Why Test-Driven Development?
- ridiculously difficult to add tests afterwards
- can only “functional test” afterwards
- code isolation techniques
- emergent design
- refactor with confidence
- throw away your debugger
- communicate your intent

Why Test Drive JavaScript?
- JavaScript is a real language, why treat it differently?
- dynamic nature of JavaScript makes it easy to test
- debugging browser quirks easier with test suite
- once you get the TDD bug, you can't go back

Testing Tools:
- Jasmine - BDD-style; my favorite!
  - github.com/pivotal/jasmine
  - github.com/pivotal/jasmine-ruby
- qUnit - old-school xUnit; jQuery's favorite
  - docs.jquery.com/QUnit
- Blue Ridge - Ruby on Rails plugin
  - www.blueridgejs.org
- JSpec - another BDD-style
  - visionmedia.github.com/jspec

To install Jasmine specs:
- gem install jasmine-ruby
- cd my-project
- jasmine init
- rake jasmine
- rake jasmine:ci (to run Selenium)

When to Test:
- anytime there’s logic
- complicated DOM effects
- training wheels

When Not to Test:
- declarative code
- event wiring
- technical spikes

Red, Green, Refactor:
- write a failing test
  - start with the assertion
  - then fill in the background
- write the simplest code that makes that test pass
- refactor to remove duplication (if any)
- repeat

Testing Guidelines:
- one assertion per test
- maintain high test coverage (but 100% not required)
- use plain English descriptions; use client-speak

Books to Read:
- Test-Driven Development: By Example by Kent Beck
- The RSpec Book by David Chelimsky, et al.

Larry Karnowski, Relevance, Inc.
- www.thinkrelevance.com
- larry@thinkrelevance.com
- Twitter & GitHub: karnowski
- Sample code: github.com/karnowski/spicywrath