Class 6: Introduction to Sprints

1. Review: Conditional Class Binding, Custom Events, Watchers
2. Overview: Scrum + Sprints
3. Review: Component Trees
4. Studio: Homework 2, Sprint 1

Review: Conditional Class Binding

```
<script setup>
defineProps({
  variant: {
    type: String,
    default: 'filled'
  }
})
</script>

<template>
  <div>
    <button v-for="num in 5" :key="num"
      :class="variant === 'filled' ? 'solid' : 'outline'">
      ☆</button>
  </div>
</template>
```

Review: Custom Events

```
<script setup>
const emit = defineEmits(['rate'])
</script>

<template>
  <div>
    <button v-for="num in 5" :key="num" @click="$emit('rate', num)">☆</button>
  </div>
</template>
```

```
<template>
  <StarRating v-on:rate="(num) => console.log(`Rated ${num} stars!`)"/>
</template>
```

Review: Watchers

Definition: A watcher is a function that observes changes to a reactive variable and executes a callback function when the variable changes.

```
<script setup>
import { ref, watch } from 'vue'
const rating = ref(0)

watch(rating, (newVal, oldVal) => {
  console.log(`Rated ${newVal} stars!`)
})
</script>

<template>
  <StarRating v-on:rate="(num) => {rating = num}"/>
</template>
```

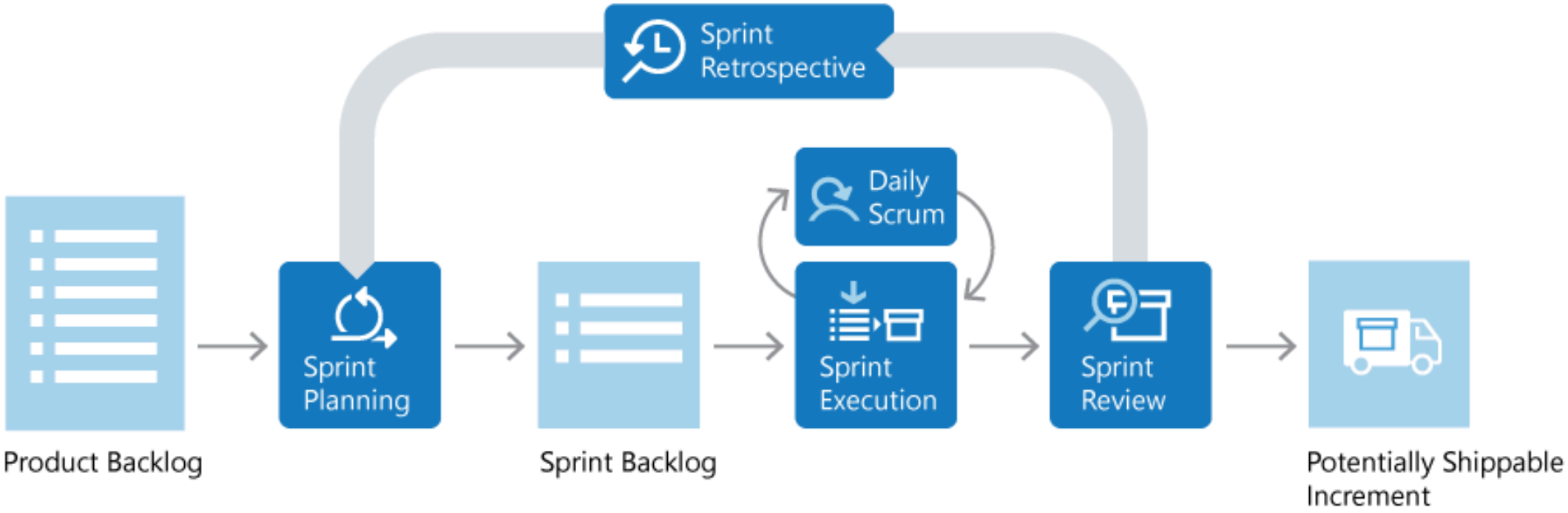
Overview: Scrum

Scrum

> Scrum is a framework used by teams to manage work and solve problems collaboratively in short cycles.

Source: Microsoft

Scrum Lifecycle



Source: [Microsoft]

Sprints

Scrum sprints are short, time-boxed periods (often a week or two, but less than a month) where teams focus on completing a set amount of work, enabling incremental delivery.

Source: Atlassian

Homework 2: Sprint 1 – An Unstyled Working Drum Simulator

Sprint Goal: Implement an unstyled, but functional drum simulator app prototype.

Discussion: Why *unstyled*?

Review: Prototype

> A user interface prototype is a **hypothesis** — a candidate design solution that you consider for a specific design problem.

- NN/Group

High-Fidelity Prototyping Risks

- **Cost:** High-fidelity prototypes can be costly. Only do as much as is necessary to test the design hypothesis.
- **Uncertainty:** Is it possible to implement the functionality of the design?
- **Design Lock-in:** Once a design is coded, it can be difficult to change (socially and technically).
- **Sunken Cost Fallacy:** Just because a lot of time and effort has been invested in a design does not mean you should continue with that design.
- **Scope Creep:** Adding new features or requirements that are not part of the original plan can lead to delays and increased complexity.
- ...

Gotcha: Coding the UI Design Too Early

Common mistake: Many teams code the user interface first (not the functionality), before evaluating whether it was possible to implement the design's functionality.

This often leads to **design lock-in**, because the team has invested a lot of time and effort into the design, and it can be **emotionally** difficult to change the design (sunken cost fallacy).

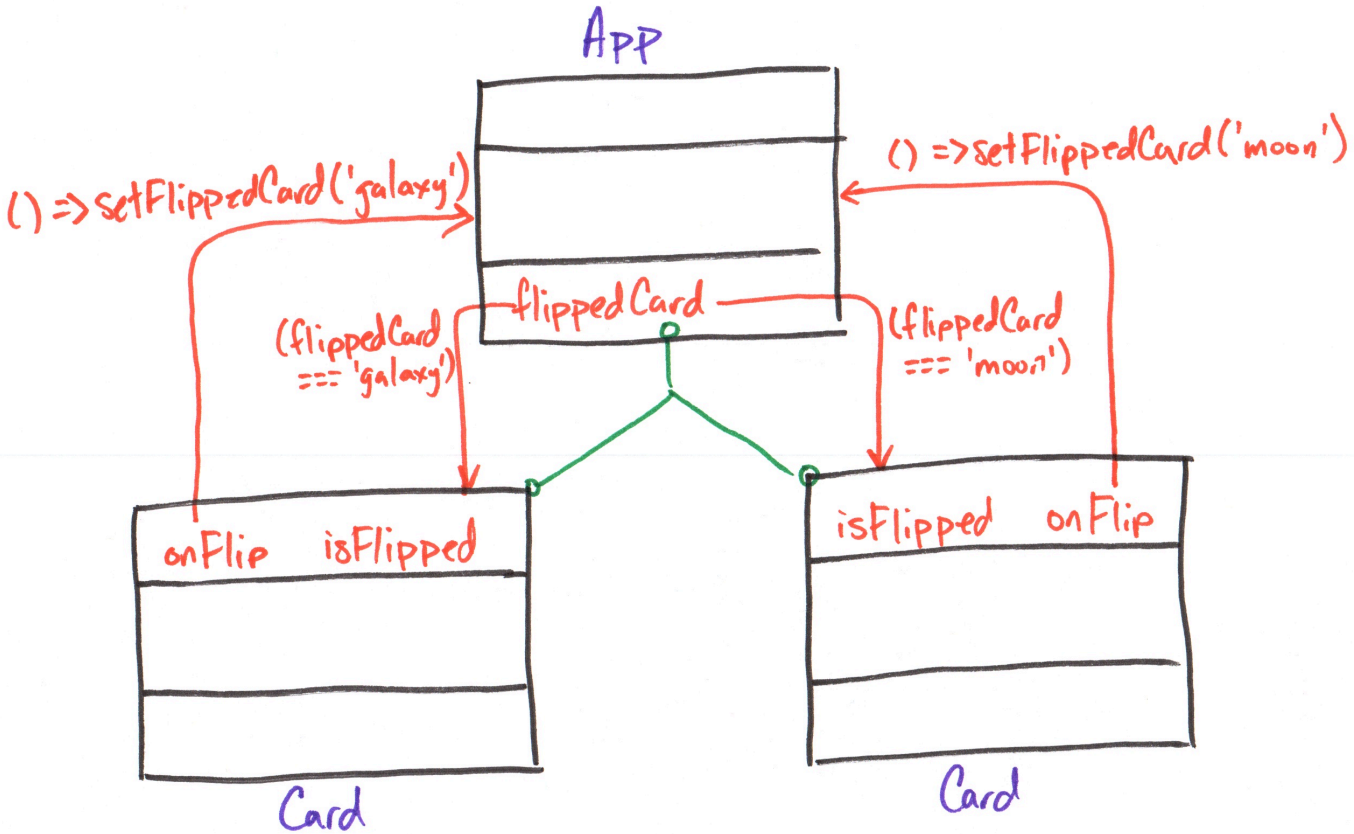
Solution: Implement the **greatest source of uncertainty** first. This is often the functionality.

Discussion: Homework 2

What is your greatest source of uncertainty for your drum simulator app?

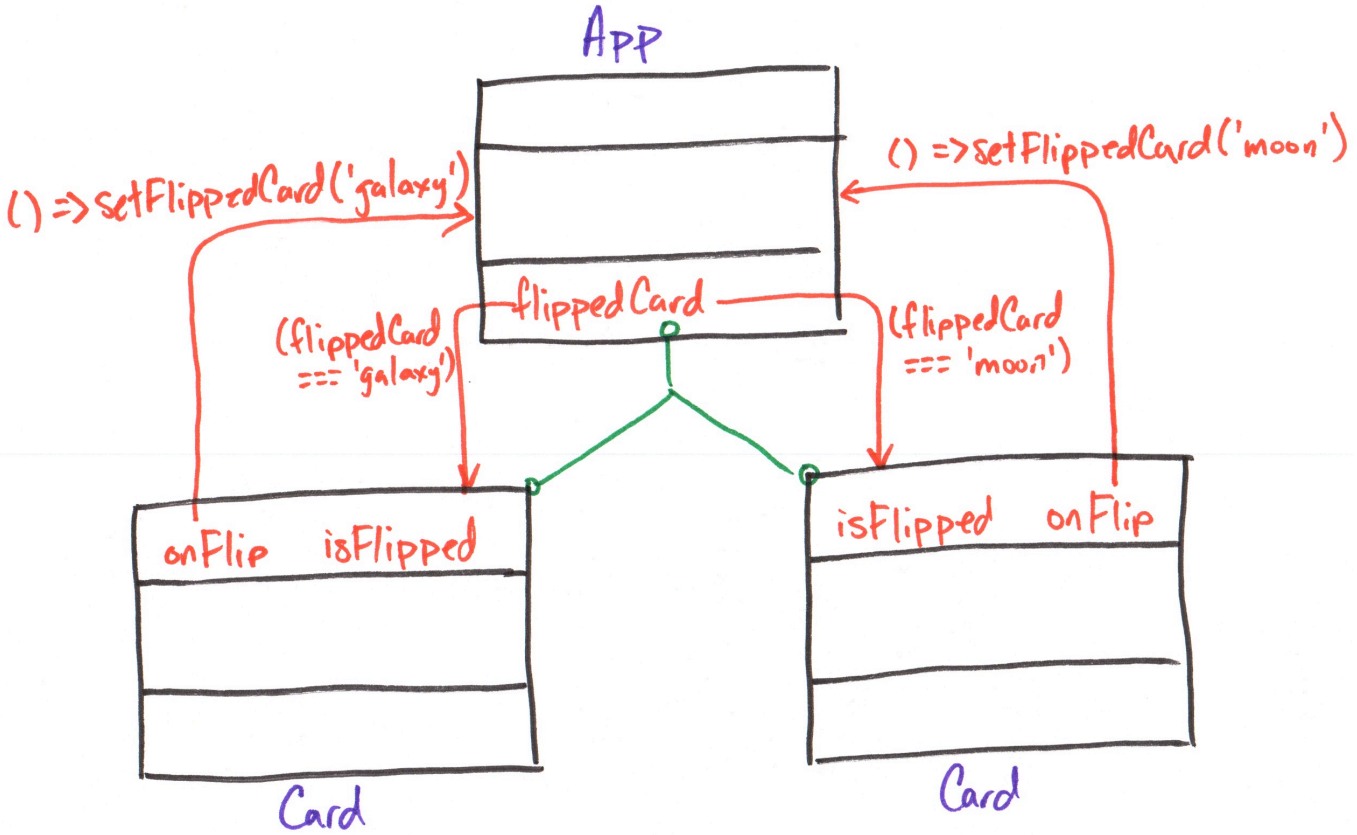
Review: Component Tree

A diagram to visualize the component hierarchy and the flow of data and state.



Activity: Drum Simulator Component Tree

Working with your **peers**, plan a component tree for your implementing your drum simulator app.



Studio: Homework 2, Sprint 1

Work on your getting the functionality of your drum simulator app working.

If you've already finished the functionality, **optionally** add these two features:

1. Dynamically increase or decrease the time between steps (i.e. tempo change)
2. Dynamically increase or decrease the number of steps (i.e. 4-step, 8-step, 16-step, etc.)

What’s Next

Homework 2, Sprint 1 - Due Monday, Sep 14, 2026, 11:59pm

Tue, Sep 15, 2026: Sprint 2 - Styling the App Prototype