Refactoring AngularJS

Connect.Tech 2016

Corinna Cohn @Corinna000

Challenges and solutions encountered building enterprise applications with Angular

Corinna Cohn @Corinna000

Grady Booch
@Grady_Booch

Revisiting code you wrote months or years ago is like seeing an old friend. Sometimes you share a laugh, sometimes you wish you'd never met.

10/5/16 8:03 PM

122 RETWEETS **155** LIKES

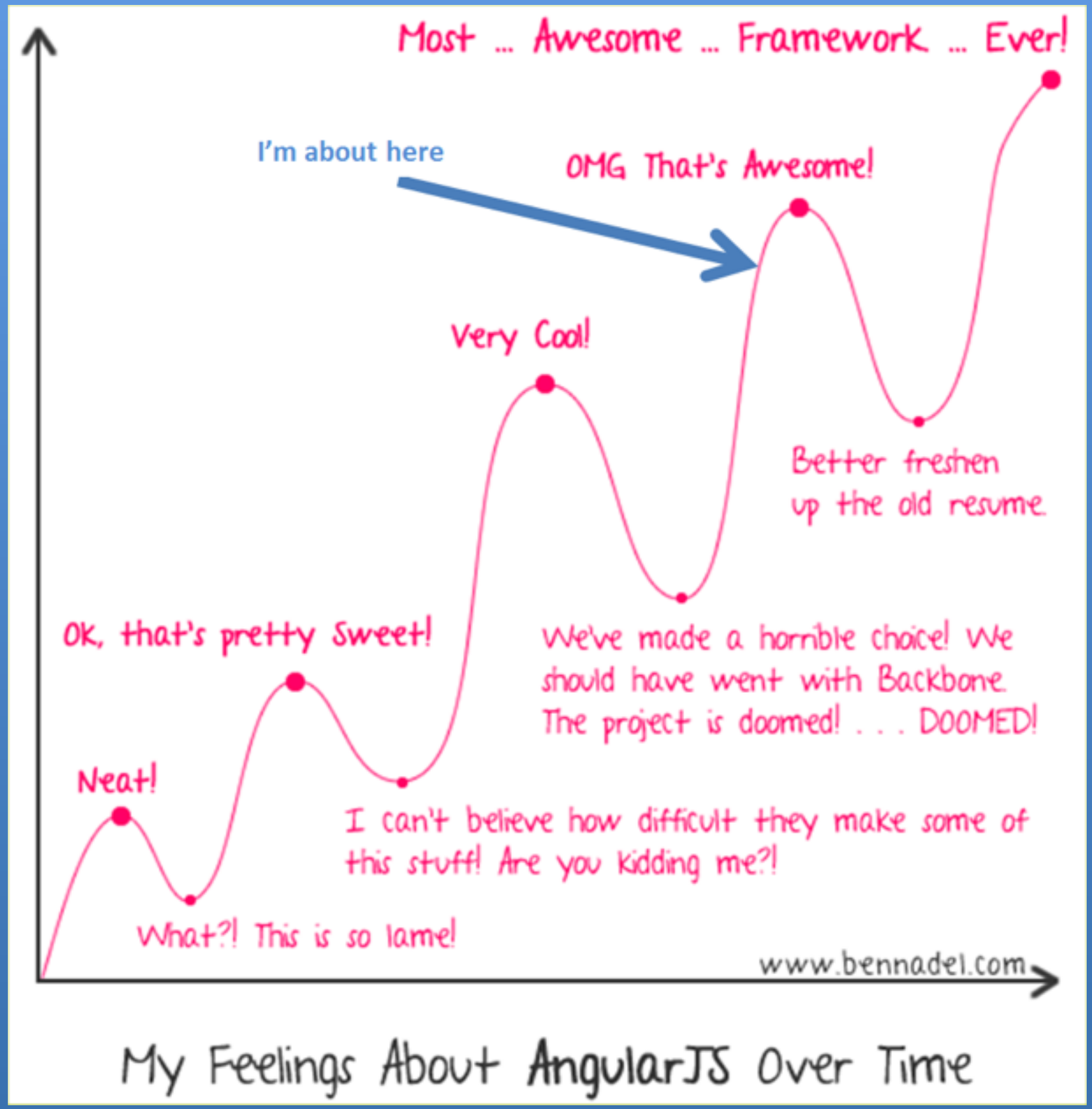
Sources of Angular technical debt

Corinna Cohn @Corinna000

Sources of Angular technical debt

1. AngularJS has a steep learning curve

Corinna Cohn @Corinna000



Ben Nadel www.bennadel.com

Sources of Angular technical debt

2. Many AngularJS developers are coming from Java and C#

Corinna Cohn @Corinna000

Sources of Angular technical debt

3. Lack of guidance on developing enterprise single-page applications

Corinna Cohn @Corinna000

Tutorials

Enterprise Applications

Refactoring

Refactoring (noun): a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior.

Why refactor?

- Resolve defects more confidently and quickly.
- Onboard new developers faster.
- Re-use features in other applications.

Goal of Refactoring

Complex  Simple

Make our code easier to reason about

Project-level refactoring

Corinna Cohn @Corinna000

Project Organization

Organize by feature, not type

```
src/
  - track-search/
  - track-player/
  - tournament-manager/
  - voting-results/
```

by feature

```
src/
  - views/
  - controllers/
  - services/
  - utilities/
```

by type

Organize by feature

- Easier to find related files
- Easier to refactor into separate modules
- See John Papa's LIFT principle

Style Guides

- John Papa's Angular 1.x
- Todd Motto's Angular 1.5 ES2015
- JSCS - JavaScript Code Style Checker

Our code base should look like it was written by one mind

Spot the difference

```
if (myCondition)
{
  someVar = callFunction(myCondition);
}
else
{
  someVar = false;
}
```

```
if (myCondition) {
          someVar = callFunction(myCondition);
} else {
          someVar = false;
}
```

Merely creating consistency is a worthwhile refactoring activity

Corinna Cohn @Corinna000

Unit testing

Corinna Cohn @Corinna000

Unit testing is not important for ensuring quality.

Unit testing is to help developers understand their solution.

Corinna Cohn @Corinna000

Unit testing is for helping the next developer understand your train of thought.

That next developer might be you.

Corinna Cohn @Corinna000

Unit tests help you find things you accidentally broke while refactoring

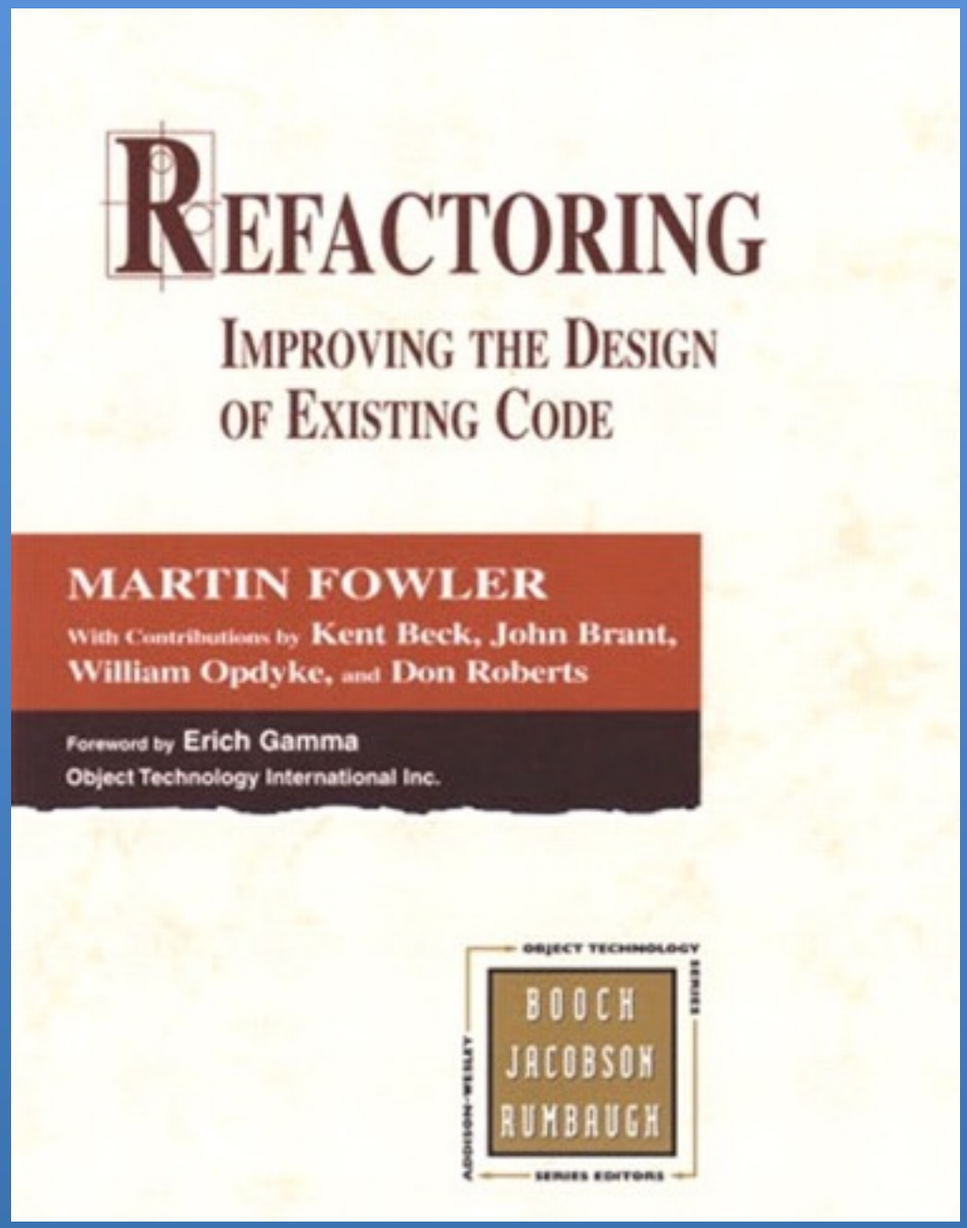
Unit tests help you refactor with confidence

Corinna Cohn @Corinna000

Code Refactoring

Corinna Cohn @Corinna000

- Renaming variables and methods
- Extracting code into separate methods and classes
- Organizing code to be at consistent levels of abstraction



Angular Problem Areas

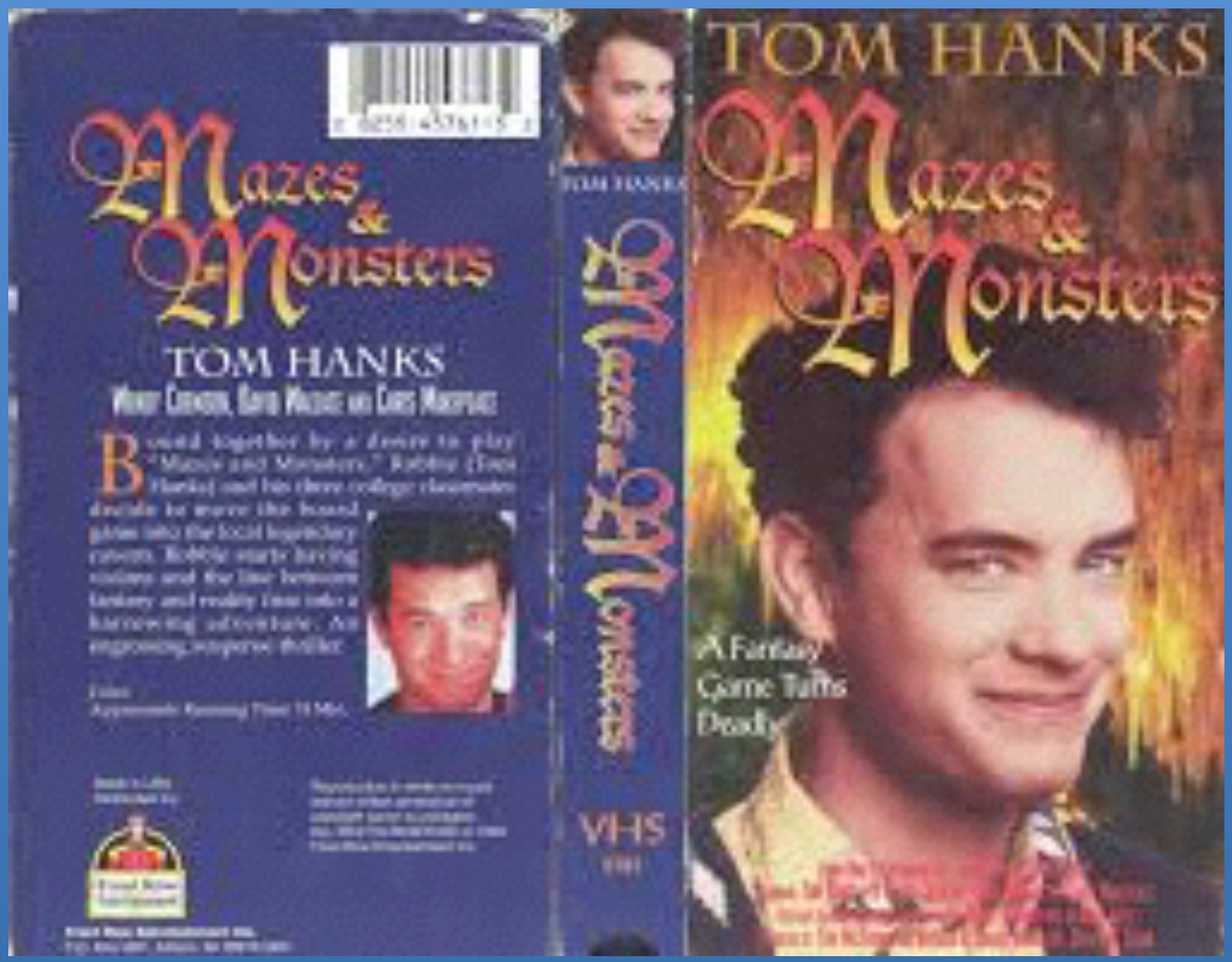
- Controllers
- Views/Templates
- Lack of modularity
- Generally, unclear separation of concerns

Controllers

So many problems, where to start?


I
CAN'T
EVEN

Modules and Monoliths



Corinna Cohn @Corinna000

Modules

```
angular.module('myApp', [
   'ui-router', 'ui-bootstrap', 'moment'
]);
```

Don’t do this

Modules

```
angular.module('myApp', [
  // first-party
  'votingHistory', 'tournamentManager',
  'userPreferences'

  // company libraries
  'soundCloudService', 'trackPlayer'

  // third-party
  'ui-router', 'ui-bootstrap', 'moment'
]);
```

Do this

Third party libraries

- Wrap non-Angular libraries into a constant or service.
- Makes testing easier by allowing developers to mock dependencies

Angular Modules

- Easier to test
  - Isolate side effects into their own modules
- Easier to reuse
- Easier to maintain

Prefer smaller pieces with fewer individual responsibilities.

Corinna Cohn @Corinna000

Controllers

Corinna Cohn @Corinna000

“Skinny” Controllers

- Ideally: mediate between models and views.
- Acceptable: view helpers, maybe some application logic
- application logic is a sign to build a component or directive

Controller trouble (1)

- Application state management
  - User login/logout
  - Error handling
  - Messages to the user (eg, Toast)

Controller refactoring (1)

- Move state management into services
  - One service per state
  - Clean, documented API
- Use $exceptionHandler service
  - Learn how to use Angular decorators

Documentation

- Documentation !== Comments
  - Documentation explains how to use the exposed interfaces in your code.
  - Comments are inserted within functions and explain specifics.

This is your default documentation

```
/**
 * @name UserPropertiesService
 *
 * This service stores user properties,
 * but guess what, it also has two other
* responsibilities that you as a
* new developer will spend hours
* tracking down.
*
* Good luck.
*/
```

Controller trouble (2)

- Business logic
  - Persistence - getting, storing data
  - Updating, creating models, validation
  - Interaction between models

Controller refactoring (2)

- Move data management into services
  - One service per resource
- Use a data library like JS-Data or Flux
- Use the factory pattern to create rich data models
  - Virtual properties and behaviors

Rich models

- Models that describe their own behaviors
  - calculating virtual properties
- Use factories to convert JSON data to instantiated functions.
- Behavior is located on the model, not in a service or controller.

Mental break time

Reduce complexity.

Emphasize simplicity.

Corinna Cohn @Corinna000

Controller trouble (3)

- DOM manipulation
- $rootScope overuse

Controller refactoring (3)

- Re-write DOM manipulation as a directive or component.
- Move properties and methods attached to $rootScope to their own services.
- Wrap a service $rootScope.$broadcast to help encapsulate messages.

Controller trouble (4)

- Separation of concerns
  - Binding multiple models
  - Binding too many services
- Uncontrolled dependencies on parent controllers

Controller refactoring (4)

- Re-write DOM manipulation as a directive or component.
- Move properties and methods attached to $rootScope to their own services.
- Wrap a service $rootScope.$broadcast to help encapsulate messages.

Template trouble (1)

- Really long templates

```
<div class="container">
  <div class="jumbotron">
    <h1><i>SlamOff</i></h1>

    <p>The Slam Poetry Bracket App</p>
  </div>
  <div>
    <div ng-include="'app/search/search-partial.html'"></div>
    <div ng-include="'app/search/search-results-partial.html'"></div>
    <div ng-show="vm.tracks.length">
      <div>Position: {{ vm.position | date:'mm:ss' }}</div>
      <div>Duration : {{ vm.duration | date:'mm:ss' }}</div>
      <div class="progress">
        <div class="progress-bar" role="progressbar" aria-valuenow="60" aria-valuemin="0" aria-valuemax="100"
             ng-style="vm.percentPlayed">
          <span class="sr-only">{{ vm.percentPlayed }} Complete</span>
        </div>
      </div>
      <table class="table" id="brackets">
        <tr>
          <td class="round-one">
            <h3>Round 1</h3>
            <div ng-hide="vm.getWinner(0)">

              <div class="track">
                {{ vm.tracks[0].title }} ({{ vm.tracks[0].duration | date:'mm:ss'}})
                <span ng-click="vm.playTrack(vm.tracks[0])"><i class="glyphicon glyphicon-play"></i></span>
                <span ng-click="vm.currentlyPlayingTrack.stop()"><i class="glyphicon glyphicon-stop"></i></span>
                <button ng-click="vm.vote(0, 0)" class="label label-success">vote!</button>
              </div>
              vs.
              <div class="track">
                {{ vm.tracks[1].title }} ({{vm.tracks[1].duration | date:'mm:ss'}})
                <span ng-click="vm.playTrack(vm.tracks[1])"><i class="glyphicon glyphicon-play"></i></span>
                <span ng-click="vm.currentlyPlayingTrack.stop()"><i class="glyphicon glyphicon-stop"></i></span>
                <button ng-click="vm.vote(0, 1)" class="label label-success">vote!</button>
              </div>
            </div>
            <div ng-show="vm.getWinner(0)">
              Winner: <span class="winner">{{ vm.getWinner(0).title }}</span>
```

```
 </div>
</td>
<td rowdiv="2" class="round-two">

  <h3>Semi-Final</h3>
  <div ng-show="vm.previousRoundsComplete(0) && !vm.getWinner(4)">

    <div class="track">
      {{ vm.getWinner(0).title }} ({{ vm.getWinner(0).duration | date:'mm:ss'}})
      <span ng-click="vm.playTrack(vm.getWinner(0))"><i class="glyphicon glyphicon-play"></i></span>
      <span ng-click="vm.currentlyPlayingTrack.stop()"><i class="glyphicon glyphicon-stop"></i></span>
      <button ng-click="vm.vote(4, vm.winners[0])" class="label label-success">vote!</button>
    </div>
    vs.
    <div class="track">
      {{ vm.getWinner(1).title }} ({{vm.getWinner(1).duration | date:'mm:ss'}})
      <span ng-click="vm.playTrack(vm.roundOne[1])"><i class="glyphicon glyphicon-play"></i></span>
      <span ng-click="vm.currentlyPlayingTrack.stop()"><i class="glyphicon glyphicon-stop"></i></span>
      <button ng-click="vm.vote(4, vm.winners[1])" class="label label-success">vote!</button>
    </div>
  </div>
  <div ng-show="vm.getWinner(4)">
    Winner: <span class="winner">{{ vm.getWinner(4).title }}</span>
  </div>

</td>
<td rowdiv="4" class="round-three">

  <h3>Final Round</h3>
  <div ng-show="vm.getWinner(4) && vm.getWinner(5) && !vm.getWinner(6)">

    <div class="track">
      {{ vm.getWinner(4).title }} ({{ vm.getWinner(4).duration | date:'mm:ss'}})
      <span ng-click="vm.playTrack(vm.getWinner(4))"><i class="glyphicon glyphicon-play"></i></span>
      <span ng-click="vm.currentlyPlayingTrack.stop()"><i class="glyphicon glyphicon-stop"></i></span>
      <button ng-click="vm.vote(6, vm.winners[4])" class="label label-success">vote!</button>
    </div>
    vs.
    <div class="track">
      {{ vm.getWinner(5).title }} ({{vm.getWinner(5).duration | date:'mm:ss'}})
```

```
          <span ng-click="vm.playTrack(vm.getWinner(5))"><i class="glyphicon glyphicon-play"></i></span>
          <span ng-click="vm.currentlyPlayingTrack.stop()"><i class="glyphicon glyphicon-stop"></i></span>
          <button ng-click="vm.vote(6, vm.winners[5])" class="label label-success">vote!</button>
        </div>
      </div>
      <div ng-show="vm.getWinner(6)">
        Winner: <span class="winner">{{ vm.getWinner(6).title }}</span>
      </div>
    </td>
  </tr>
  <tr>
    <td class="round-one">
      <div ng-hide="vm.getWinner(1)">
        <div class="track">
          {{ vm.tracks[2].title }} ({{ vm.tracks[2].duration | date:'mm:ss'}})
          <span ng-click="vm.playTrack(vm.tracks[2])"><i class="glyphicon glyphicon-play"></i></span>
          <span ng-click="vm.currentlyPlayingTrack.stop()"><i class="glyphicon glyphicon-stop"></i></span>
          <button ng-click="vm.vote(1, 2)" class="label label-success">vote!</button>
        </div>
        vs.
        <div class="track">
          {{ vm.tracks[3].title }} ({{vm.tracks[3].duration | date:'mm:ss'}})
          <span ng-click="vm.playTrack(vm.tracks[3])"><i class="glyphicon glyphicon-play"></i></span>
          <span ng-click="vm.currentlyPlayingTrack.stop()"><i class="glyphicon glyphicon-stop"></i></span>
          <button ng-click="vm.vote(1, 3)" class="label label-success">vote!</button>
        </div>
      </div>
      <div ng-show="vm.getWinner(1)">
        Winner: <span class="winner">{{ vm.getWinner(1).title }}</span>
      </div>
    </td>
  </tr>
  <tr>
    <td class="round-one">
      <div ng-hide="vm.getWinner(2)">
        <div class="track">
          {{ vm.tracks[4].title }} ({{ vm.tracks[4].duration | date:'mm:ss'}})
          <span ng-click="vm.playTrack(vm.tracks[4])"><i class="glyphicon glyphicon-play"></i></span>
          <span ng-click="vm.currentlyPlayingTrack.stop()"><i class="glyphicon glyphicon-stop"></i></span>
          <button ng-click="vm.vote(2, 4)" class="label label-success">vote!</button>
        </div>
```

Template refactoring (1)

- Use ng-include to break large templates into smaller templates.
- Create components or directives

```
<div>
  <div ng-include="'app/search/search-partial.html'"></div>
  <div ng-include="'app/search/search-results-partial.html'"></div>
  <div ng-show="vm.bm.tracks.length">
    <div ng-include="'app/track-player/track-progress-partial.html'"></div>
    <div ng-include="'app/brackets/brackets-partial.html'"></div>
    <div ng-include="'app/brackets/voting-summary.html'"></div>
  </div>
</div>
```

Template trouble (2)

- Avoid display logic in the template

```
<div
   ng-show="vm.getWinner(4) &&
      vm.getWinner(5) &&
      !vm.getWinner(6)”>
</div>
```

Resources

- AngularJS Style Guide by John Papa
- AngularJS Patterns: Clean Code by John Papa
- Clean Code by Robert Martin
- Refactoring by Martin Fowler
- Refactoring to Patterns by Joshua Kerievsky
- Working with Legacy Code by Michael Feathers

Questions

Corinna Cohn @Corinna000