Testing and Testability

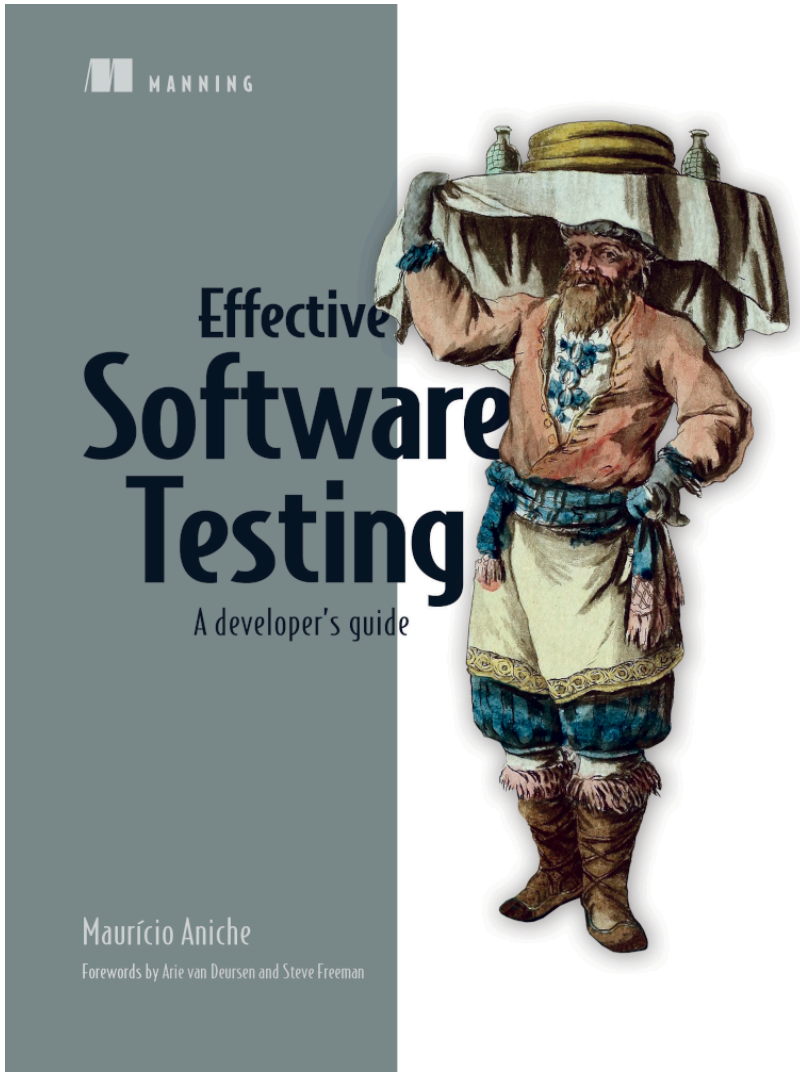
17-356/17-766
Software Engineering for Startups

https://cmu-17-356.github.io

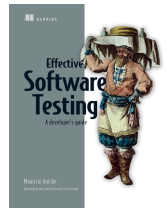
Administrivia

Software Testing

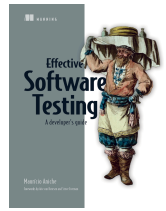
- Effective Software Testing should be systematic



Principles of software testing (or, why testing is so difficult)

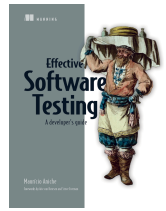


Principles of software testing (or, why testing is so difficult)



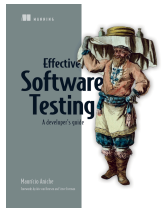
- Exhaustive testing is impossible

Principles of software testing (or, why testing is so difficult)



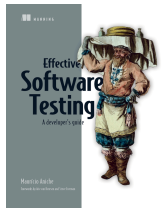
- Exhaustive testing is impossible
- Knowing when to stop testing

Principles of software testing (or, why testing is so difficult)



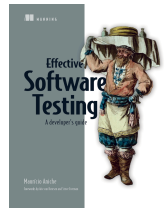
- Exhaustive testing is impossible
- Knowing when to stop testing
- Variability is important (the pesticide paradox)

Principles of software testing (or, why testing is so difficult)



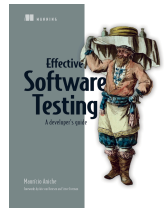
- Exhaustive testing is impossible
- Knowing when to stop testing
- Variability is important (the pesticide paradox)
- Bugs happen in some places more than others

Principles of software testing (or, why testing is so difficult)



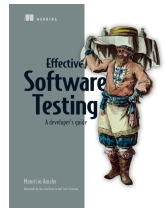
- Exhaustive testing is impossible
- Knowing when to stop testing
- Variability is important (the pesticide paradox)
- Bugs happen in some places more than others
- No matter what testing you do, it will never be perfect or enough

Principles of software testing (or, why testing is so difficult)



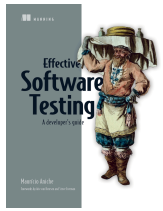
- Exhaustive testing is impossible
- Knowing when to stop testing
- Variability is important (the pesticide paradox)
- Bugs happen in some places more than others
- No matter what testing you do, it will never be perfect or enough
- Context is king

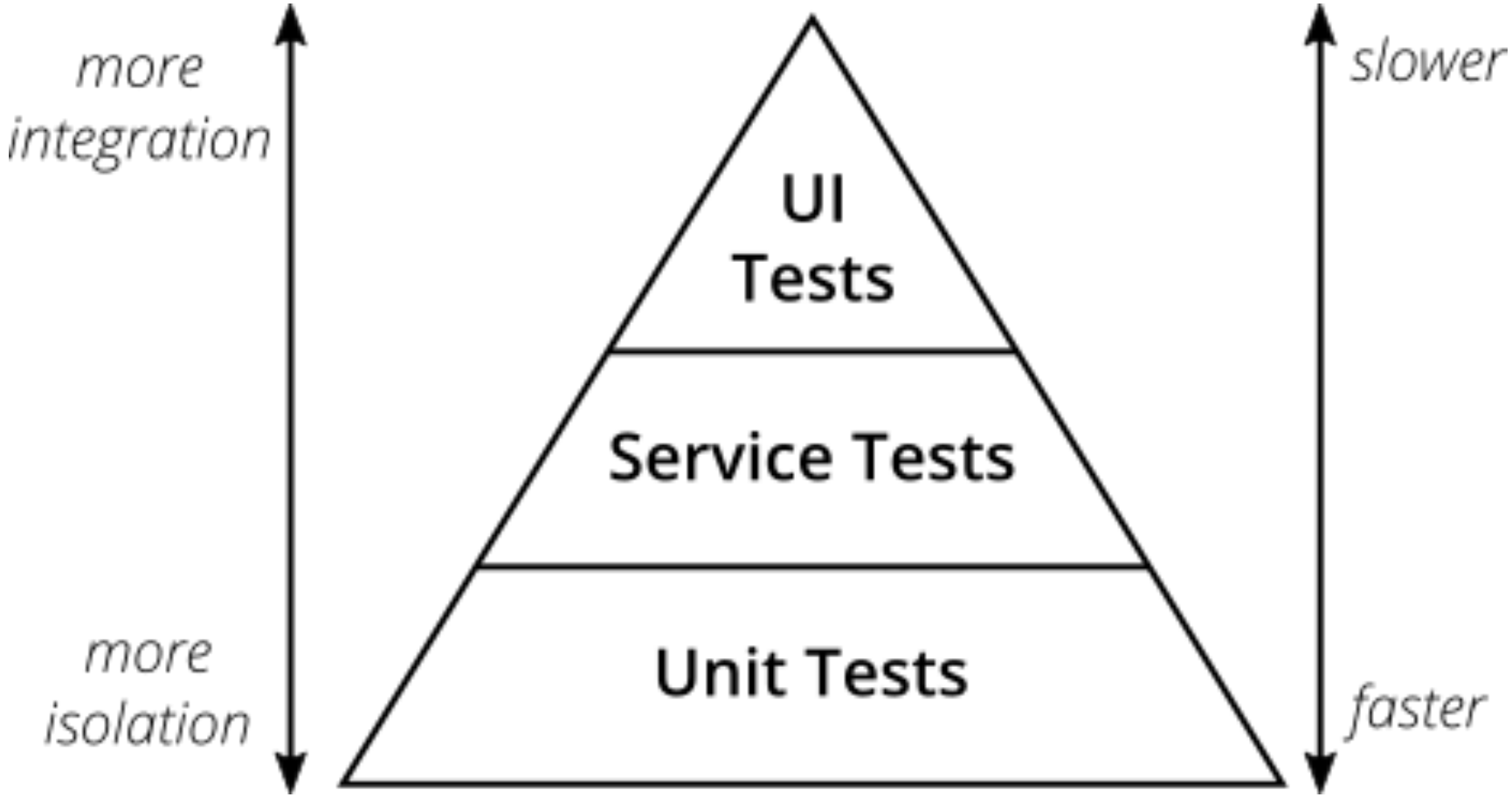
Principles of software testing (or, why testing is so difficult)



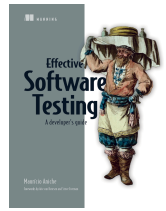
- Exhaustive testing is impossible
- Knowing when to stop testing
- Variability is important (the pesticide paradox)
- Bugs happen in some places more than others
- No matter what testing you do, it will never be perfect or enough
- Context is king
- Verification is not validation

Testing Pyramid

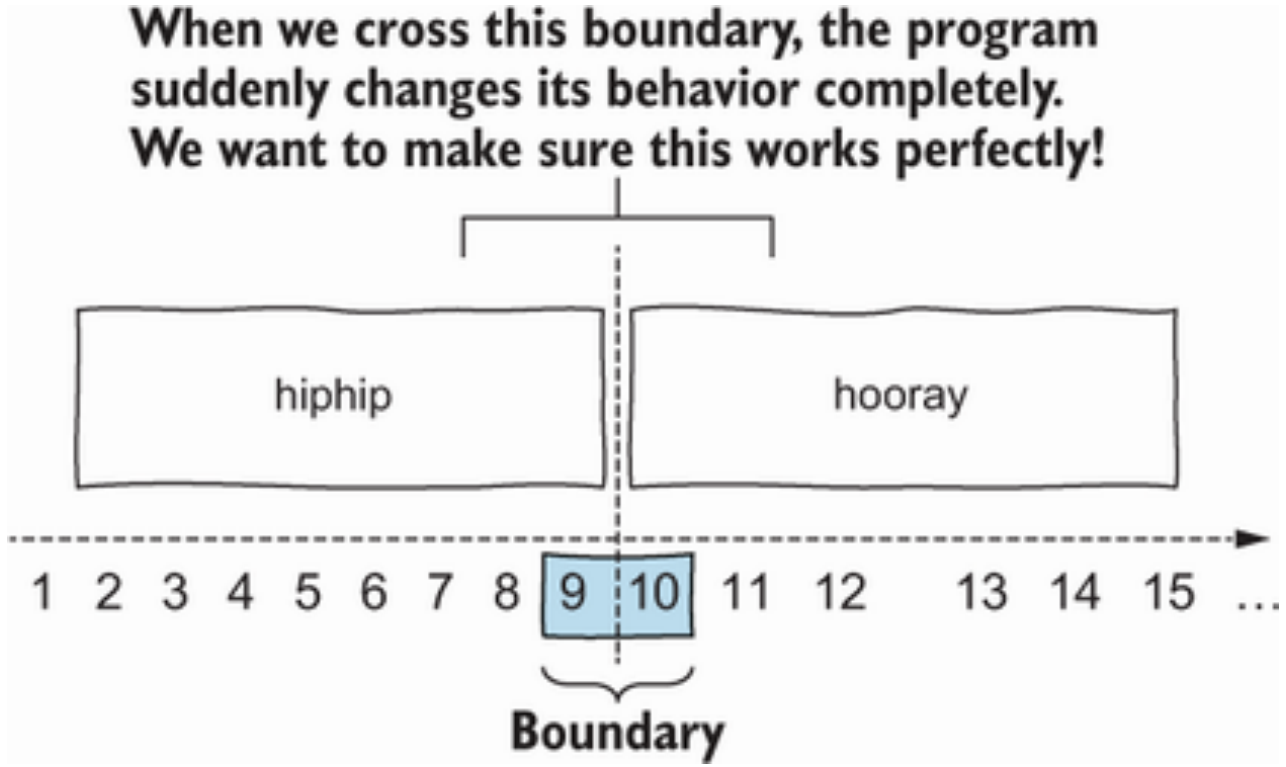




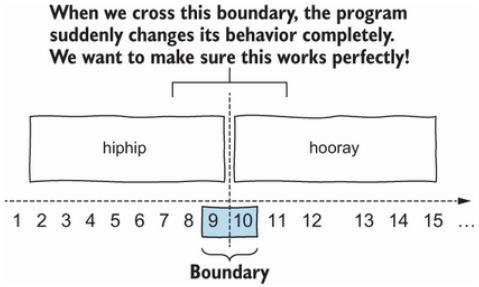
Testing should systematic

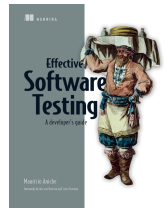


- Analyze the boundaries

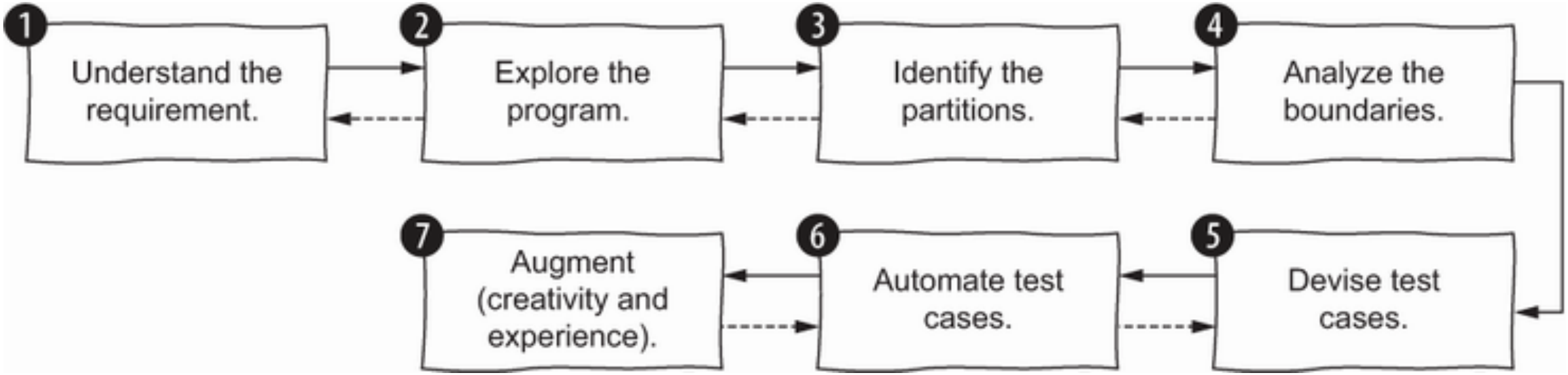


Specification-based Testing

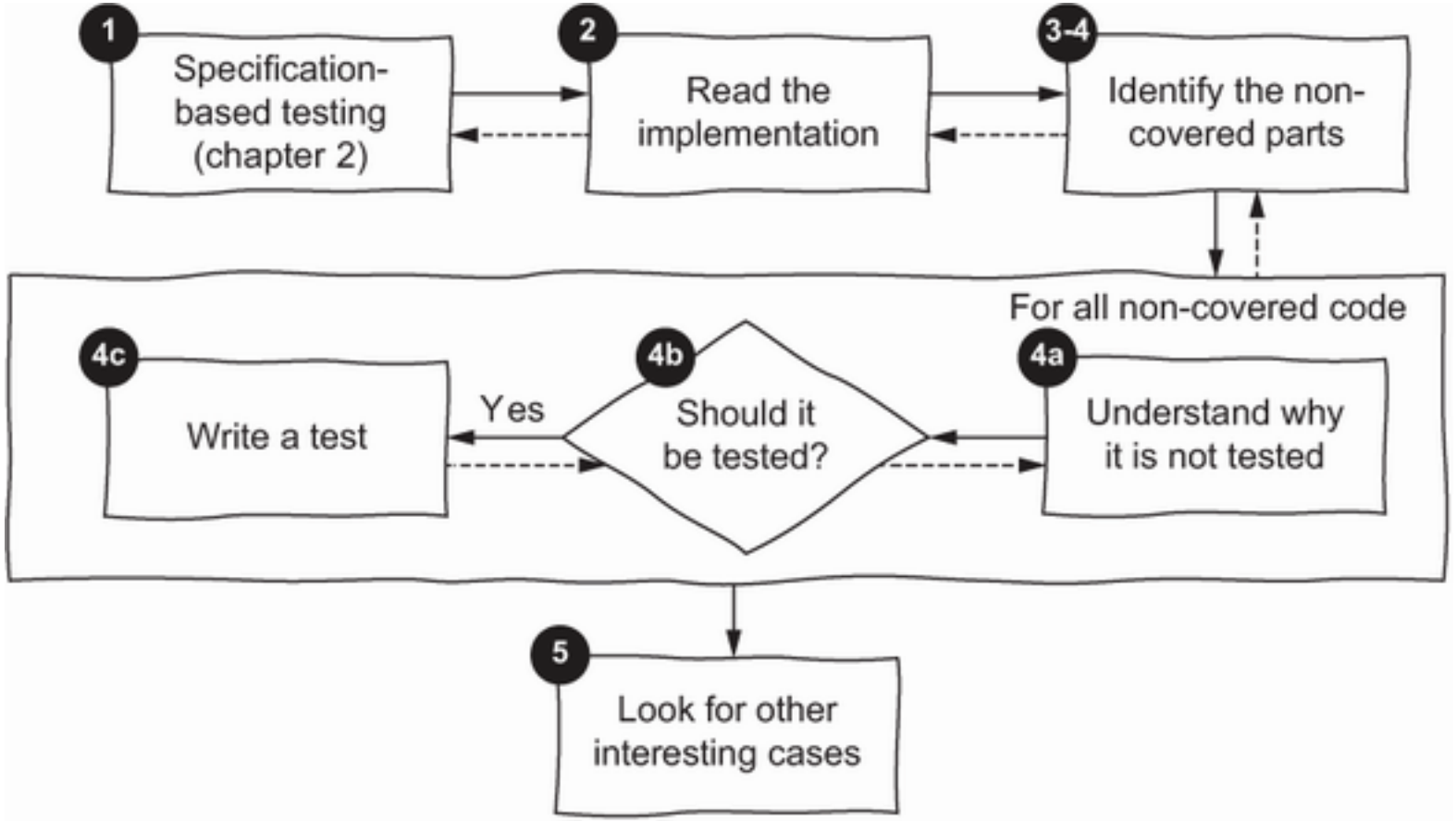




- Analyze the boundaries



Structural Testing



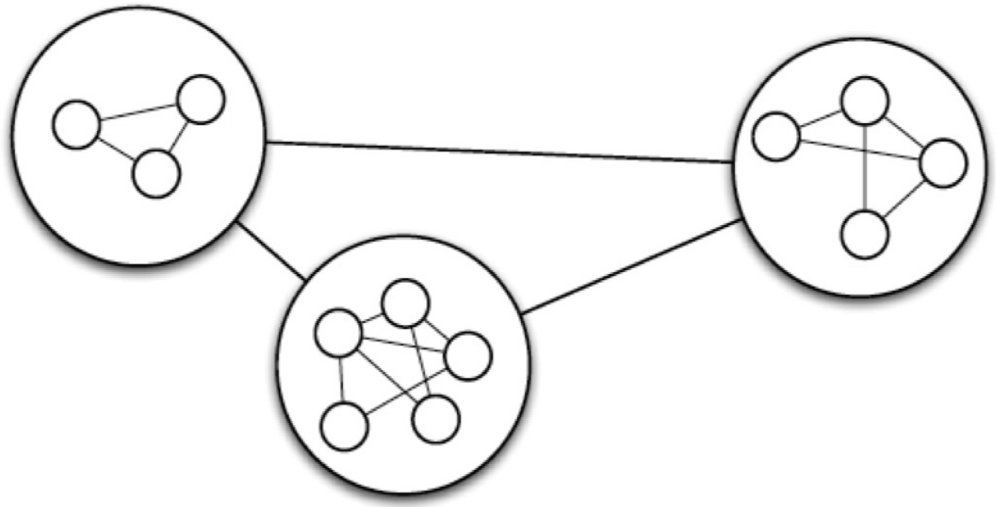
Code Coverage

- High code coverage isn't enough to guarantee correctness
- Low code coverage is an indicator of high uncertainty

```java
/**
 * @param price   The price to set.
 */
public void setPrice(String price) throws RecipeException{
    int amtPrice = 0;
    try {
        amtPrice = Integer.parseInt(price);
    } catch (NumberFormatException e) {
        throw new RecipeException("Price must be a positive integer");
    }
    if (amtPrice >= 0) {
        this.price = amtPrice;
    } else {
        throw new RecipeException("Price must be a positive integer");
    }
}
```

Design principle: Modularity

- Partitioning software into separate components in such a way that dependencies *between* components are *minimized*, while *maximizing* dependencies *within* components



Tests should have a single reason to fail

```
@Test
public void testAddPositive() {

    // Instantiate the object to test
    Calculator tester = new Calculator();

    // the numbers used in the test
    Integer[] list = {1,2,3};

    assertEquals(null, tester.calculate(null,"+"));
    assertEquals(Integer.valueOf(6), tester.calculate(list,"+"));
}
```

Test doubles

- **Stubs**: A dummy stand-in for the real collaborator for testing purposes
- **Fakes**: An optimized, thinned-down version of the real thing that replicates the behavior of the real thing, but without the side effects and other –[undesirable] consequences of using the real thing.
- **Spies**: Use a spy when the state of a collaborator is a secret, and you need to access that state to test an object
- **Mocks**: Object configured at runtime to behave in a certain way under certain circumstances

Dependency Injection

- Complex high-level elements should not directly depend on low-level elements that are likely to change: instead both should depend on abstractions.
- Abstractions should not depend on details; details should depend on abstractions.
- An object should not have to know what it is; is should instead care about what it does.

Dependency Injection

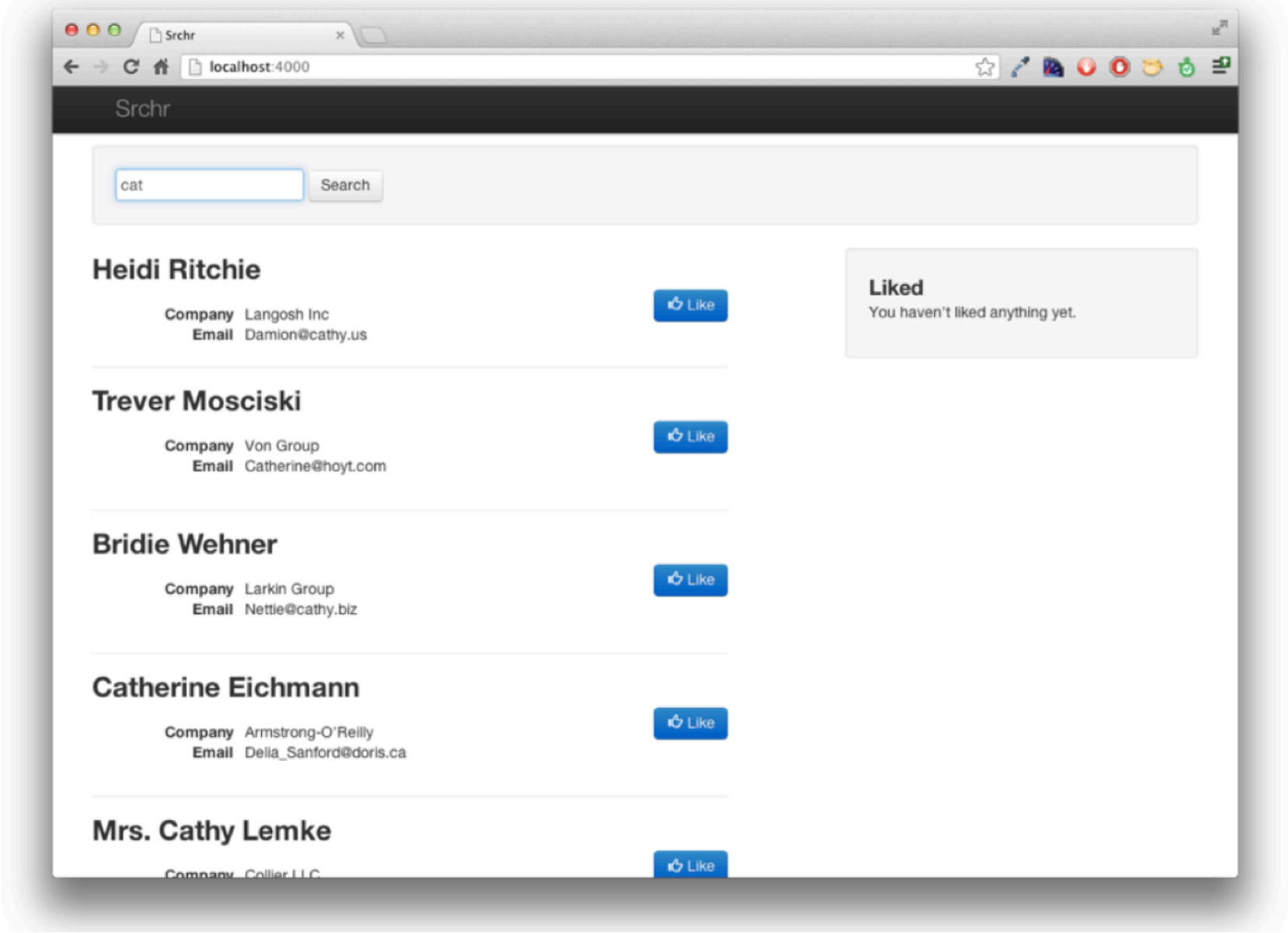
```
Time get_deadline() {
  return Clock::Now() + Seconds(30);
}
```

```
Time get_deadline(Clock* clock) {
  return clock->Now() + Seconds(30);
}
```

Testability Inhibitors

- Restrictions on
  - Instantiation (especially reliance on environment)
  - Invocation (private methods)
  - Observation (methods with no return values)
  - Substitution (hard coded collaborators)
  - Overriding (complicates doubles)
- Flakiness / Brittleness
  - Hermeticity
- Readability

Testability



http://alistapart.com/article/writing-testable-javascript/

Look at code handout

Aside: integration tests

- Unit test: “given input x, is the output y?”

- Integration test: do the pieces work together as expected?
  - This code is relatively integration-testable.

<back to unit tests>

This code has problems.

This code has problems.

- *Lack of structure*: almost everything happens in a single callback.
- Complex functions. Rule of thumb: if it's more than 10 lines, it's doing too much.

This code has problems.

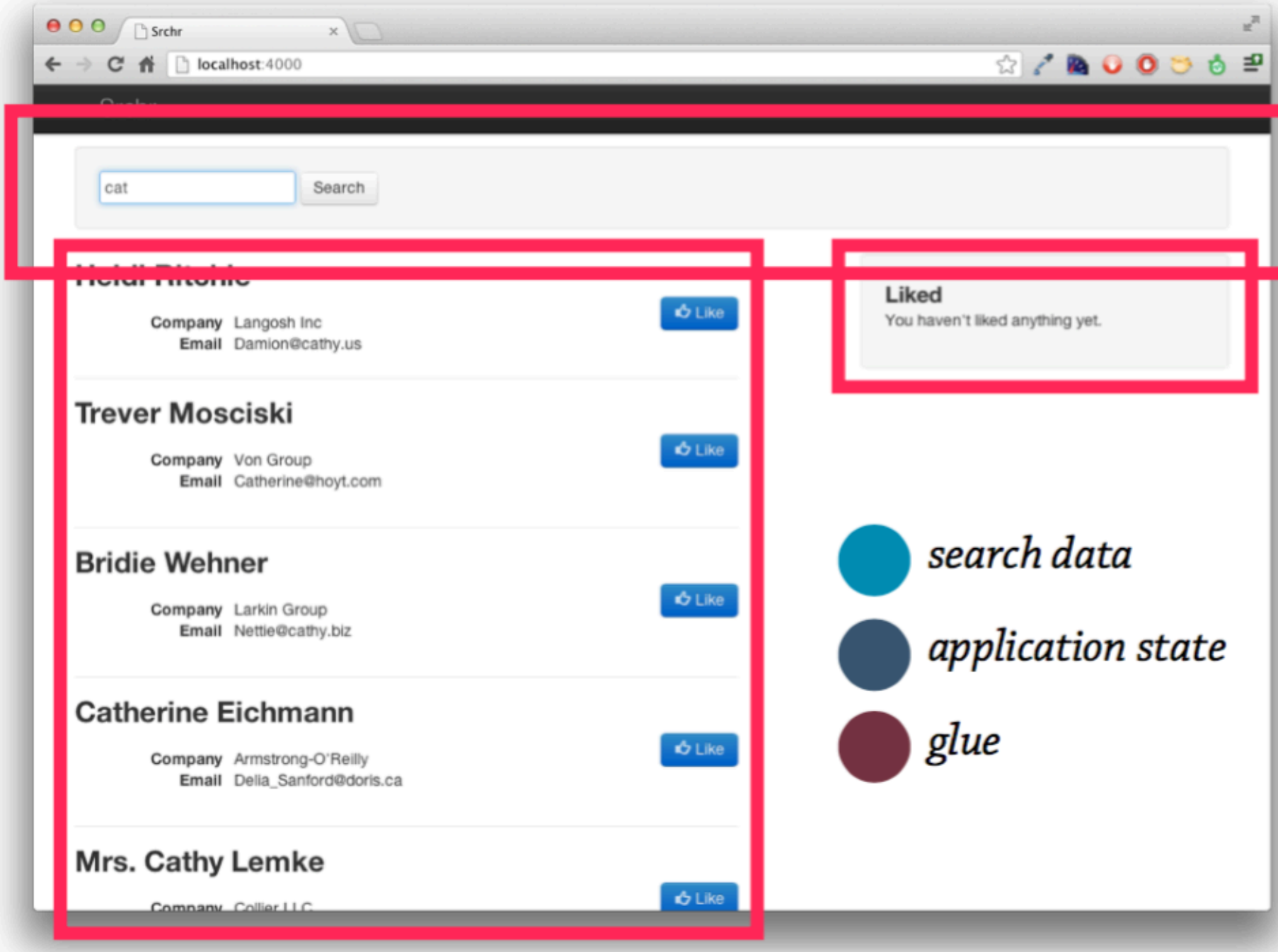
- *Lack of structure*: almost everything happens in a single callback.
- Complex functions. Rule of thumb: if it's more than 10 lines, it's doing too much.
- Hidden or shared state.

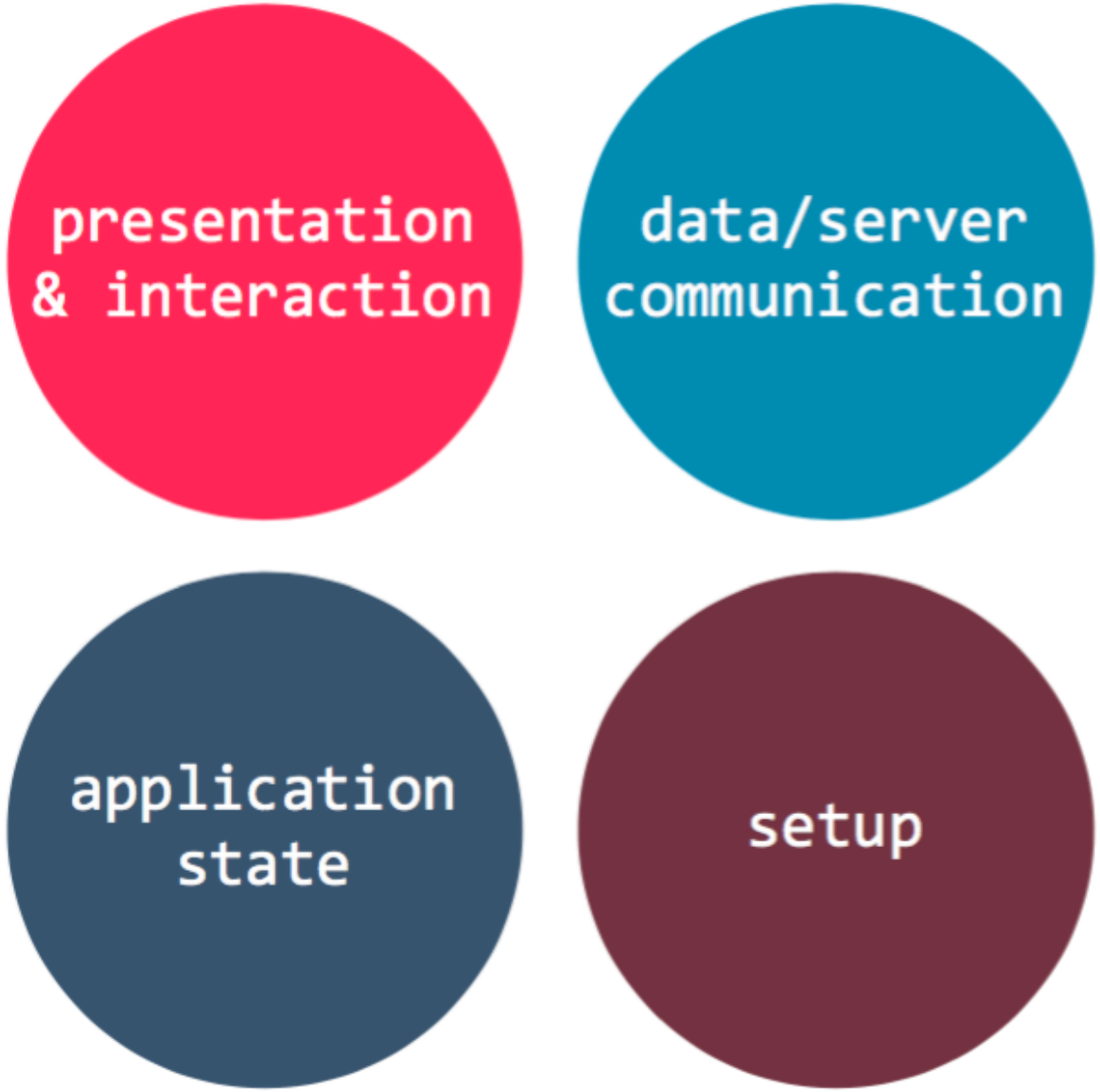
This code has problems.

- *Lack of structure*: almost everything happens in a single callback.
- Complex functions. Rule of thumb: if it's more than 10 lines, it's doing too much.
- Hidden or shared state.
- Tight coupling.

This code has problems.

- *Lack of structure*: almost everything happens in a single callback.
- Complex functions. Rule of thumb: if it's more than 10 lines, it's doing too much.
- Hidden or shared state.
- Tight coupling.
- Lack of configurability.


Srchr
localhost:4000
cat
Search
Company Langosh Inc
Email Damion@cathy.us
Like
Trever Mosciski
Company Von Group
Email Catherine@hoyt.com
Like
Bridie Wehner
Company Larkin Group
Email Nettie@cathy.biz
Like
Catherine Eichmann
Company Armstrong-O'Reilly
Email Delia_Sanford@doris.ca
Like
Mrs. Cathy Lemke
Like
Liked
You haven't liked anything yet.
search data
application state
glue


presentation
& interaction
data/server
communication
application
state
setup

Let’s reorganize:

- Represent each piece of behavior as a separate object that falls into one of the four areas of responsibility and doesn’t need to know about other objects.
- Support configurability, so we don’t have to replicate the entire HTML environment to write tests.
- Keep objects’ methods simple and brief.
- Use constructor functions to create instances of objects.

Let's reorganize:

- Represent each piece of behavior as a separate object that falls into one of the four areas of responsibility and doesn't need to know about other objects.
- Support configurability, so we don't have to replicate the entire HTML environment to write tests.
- Keep objects' methods simple and brief.
- Use constructor functions to create instances of objects.

Rewrite the code!

Sample Solution - Ajax

```
$.ajax('/data/search.json', {
  data : { q: query },
  dataType : 'json',
  success : function( data ) {
    loadTemplate('people-detailed.tmpl').then(function(t) {
      var tmpl = _.template( t );
      resultsList.html( tmpl({ people : data.results }) );
      pending = false;
    });
  }
});
```

Untestable

Testable

Sample Solution - Ajax

```
$.ajax('/data/search.json', {
  data : { q: query },
  dataType : 'json',
  success : function( data ) {
    loadTemplate('people-detailed.tmpl').then(function(t) {
      var tmpl = _.template( t );
      resultsList.html( tmpl({ people : data.results }) );
      pending = false;
    });
  }
});
```

Untestable

```
var SearchData = function () { };
SearchData.prototype.fetch = function (query) {
  var dfd;
  if (!query) {
    dfd = $.Deferred();
    dfd.resolve([]);
    return dfd.promise();
  }
  return $.ajax( '/data/search.json', {
    data : { q: query },
    dataType : 'json'
  }).pipe(function( resp ) {
    return resp.results;
  });
};
```

Testable

Sample Solution - Likes

```
var liked = $('#liked');


var resultsList = $('#results');


// ...


resultsList.on('click', '.like', function (e) {
  e.preventDefault();
  var name = $(this).closest('li').find('h2').text();
  liked.find( '.no-results' ).remove();
  $('<li>', { text: name }).appendTo(liked);
});
```

Untestable

Testable

Sample Solution - Likes

```
var liked = $('#liked');


var resultsList = $('#results');


// ...


resultsList.on('click', '.like', function (e) {
  e.preventDefault();
  var name = $(this).closest('li').find('h2').text();
  liked.find( '.no-results' ).remove();
  $('<li>', { text: name }).appendTo(liked);
});
```

Untestable

```
var Likes = function (el) {
  this.el = $(el);
  return this;
};

Likes.prototype.add = function (name) {
  this.el.find('.no-results').remove();
  $('<li>', { text:
name }).appendTo(this.el);
};
```

Testable

Sample Solution - Search Results

```
var SearchResults = function (el) {
  this.el = $(el);
  this.el.on( 'click', '.btn.like', _.bind(this._handleClick, this) );
};
SearchResults.prototype.setResults = function (results) {
  var templateRequest = $.get('people-detailed.tmpl');
  templateRequest.then( _.bind(this._populate, this, results) );
};
SearchResults.prototype._handleClick = function (evt) {
  var name = $(evt.target).closest('li.result').attr('data-name');
  $(document).trigger('like', [ name ]);
};
SearchResults.prototype._populate = function (results, tmpl) {
  var html = _.template(tmpl, { people: results });
  this.el.html(html);
};
```

Testable

Sample Final Solution

```
$(function() {
  var pending = false;
  var searchForm = new SearchForm('#searchForm');
  var searchResults = new SearchResults('#results');
  var likes = new Likes('#liked');
  var searchData = new SearchData();
  $(document).on('search', function (event, query) {
    if (pending) { return; }
    pending = true;
    searchData.fetch(query).then(function (results) {
      searchResults.setResults(results);
      pending = false;
    });
    searchResults.pending();
  });
```

```
<cont>
  $(document).on('like', function (evt, name) {
    likes.add(name);
  });
});
```

Advanced problems

- What to do about randomness?
- Network?
- Databases?
- Other challenges?