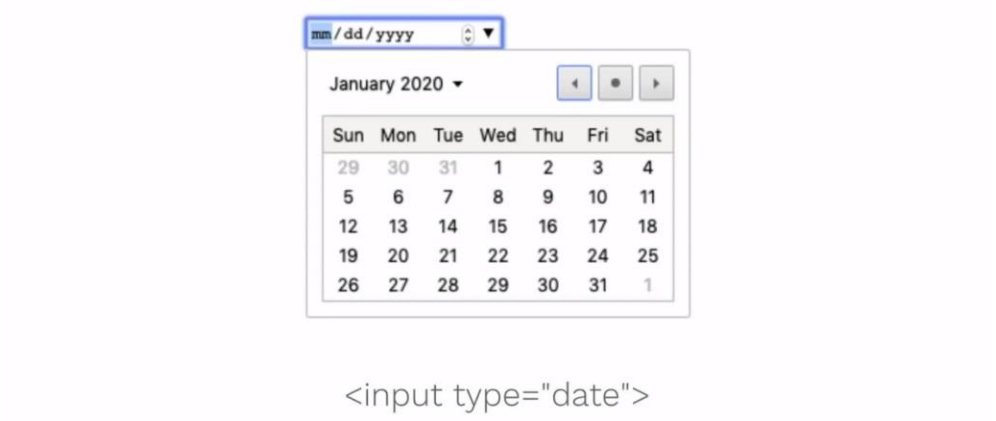

If we have a real UI supply chain that allows us to plug in reusable UI parts into new apps we are building, we can reduce the cost of development and not have to rebuild all things every time along with the frames.


We need a marketplace/way to reuse UI elements within our application

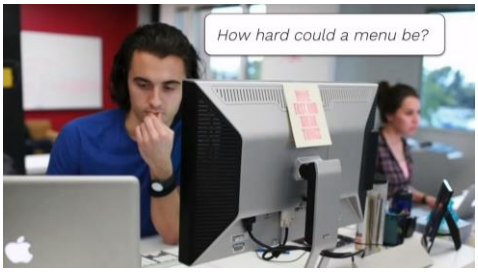


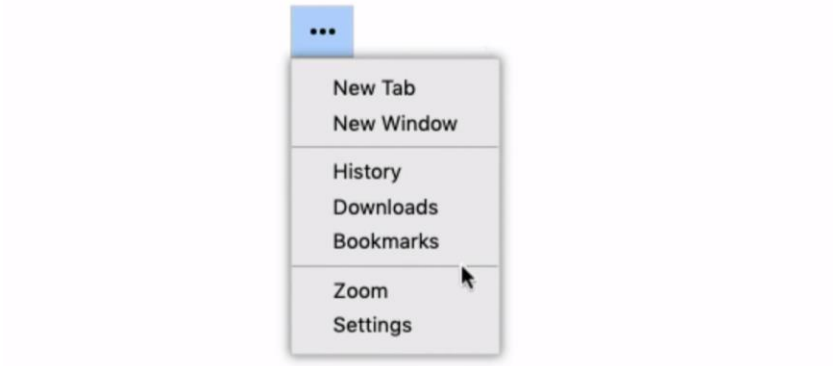

Most people will blame the browser, the HTML or CSS standard for the web.




This is what we get in Chrome, which does not look modern and things like copy & paste don’t work and it has 2 different looking dropdown looks.

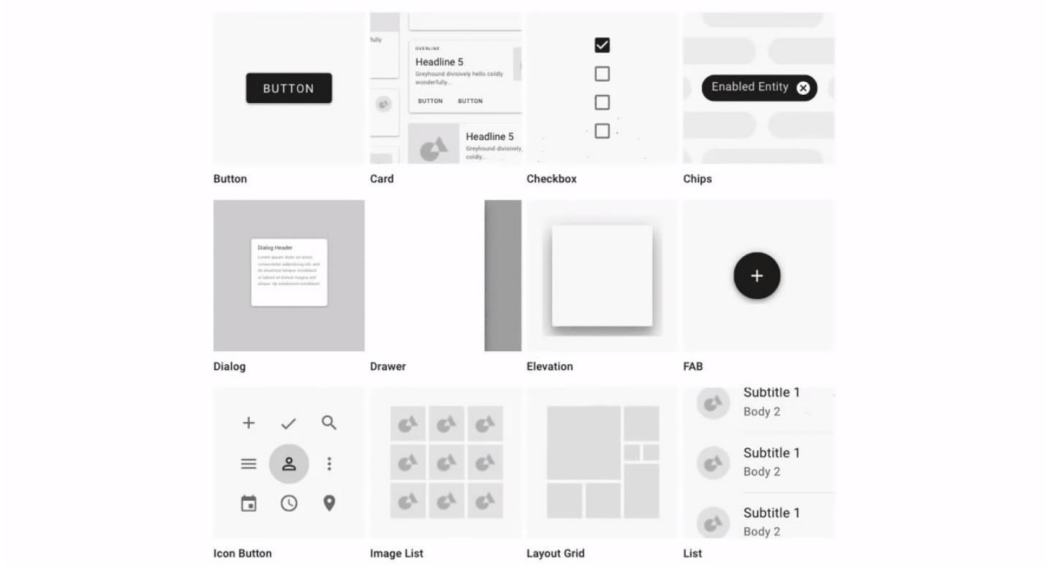




You still see the same complexity at every zoom level, same as UI challenging work.

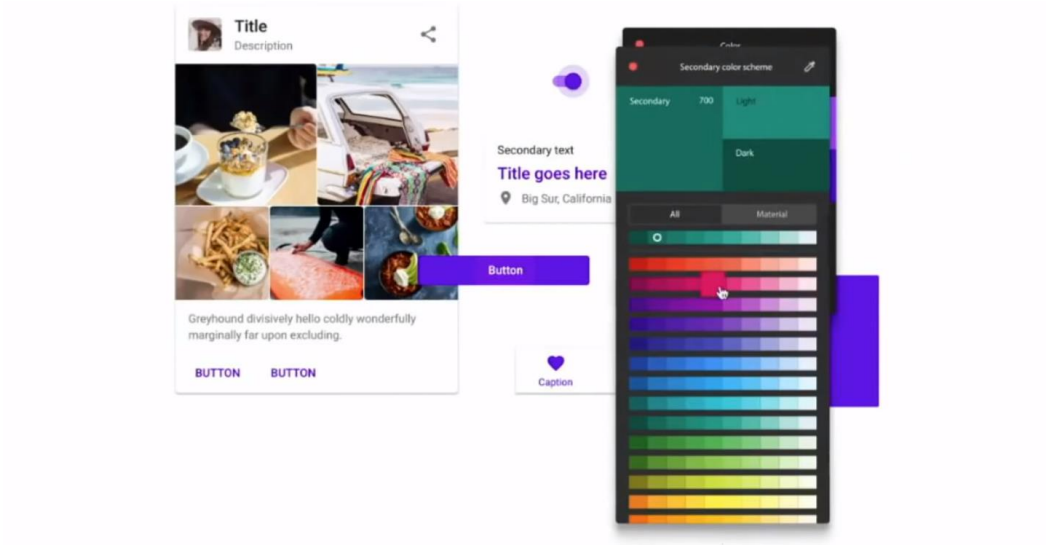


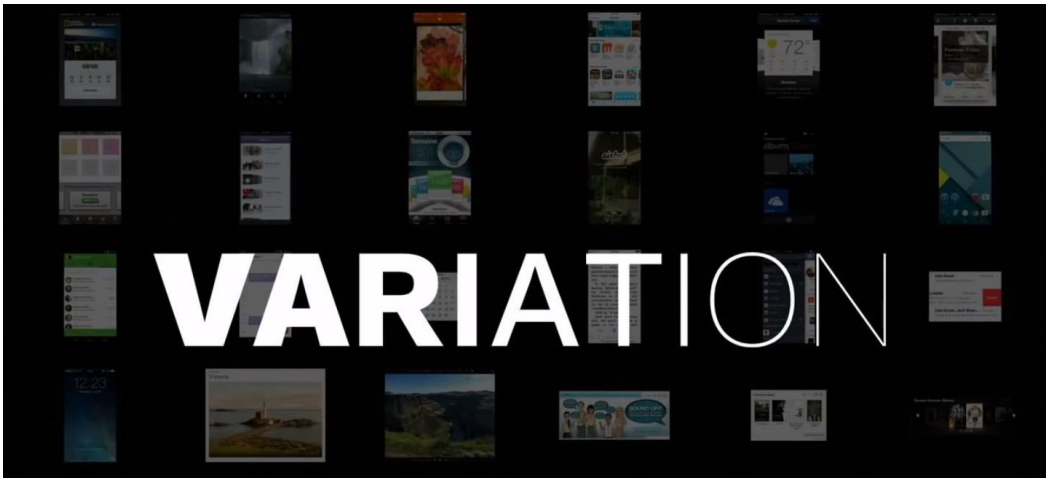

What should happen when a user tries to select something that is not selectable like the menu separator?

UI requires cross-discipline collaboration


Some companies do provide their design language as OSS but this is NOT open governance.






## Variation is fundamental

- Aesthetic
- Accidental
- Historical
- Cultural
- Situational
- Functional
- Gratuitous
- Differentiable

## Variation ≠ Styling

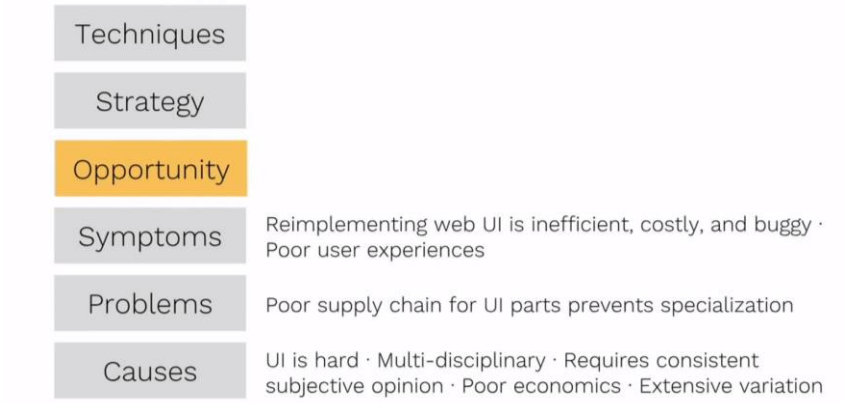
E.g., different ways a popup might be closeable:

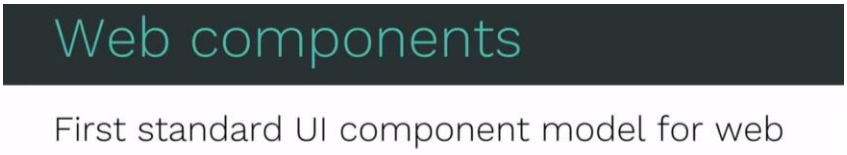
- Click outside it
- Click inside it
- Make a selection
- Click a close button
- Hover over another element (riffing)
- Press Esc
- Press ALT
- Wait
- Scroll the window
- Resize the window
- Tab to another window

Most UI libraries try to accommodate variation by adding knobs

```
<super-widget
  show-save-button="false"
  toolbar-background="gray"
  open-on-click="always"
  picker-label="Widget"
  dismiss-line="false"
  icon-style="reversed"
  shift="reduce"
  render-on="mobileOnly"
  underlined="true"
  ...>
</super-widget>
```

Most UI designers use multiple variations of doing the same thing in their component like above by using some configuration knob settings




There is a new web standard called Web Components API that we can use.

```
export class MyElement extends HTMLElement {
 constructor() {
   super();
   this.attachShadow({ mode: 'open' });
   this.shadowRoot.innerHTML = `
     <style>
       span { color: red; }
     </style>
     <span>Hello, world!</span>
   `;
 }
}

customElements.define('my-element', MyElement);
```

You can write JS that defines a class and registers that class with the browser as a custom HTML tag, the browser then runs your code when it sees your element.

```
export class MyElement extends HTMLElement {
 constructor() {
   super();
   this.attachShadow({ mode: 'open' });
   this.shadowRoot.innerHTML = `
     <style>
       span { color: red; }
     </style>
     <span>Hello, world!</span>
   `;
 }
}

customElements.define('my-element', MyElement);
```

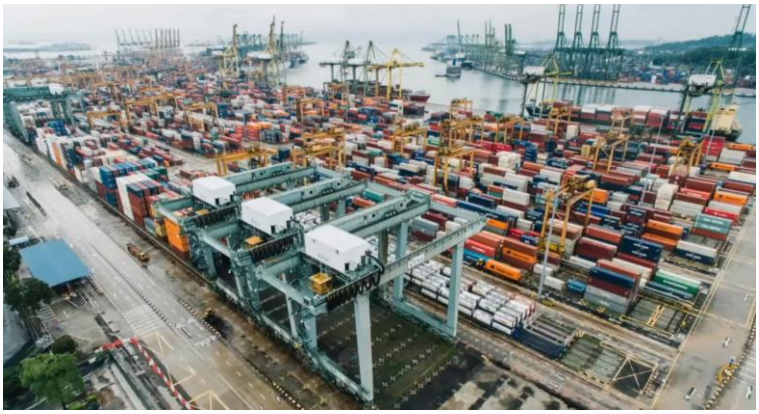
```
<html>
  <head>
    <style>
      span {
        color: blue !important;
      }
    </style>
  </head>
  <body>
    <my-element>
      #shadow-root
        <style>
          span { color: red; }
        </style>
        <span>Hello, world!</span>
    </my-element>
  </body>
</html>
```

Hello, world!

The shadow Dom is a private little tree inside your custom element that allows you to display your custom UI managed by your custom element.

## Web components assessment

- Fine for organizing your own app UI code
- Framework-independent
- Some rough edges
- Styling story is incomplete
- Barely good enough for reuse across orgs
- Ripple effects will be significant

## API existence is not enough

Still substantial work to make elements:

- Well-designed
- Reliable
- Capable of handling variation

The Web Components API in the browser does not guarantee that the UI components are good and resilient. Static HTML is just the initial state of the DOM, JS then comes in to add interactivity and things like images.


Frameworks are power tools

You still can't make a modern window by yourself

But you are still building everything by hand, frameworks cannot solve all the things for us

| | |
|---|---|
| Techniques | |
| Strategy | |
| Opportunity | Web components define a standard API for exchanging user interface elements |
| Symptoms | Reimplementing web UI is inefficient, costly, and buggy · Poor user experiences |
| Problems | Poor supply chain for UI parts prevents specialization |
| Causes | UI is hard · Multi-disciplinary · Requires consistent subjective opinion · Poor economics · Extensive variation |

## Composition

Instead of big complex components:

- Identify separable UI primitives
- Compose those to build elements

Composition might be a more interesting way to build complex components than trying to build monolithic components with a lot of configuration options.

## LISP computing primitives

| | |
|---|---|
| **defun** | functions |
| **t nil** names | atomic symbols |
| **car cdr cons** | list operations |
| **cond eq atom** | conditionals |

- Dynamic list of items
- Single selection
- Touch swipes
- Trackpad swipes
- Keyboard navigation
- Presentation + transitions
- Accessibility via ARIA
- Arrow buttons
- Page dots
- Page numbers
- Thumbnails
- Focus aggregation
- Language direction

Presentation and transitions


Modal stage Crossfade stage Sliding stage

Component UI pipeline

Events → Semantic API → State → Render

- Items management
- Selection model
- Touch swipes
- Trackpad swipes
- Keyboard navigation
- Presentation + transitions
- Accessibility via ARIA
- Arrow buttons
- Page dots
- Page numbers
- Thumbnails
- Focus aggregation
- Language direction
- DOM attributes

Most UI component libraries will have 20 to 30 components in them.

Accordion
AlertDialog
Alphabetic indices
Async operation button
AutoCompleteInput
AutoCompleteComboBox
Auto-format
AutoSizeTextarea
Backdrop
Breadcrumb bar
Button
CalendarDay
CalendarDays
CalendarDayNamesHeader
CalendarMonth
CalendarMonthNavigator
Calendar months
CalendarMonthYearHeader
Calendar year
Carousel
CarouselSlideshow
CarouselWithThumbnails
CenteredStrip
CenteredStripHighlight
CenteredStripOpacity
Checked list box
Close box
Closable panel
Color wheel
Combed text box
ComboBox
Content with banners
Content with sidebars
CrossfadeStage
Current anchor
DateComboBox
Date range calendar
Date text box
Delimited list
Dialog
Drawer
DropdownList
Editable in place
Editable text
ExpandablePanel
Expandable summary
Explorer
Fade overflow
File uploader
FilterComboBox
FilterListBox
Full screen expandable
Full screen
Gesture navigation
HamburgerMenuButton
Infinite list
Labeled input
Lazy list
Link list
ListBox
ListExplorer
ListComboBox
Marquee selection
ListWithSearch
ModalBackdrop
Masked text
Menu bar
Menu
MenuButton
MenuItem
MenuSeparator
Mobile date/time picker
ModalBackdrop
Modes
Multi list box
Multiple file uploader
Number with units
Overlay
OverlayFrame
Packed columns
PageDot
Page number navigator
Page transitions
Palette window
Panel with overflow menu
Persistent header
Persistent navigation bar
Persistent panel
Popout
Popup
PopupSource
Process steps
Progress bar
ProgressSpinner
PullToRefresh
Radio button list
Range slider
Ribbon
Rich text editor
Scrolled anchors
Scroll-up bar
SeamlessButton
Seamless iframe
Search box
Shimmer
Slider
Slideshow
SlideshowWithPlayControls
SlidingPages
SlidingStage
Spin box
Splitter
Stacked navigation pages
Star rating
TabButton
Tabs
TabStrip
Tag text box
Text box with button
Time combo box / time picker
Timeline
Toggle button
ToolTip
Toast
Tree view
Validating text input
Vote up/down
Wizard

These are some of the common available UI patterns

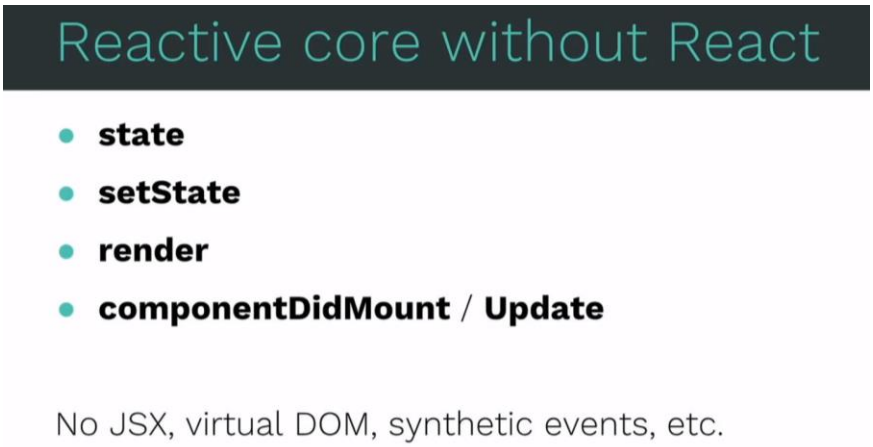
## ELIX

- Open source web component library
- Reusable UI primitives
- Comprehensive collection of all general-purpose UI patterns
- Support extensive variation
- Quality goal: as good as HTML

## Fundamentals

- Cross-browser
- Accessible
- All input modes
- International
- Resilient
- Performant
- Styleable
- Well-tested






We have an immutable **state** that you can call **setState** on that will then trigger a **render**, but we don't need the rest of the React component model like we can use browser events instead of React events.

```
class SpinBox extends ReactiveMixin(HTMLElement) {

  get defaultState() {
    return { value: 0 };
  }

  decrement() {
    this.setState({ value: this.state.value - 1 });
  }
  increment() {
    this.setState({ value: this.state.value + 1 });
  }

  render() {
    if (!this.shadowRoot) {
      this.attachShadow({ mode: 'open' });
      this.shadowRoot.innerHTML = `
        <span id="value"></span>
        <button id="upButton">▲</button>
        <button id="downButton">▼</button>
      `;
    }
    const value = this.shadowRoot.getElementById('value');
    value.textContent = this.state.value;
  }

}
```

0

```
<!-- Initial render -->
<spin-box>
  #shadow-root
    <span id="value">0</span>
    <button id="upButton">▲</button>
    <button id="downButton">▼</button>
</spin-box>
```

1

```
<!-- After increment -->
<spin-box>
  #shadow-root
    <span id="value">1</span>
    <button id="upButton">▲</button>
    <button id="downButton">▼</button>
</spin-box>
```

## Functional mixins

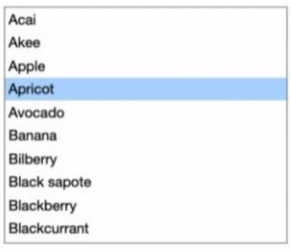
- Compose behavior in plain JavaScript without a framework
- A mixin is just a function that takes a class and returns a new subclass

```
// Functional mixin
function GreetMixin(Base) {
  return class Greet extends Base {
    greet() {
      alert('Hello');
    }
  };
}

// Define a web component that uses the mixin.
class GreetElement extends GreetMixin(HTMLElement) {}
customElements.define('greet-element', GreetElement);

// Instantiate the component.
const element = new GreetElement();
element.greet(); // "Hello"
```

This is an additive mixin that acts like a normal HTMLElement and also now knows how to say hello. This pattern is sufficient for our reactive features and the UI primitives that we want.




If creating UI elements like a list box, we can identify the UI elements and build them as mixins

```
class ListBox extends HTMLElement {
  constructor() {
    super();
    this.selectedItem = null;
    this.addEventListener('click', event => {
      if (this.selectedItem) {
        // Deselect previous item.
        this.selectedItem.style.backgroundColor = null;
        this.selectedItem.style.color = null;
        this.selectedItem.setAttribute('aria-selected', false);
      }
      // Select new item.
      this.selectedItem = event.target;
      this.selectedItem.style.backgroundColor = 'highlight';
      this.selectedItem.style.color = 'highlighttext';
      this.selectedItem.setAttribute('aria-selected', true);
      this.setAttribute('aria-activedescendant', this.selectedItem.id);
    });
  }
}
```

This is a monolithic web component approach that works but does not permit variations in the behavior of the list box

Goal: a tap/click on a list item highlights it and applies appropriate ARIA attributes for accessibility

| Primitives: | Mixins: |
|---|---|
| Track items in the list | **SlotItemsMixin** |
| Track selection state | **SingleSelectionMixin** |
| Map taps/clicks to selection | **TapSelectionMixin** |
| Render visual highlight | (custom) |
| Render ARIA attributes | **AriaListMixin** |

We can map each of these highlights to functional mixins that we take care of each part

```
function TapSelectionMixin(Base) {
  return class TapSelection extends Base {
    constructor() {
      super();
      this.addEventListener('click', event => {
        if (event.target !== this) {
          this.selectedItem = event.target;
        }
      });
    }
  };
}
```

This mixin will give any base class a click handler, then call a selection API and passing it what was clicked

```
function SingleSelectionMixin(Base) {
  return class SingleSelection extends Base {

    get defaultState() {
      return Object.assign({}, super.defaultState, {
        selectedItem: null
      });
    }

    get selectedItem() {
      return this.state.selectedItem;
    }
    set selectedItem(item) {
      this.setState({ selectedItem: item });
    }

    selectFirst() { ... }
    selectLast() { ... }
    selectNext() { ... }
    selectPrevious() { ... }

  };
}
```

This mixin handles the selection API and simply remembers what was clicked and exposes some methods for navigating a selection. This is a semantic operation that will update the state of the list box

```
function AriaListMixin(Base) {
  return class AriaList extends Base {
    render() {
      if (super.render) { super.render(); }
      const { items, selectedItem } = this.state;
      items.forEach(item => {
        const isSelected = item === selectedItem;
        item.setAttribute('aria-selected', isSelected);
      });
      const selectedItemId = this.selectedItem ? this.selectedItem.id : null;
      this.setAttribute('aria-activedescendant', selectedItemId);
    }
  };
}
```

We can then render the new state of the lost box using the mixin above, this will handle the Aria attributes on both the individual list items and on the top-level list itself

```
const Base =
  SlotItemsMixin(
  SingleSelectionMixin(
  TapSelectionMixin(
  AriaListMixin(
  ReactiveMixin(
    HTMLElement
  )))));

class ListBox extends Base {
  render() {
    if (super.render) { super.render(); }
    const { items, selectedItem } = this.state;
    items.forEach(item => {
      const isSelected = item === selectedItem;
      item.style.backgroundColor = isSelected ? 'highlight' : null;
      item.style.color = isSelected ? 'highlighttext' : null;
    });
  }
}
```

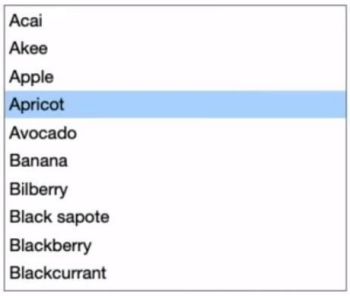
We can put all the mixins together by applying a bunch of functions on the HTMLElement by invoking all the mixin functions as above. This gives us a base/foundation for a list.

```
const ListBoxBase =
  AriaListMixin(
  AttributeMarshallingMixin(
  ComposedFocusMixin(
  ContentItemsMixin(
  DirectionSelectionMixin(
  FocusVisibleMixin(
  ItemsTextMixin(
  KeyboardDirectionMixin(
  KeyboardMixin(
  KeyboardPagedSelectionMixin(
  KeyboardPrefixSelectionMixin(
  LanguageDirectionMixin(
  ReactiveMixin(
  RenderUpdatesMixin(
  SelectedItemTextValueMixin(
  SelectionInViewMixin(
  ShadowTemplateMixin(
  SingleSelectionMixin(
  SlotContentMixin(
  TapSelectionMixin(
    HTMLElement
  ))))))))))))))))))));
```

This is what we have in the Elix project, a list box with about 20 mixins for UI primitives variations we might need like auto-select, up and down keys, etc.


```
const ListBoxBase =
  AriaListMixin(
  AttributeMarshallingMixin(
  ComposedFocusMixin(
  ContentItemsMixin(
  DirectionSelectionMixin(
  FocusVisibleMixin(
  ItemsTextMixin(
  KeyboardDirectionMixin(
  KeyboardMixin(
  KeyboardPagedSelectionMixin(
  KeyboardPrefixSelectionMixin(
  LanguageDirectionMixin(
  ReactiveMixin(
  RenderUpdatesMixin(
  SelectedItemTextValueMixin(
  SelectionInViewMixin(
  ShadowTemplateMixin(
  SingleSelectionMixin(
  SlotContentMixin(
  TapSelectionMixin(
    HTMLElement
  ))))))))))))))))))));
```

```
const MenuBase =
  AriaMenuMixin(
  AttributeMarshallingMixin(
  ContentItemsMixin(
  DelegateFocusMixin(
  DirectionSelectionMixin(
  FocusVisibleMixin(
  ItemsTextMixin(
  KeyboardDirectionMixin(
  KeyboardMixin(
  KeyboardPagedSelectionMixin(
  KeyboardPrefixSelectionMixin(
  LanguageDirectionMixin(
  ReactiveMixin(
  RenderUpdatesMixin(
  SelectedItemTextValueMixin(
  SelectionInViewMixin(
  ShadowTemplateMixin(
  SingleSelectionMixin(
  SlotContentMixin(
  TapSelectionMixin(
    HTMLElement
  ))))))))))))))))))));
```




They are not similar in look but they have similar behaviors that allows code to be shared, about 85%.

```
class SpinBox extends ReactiveMixin(HTMLElement) {
  get defaultState() {
    return { value: 0 };
  }
  decrement() {
    this.setState({ value: this.state.value - 1 });
  }
  increment() {
    this.setState({ value: this.state.value + 1 });
  }
  render() {
    if (!this.shadowRoot) {
      this.attachShadow({ mode: 'open' });
      this.shadowRoot.innerHTML = `
        <span id="value"></span>
        <button id="up">▲</button>
        <button id="down">▼</button>
      `;
    }
    const value = this.shadowRoot.getElementById('value');
    value.textContent = this.state.value;
  }
}
```

```
class SpinBox extends ReactiveMixin(ShadowTemplateMixin(HTMLElement)) {
  get defaultState() {
    return { value: 0 };
  }
  decrement() {
    this.setState({ value: this.state.value - 1 });
  }
  increment() {
    this.setState({ value: this.state.value + 1 });
  }
  get template() {
    return html`
      <span id="value"></span>
      <button id="up">▲</button>
      <button id="down">▼</button>
    `;
  }
  render() {
    if (super.render) { super.render(); }
    const value = this.shadowRoot.getElementById('value');
    value.textContent = this.state.value;
  }
}
```

```
class SpinBox extends ReactiveElement {
  get defaultState() {
    return { value: 0 };
  }
  decrement() {
    this.setState({ value: this.state.value - 1 });
  }
  increment() {
    this.setState({ value: this.state.value + 1 });
  }
  get template() {
    return html`
      <span id="value"></span>
      <button id="up">▲</button>
      <button id="down">▼</button>
    `;
  }
  render() {
    if (super.render) { super.render(); }
    const value = this.shadowRoot.getElementById('value');
    value.textContent = this.state.value;
  }
}
```

As a convenience, we then bundle these common components into a base class called ReactiveElement that is inherited from by other components.

## Overridable DOM updates

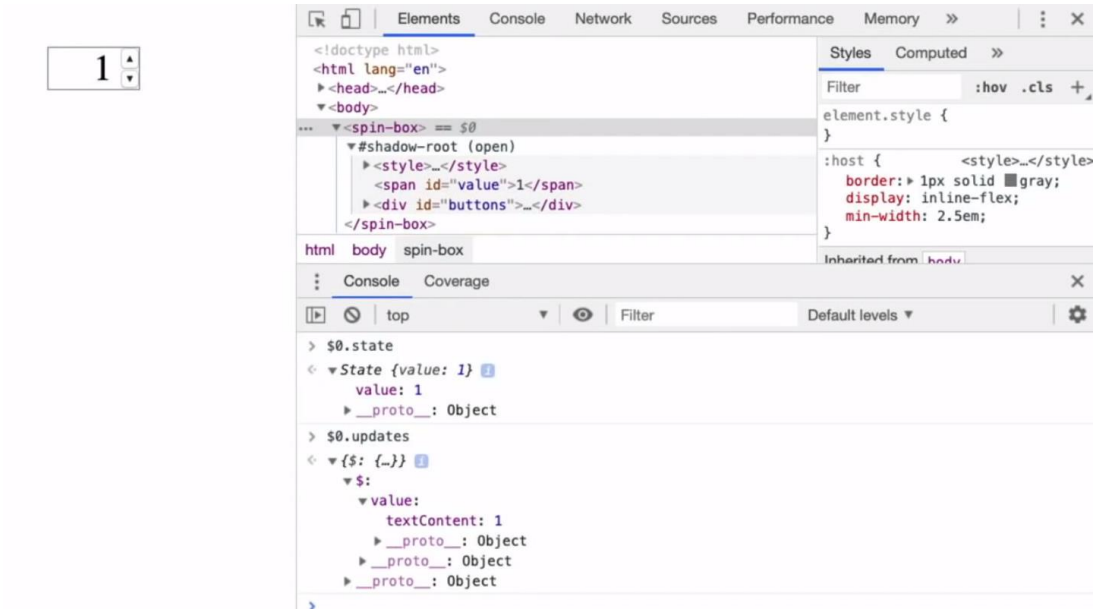
```
this.setAttribute('id', 'save');
this.style.color = 'red';
this.textContent = 'OK';
```

→

```
{
  attributes: {
    id: 'save'
  },
  style: {
    color: 'red'
  },
  textContent: 'OK'
}
```

```
class SpinBox extends ReactiveElement {
  ...
  get template() {
    return html`
      <span id="value"></span>
      <button id="up">▲</button>
      <button id="down">▼</button>
    `;
  }
  render() {
    if (super.render) { super.render(); }
    this.$.value.textContent = this.state.value;
  }
}
```

```
class SpinBox extends ReactiveElement {
  ...
  get template() {
    return html`
      <span id="value"></span>
      <button id="up">▲</button>
      <button id="down">▼</button>
    `;
  }
  get updates() {
    return {
      $: {
        value: {
          textContent: this.state.value
        }
      }
    };
  }
}
```




```
class SpinBox extends ReactiveElement {
  ...
  get updates() {
    return {
      $: {
        value: {
          textContent: this.state.value
        }
      }
    };
  }
}

class CustomSpinBox extends SpinBox {
  get updates() {
    return merge(super.updates, {
      $: {
        value: {
          style: {
            color: this.state.value < 0 ? 'red' : null
          }
        }
      }
    });
  }
}
```

-1

-1

```
class SpinBox extends ReactiveElement {
  ...
  get updates() {
    return {
      $: {
        value: {
          textContent: this.state.value
        }
      }
    };
  }
}

class RomanSpinBox extends SpinBox {
  get updates() {
    return merge(super.updates, {
      $: {
        value: {
          textContent: romanize(this.state.value)
        }
      }
    });
  }
}
```

40

XL

```
class SpinBox extends ReactiveElement {
  get updates() {
    return {
      $: {
        value: {
          textContent: this.state.value
        }
      }
    };
  }
}
class CustomSpinBox extends SpinBox {
  get updates() {
    return merge(super.updates, {
      $: {
        value: {
          style: {
            color: 'red'
          }
        }
      }
    });
  }
}
```

0

0

## Template patching

- Ask base class for a copy of its template
- Patch it however we want
- Return the patched template

```
class SpinBox extends ReactiveElement {
  get template() {
    return html`
      <!-- Defined by SpinBox -->
      <span id="value"></span>
      <button id="up">▲</button>
      <button id="down">▼</button>
    `;
  }
}
```

```
<spin-box>
  #shadow-root
    <!-- Defined by SpinBox -->
    <span id="value">0</span>
    <button id="up">▲</button>
    <button id="down">▼</button>
</spin-box>
```

0

```
class CustomSpinBox extends SpinBox {
  get template() {
    return concat(
      super.template,
      html`
        <!-- Defined by CustomSpinBox -->
        <style>
          #value {
            color: red;
          }
        </style>
      `
    );
  }
}
```

```
<custom-spin-box>
  #shadow-root
    <!-- Defined by SpinBox -->
    <span id="value">0</span>
    <button id="up">▲</button>
    <button id="down">▼</button>
    <!-- Defined by CustomSpinBox -->
    <style>
      #value {
        color: red;
      }
    </style>
</custom-spin-box>
```

0


These variations will be really hard to do with a single carousel to rule them all, it is a lot easier to have the building component blocks to put it together by adding behaviors as needed.


Most apps already have elements in their design language, we can use this approach to make them better as styling and behaviors.

```
class SpinBox extends ReactiveElement {
  get template() {
    return html`
      <!-- Defined by SpinBox -->
      <span id="value"></span>
      <button id="up">▲</button>
      <button id="down">▼</button>
    `;
  }
}

class CustomSpinBox extends SpinBox {
  get template() {
    return concat(
      super.template,
      html`
        <!-- Defined by CustomSpinBox -->
        <style>
          button {
            background: darkgray;
            color: white;
          }
        </style>
      `
    );
  }
}
```

```
<spin-box>
  #shadow-root
    <!-- Defined by SpinBox -->
    <span id="value">0</span>
    <button id="up">▲</button>
    <button id="down">▼</button>
</spin-box>

<custom-spin-box>
  #shadow-root
    <!-- Defined by SpinBox -->
    <span id="value">0</span>
    <button id="up">▲</button>
    <button id="down">▼</button>
    <!-- Defined by CustomSpinBox -->
    <style>
      button {
        background: darkgray;
        color: white;
      }
    </style>
</custom-spin-box>
```

```
<!-- Spin box uses regular buttons -->
<spin-box>
  #shadow-root
    <span id="value">0</span>
    <button id="up">▲</button>
    <button id="down">▼</button>
</spin-box>

<!-- Spin box variation using custom buttons -->
<spin-box>
  #shadow-root
    <span id="value">0</span>
    <custom-button id="up">▲</custom-button>
    <custom-button id="down">▼</custom-button>
</spin-box>
```

But Web Components are really a way of packaging up styling and behaviors, we can swap out the DOM UI elements (like a button) we want to enhance with the web component version.

# Replaceable element roles

- Identify a key node in the template
- Role defines the type used at that node
- Fill role with some other element class


We want to take an element and hand it to our custom element as a parameter to swap

```
export class SpinBox extends ReactiveElement {
  ...
  get defaultState() {
    return {
      buttonRole: 'button',
      value: 0
    };
  }
  get template() {
    return html`
      <span id="value"></span>
      <button id="up">▲</button>
      <button id="down">▼</button>
    `;
  }
  componentDidUpdate(previousState) {
    if (this.state.buttonRole !== previousState.buttonRole) {
      transmute(this.$.up, this.state.buttonRole);
      transmute(this.$.down, this.state.buttonRole);
    }
  }
}
```

We use the transmute() operation to do the swapping

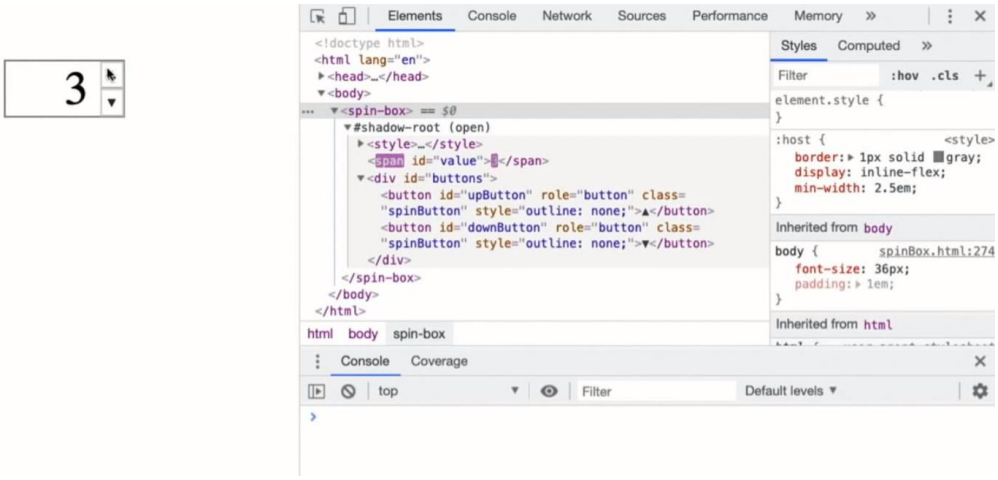
```
export class SpinBox extends ReactiveElement {
  ...
  get defaultState() {
    return {
      buttonRole: 'button',
      value: 0
    };
  }
  get template() {
    return html`
      <span id="value"></span>
      <button id="up">▲</button>
      <button id="down">▼</button>
    `;
  }
  componentDidUpdate(previousState) {
    if (this.state.buttonRole !== previousState.buttonRole) {
      transmute(this.$.up, this.state.buttonRole);
      transmute(this.$.down, this.state.buttonRole);
    }
  }
}
```

```
<spin-box>
  #shadow-root
    <span id="value">0</span>
    <button id="up">▲</button>
    <button id="down">▼</button>
</spin-box>
```

0

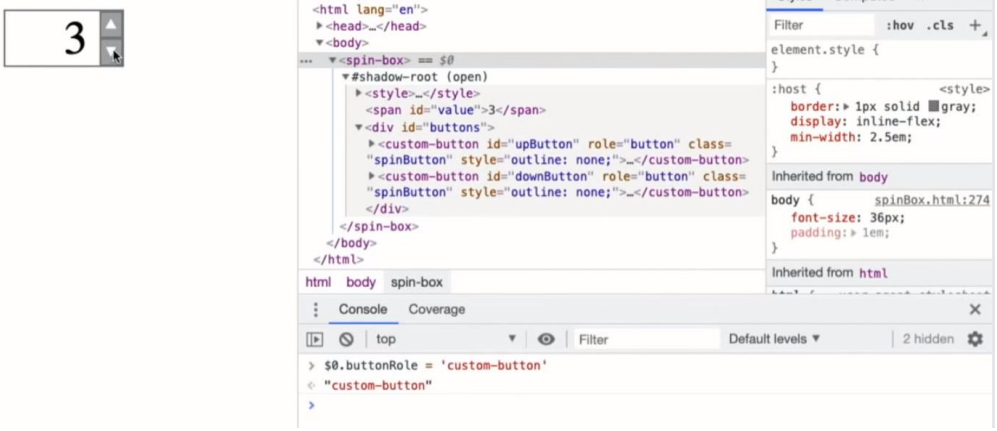
```
<spin-box
    button-role="custom-button">
  #shadow-root
    <span id="value">0</span>
    <custom-button id="up">▲</custom-button>
    <custom-button id="down">▼</custom-button>
</spin-box>
```

0


3
Elements Console Network Sources Performance Memory
<!doctype html>
<html lang="en">
<head>…</head>
<body>
<spin-box> == $0
#shadow-root (open)
<style>…</style>
3

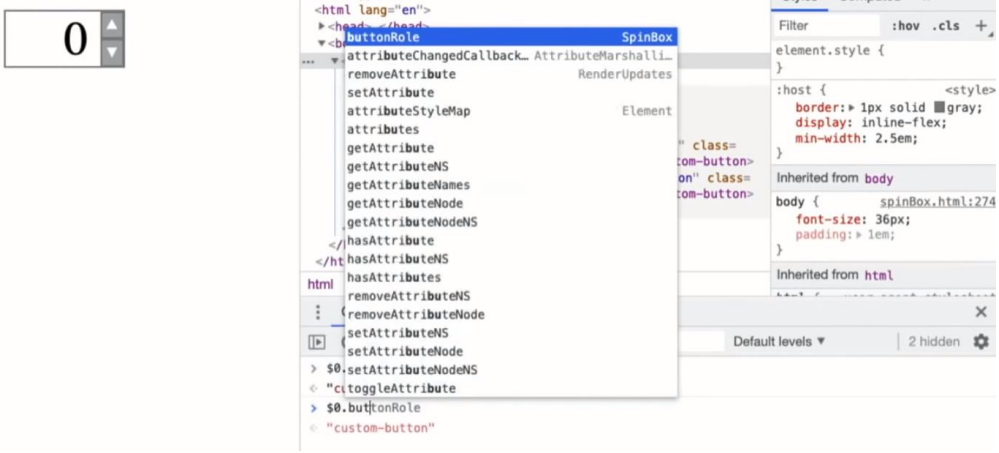
<button id="upButton" role="button" class="spinButton" style="outline: none;">▲</button>
<button id="downButton" role="button" class="spinButton" style="outline: none;">▼</button>

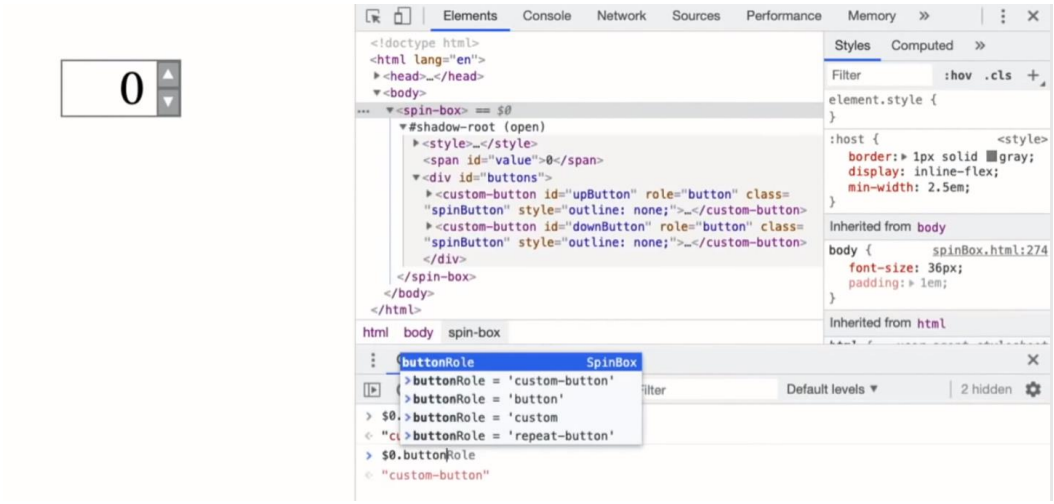
</spin-box>
</body>
</html>
html body spin-box
Styles Computed
Filter :hov .cls
element.style {
}
:host { <style>
border: 1px solid gray;
display: inline-flex;
min-width: 2.5em;
}
Inherited from body
body { spinBox.html:274
font-size: 36px;
padding: 1em;
}
Inherited from html
Console Coverage
top Filter Default levels


3
Elements Console Network Sources Performance Memory
<!doctype html>
<html lang="en">
<head>…</head>
<body>
<spin-box> == $0
#shadow-root (open)
<style>…</style>
3

<custom-button id="upButton" role="button" class="spinButton" style="outline: none;">…</custom-button>
<custom-button id="downButton" role="button" class="spinButton" style="outline: none;">…</custom-button>

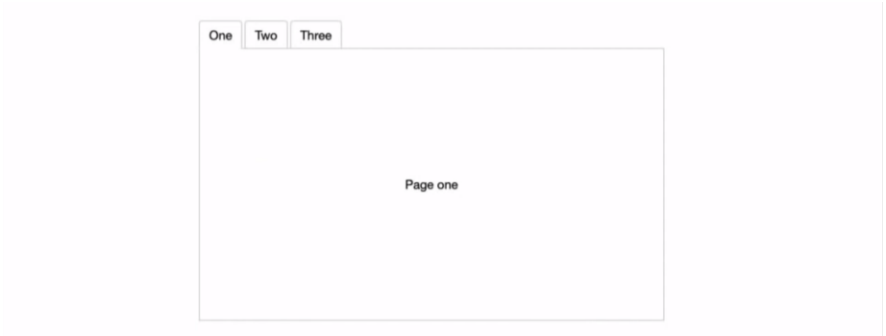
</spin-box>
</body>
</html>
html body spin-box
Styles Computed
Filter :hov .cls
element.style {
}
:host { <style>
border: 1px solid gray;
display: inline-flex;
min-width: 2.5em;
}
Inherited from body
body { spinBox.html:274
font-size: 36px;
padding: 1em;
}
Inherited from html
Console Coverage
top Filter Default levels 2 hidden
$0.buttonRole = 'custom-button'
"custom-button"

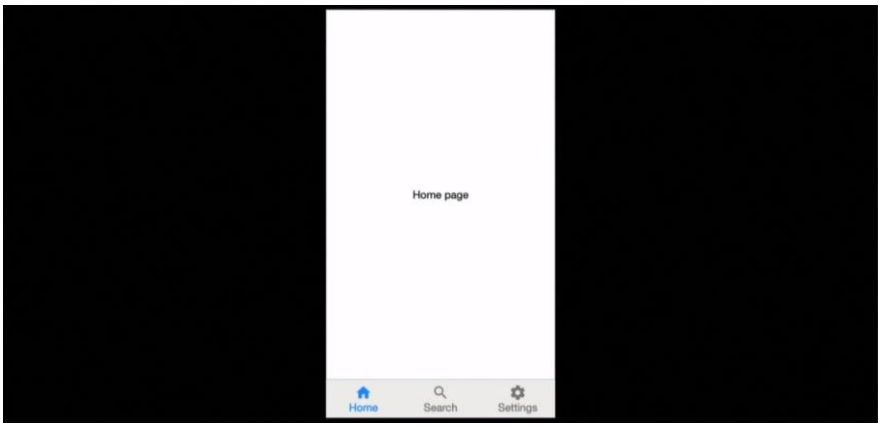

0
Elements Console Network Sources Performance Memory
buttonRole SpinBox
attributeChangedCallback… AttributeMarshalli…
removeAttribute RenderUpdates
setAttribute
attributeStyleMap Element
attributes
getAttribute
getAttributeNS
getAttributeNames
getAttributeNode
getAttributeNodeNS
hasAttribute
hasAttributeNS
hasAttributes
removeAttributeNS
removeAttributeNode
setAttributeNS
setAttributeNode
setAttributeNodeNS
toggleAttribute
Styles Computed
Filter :hov .cls
element.style {
}
:host { <style>
border: 1px solid gray;
display: inline-flex;
min-width: 2.5em;
}
Inherited from body
body { spinBox.html:274
font-size: 36px;
padding: 1em;
}
Inherited from html
Default levels 2 hidden
$0.buttonRole
"custom-button"

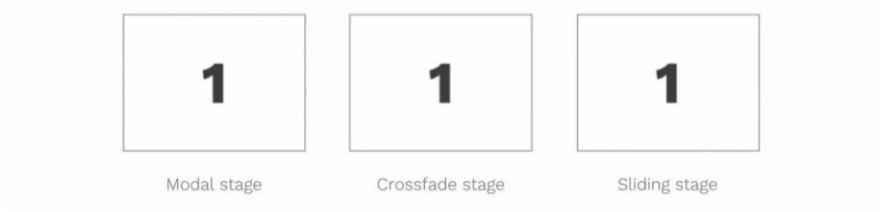

0
Elements
Console
Network
Sources
Performance
Memory
<!doctype html>
<html lang="en">
<head>…</head>
<body>
<spin-box> == $0
#shadow-root (open)
<style>…</style>
0

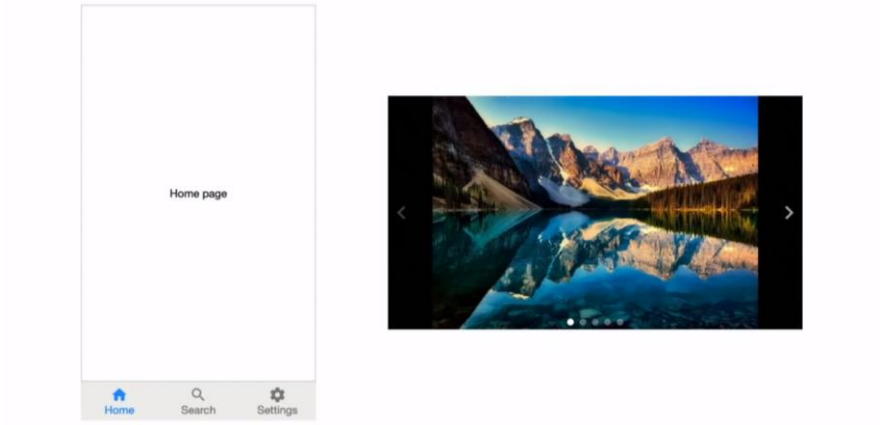
<custom-button id="upButton" role="button" class="spinButton" style="outline: none;">…</custom-button>
<custom-button id="downButton" role="button" class="spinButton" style="outline: none;">…</custom-button>

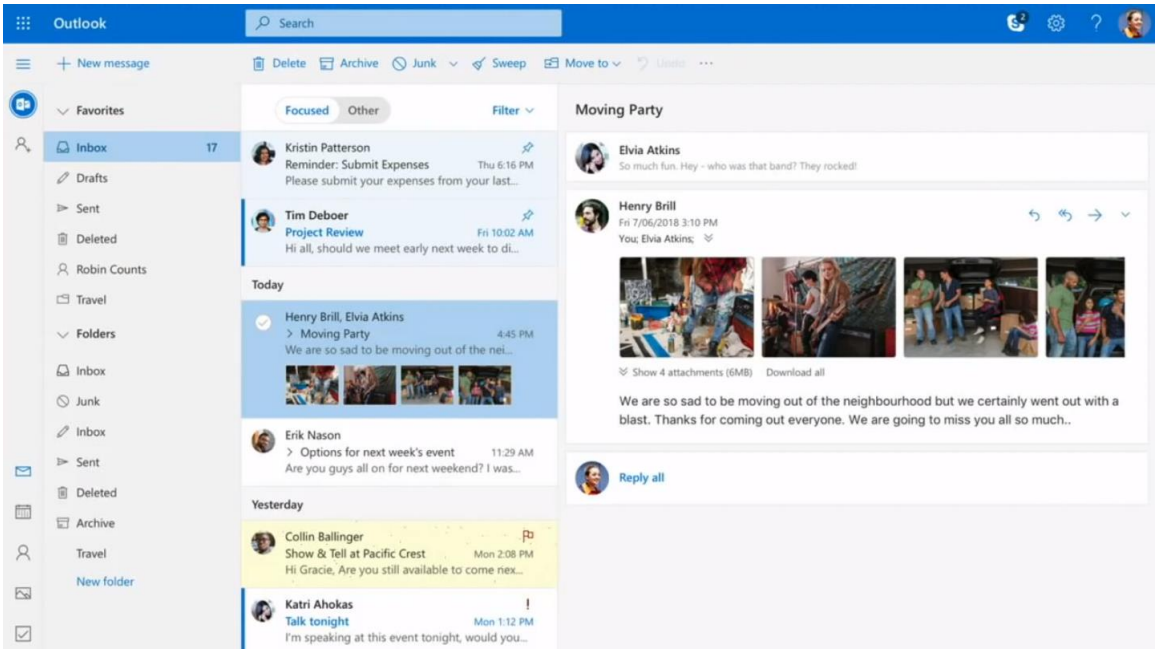
</spin-box>
</body>
</html>
html
body
spin-box
Styles
Computed
Filter
:hov .cls
element.style {
}
:host {
<style>
border: 1px solid gray;
display: inline-flex;
min-width: 2.5em;
}
Inherited from body
body {
spinBox.html:274
font-size: 36px;
padding: 1em;
}
Inherited from html
buttonRole
SpinBox
buttonRole = 'custom-button'
buttonRole = 'button'
buttonRole = 'custom
buttonRole = 'repeat-button'
Default levels
2 hidden
$0.buttonRole
"custom-button"

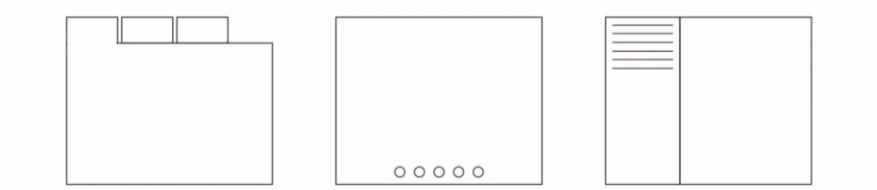

One
Two
Three
Page one


Home page
Home
Search
Settings

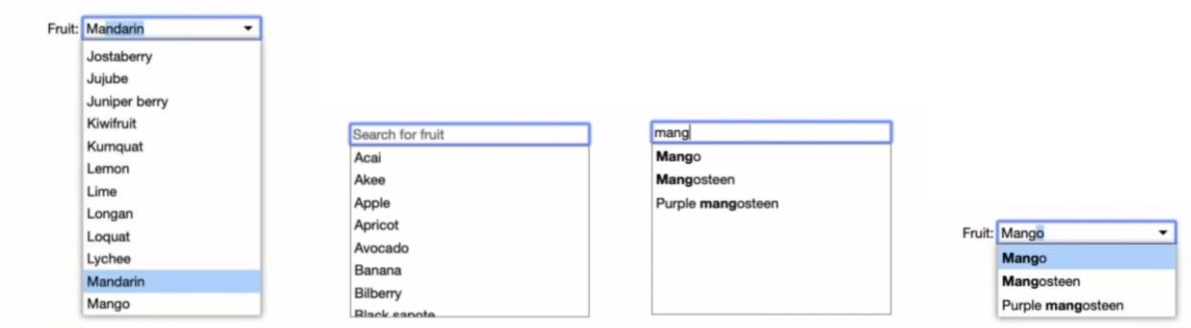

1
1
1
Modal stage
Crossfade stage
Sliding stage


Home page
Home
Search
Settings

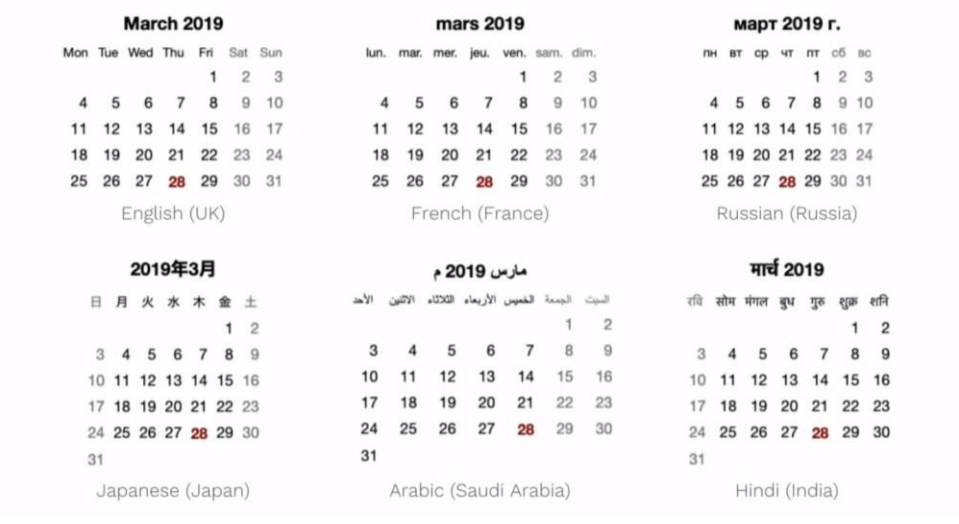

Outlook
Search
New message
Delete
Archive
Junk
Sweep
Move to
Favorites
Inbox 17
Drafts
Sent
Deleted
Robin Counts
Travel
Folders
Inbox
Junk
Inbox
Sent
Deleted
Archive
Travel
New folder
Focused
Other
Filter
Kristin Patterson
Reminder: Submit Expenses
Thu 6:16 PM
Please submit your expenses from your last...
Tim Deboer
Project Review
Fri 10:02 AM
Hi all, should we meet early next week to di...
Today
Henry Brill, Elvia Atkins
Moving Party
4:45 PM
We are so sad to be moving out of the nei...
Erik Nason
Options for next week's event
11:29 AM
Are you guys all on for next weekend? I was...
Yesterday
Collin Ballinger
Show & Tell at Pacific Crest
Mon 2:08 PM
Hi Gracie, Are you still available to come nex...
Katri Ahokas
Talk tonight
Mon 1:12 PM
I'm speaking at this event tonight, would you...
Moving Party
Elvia Atkins
So much fun. Hey - who was that band? They rocked!
Henry Brill
Fri 7/06/2018 3:10 PM
You; Elvia Atkins;
Show 4 attachments (6MB)
Download all
We are so sad to be moving out of the neighbourhood but we certainly went out with a blast. Thanks for coming out everyone. We are going to miss you all so much..
Reply all



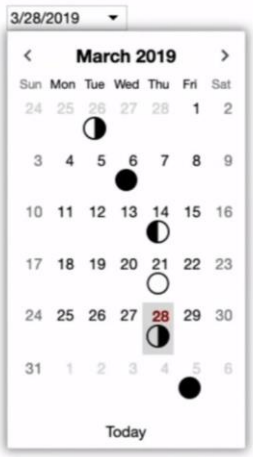
They are all the same thing in essence and behaviors













| | |
|---|---|
| Techniques | Reactive core · Functional mixins · Overridable DOM updates · Template patching · Replaceable element roles |
| Strategy | UI primitives · Comprehensive library of general-purpose UI patterns · As good as HTML |
| Opportunity | Web components define a standard API for exchanging user interface elements |
| Symptoms | Reimplementing web UI is inefficient, costly, and buggy · Poor user experiences |
| Problems | Poor supply chain for UI parts prevents specialization |
| Causes | UI is hard · Multi-disciplinary · Requires consistent subjective opinion · Poor economics · Extensive variation |

https://component.kitchen/elix

These slides: bit.ly/better-elements

jan@component.kitchen or @JanMiksovsky

rajeshsu@microsoft.com