


Refactoring AngularJS Controllers

Corinna Cohn (@corinna000)
<https://github.com/corinna000>

Consultant with Fusion Alliance

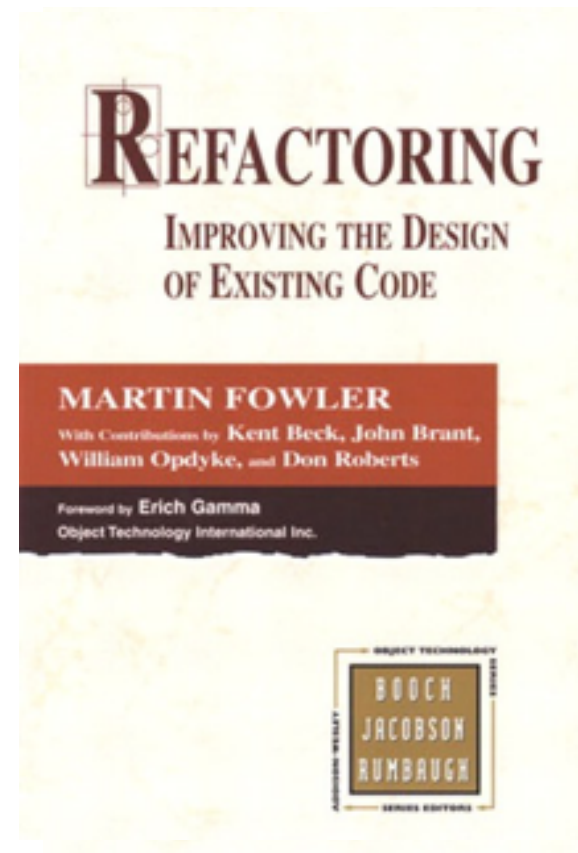
1 minute

About Me (@corinna000)

- I used to write awful code.
 - Now I write better code.
 - Someday I will write good code.
 - (This is the holy grail)
 - (follow me on Twitter and/or GitHub)
- 
- A hand-drawn red arrow originates from the bottom of the list, specifically near the text "(follow me on Twitter and/or GitHub)", and points diagonally upwards and to the left towards the first bullet point, "I used to write awful code.".

“Refactoring is a controlled technique for improving the design of an existing code base.”

—Martin Fowler





Do your controllers look like this?
(this is not my kitchen)

- * 1 minute
- * Not opinionated
- * Dependency Injection is optional
- * Examples demonstrate putting code into global space



Let's make them look like this.
(yeah, our code will never look this good, but let's try)

- * 1 minute
- * Dependency Injection works great for unit testing
- * Karma Runner, Jasmine, Angular mocking library

SlamOff!

Poetry Exploration App

- Search poetry on SoundCloud by topic.
- Listen to 8 poems and SlamOff will guide you through discovering qualities about poetry you enjoy.

4 minutes

MainController Responsibilities

1. Search for SoundCloud tracks
2. Select the tracks to add to the bracket
3. Play/Stop tracks
4. Run the progress bar
5. Describe the display logic for the bracket
6. Setup the tracks for the tournament
7. Update the vote tally for the tracks
8. Save/Load the voting data (a vestigial feature)
9. Sort/Display the final results of the tournament

3 minutes (15 minute mark!)

Why This Approach Is Painful

- Updating a track required changing similar HTML 10+ times.
- Logic for interacting with SoundCloud is not reusable.
- Complicated rules for tournament display are not testable or reusable.
- Plain JSON model does not contain “vote” property nor any logic for recording a vote.
- Too many lines of HTML in one file harms maintainability and collaboration (hello, merge conflicts!).

4 minutes

Angular Refactoring Patterns

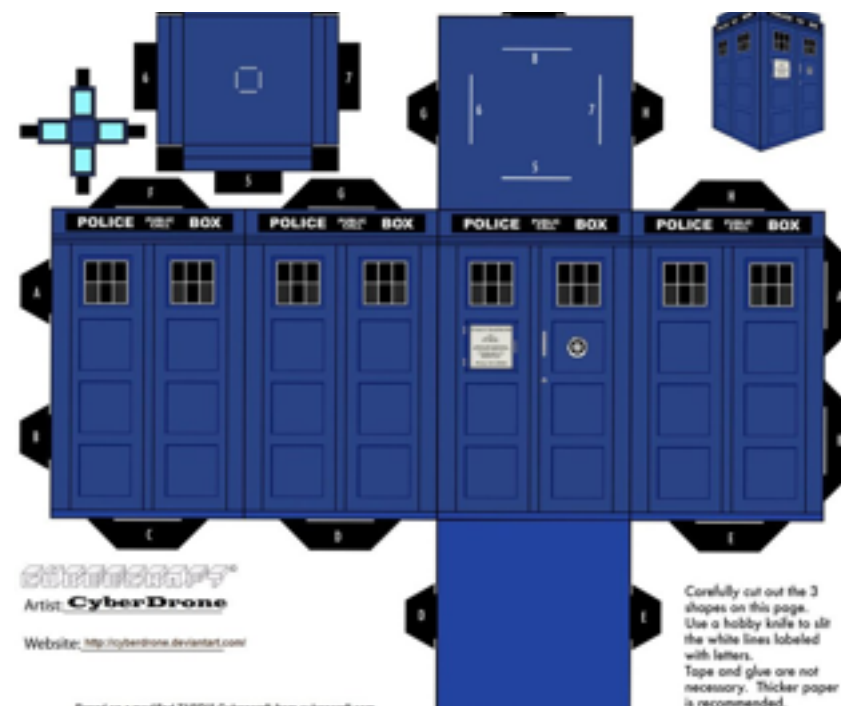
- Find repeated groups of HTML that have a clear interface and build a directive from them. (*[Extract Directive From View](#)*)
- Call services from directives instead of controllers wherever possible. (*[Encapsulate Service With Directive](#)*)
- Move complex display rules into separate services. (*[Separate Domain from Presentation](#)*)

patterns ctd...

- Convert JSON into rich models that are responsible for their own updates. ([*Replace Data Value with Object*](#))
- Use `ng-include` to separate discreet views into their own templates. ([*Move Templates*](#))



Unit testing is like
owning a time machine



okay, they do require a little
extra work

Refactoring process

- Discover the interfaces of the current implementation.
- Figure out how to separate a piece of the implementation.
- Write unit tests.
- Integrate our changes.

<so-voting-track>

- External interface: We need a **track** to play
- Internal interface: play(), stop()

```
3  
4 <so-voting-track track="track[0]"></so-voting-track>
```

.factory('trackPlayer')

- Wraps the SoundCloud sound manager object.
- API: play(), stop(), getDuration(), getPosition()
- Used by: <so-track-player>,
<track-player-progress>

```
var service = {  
  play: play,  
  stop: stop,  
  getDuration: getDuration,  
  getPosition: getPosition  
};
```

<track-player-progress>

- External interface: None! Uses `trackPlayer` service to determine information about the current track.

```
1   
2 <track-player-progress></track-player-progress>  
3
```

Skip if running low on time

.factory('trackSearch')

- Moves the SoundCloud track search into a reusable service.
- Sets up the defaults for searching in our application.
- Returns a Promise that resolves to a track list.

```
var service = {  
  search: search  
};
```

.factory('bracketManager')

- Initializes a new tournament round.
- Tracks the state of tournament progress.
- Provides a method to help the UI know what to display.
- Registers votes for tracks.

```
var service = {  
  initialize: initialize,  
  vote: vote,  
  getWinner: getWinner,  
  showCurrentRound: showCurrentRound  
};
```

<div ng-include=""search.html"">

- `main.html` still has too many responsibilities.
- If one feature needs to be updated, a file affecting three other features needs to be changed.
- Solution: move html into sub templates and use `ng-include` to add to view.

```
<div ng-include=""app/main/templates/search-form.html""></div>
<div ng-include=""app/main/templates/search-results.html""></div>
  <div ng-show=""vm.bs.tracks.length">
    <track-player-progress></track-player-progress>
    <div ng-include=""app/main/templates/brackets.html""></div>
    <div ng-include=""app/main/templates/voting-summary.html""></div>
```