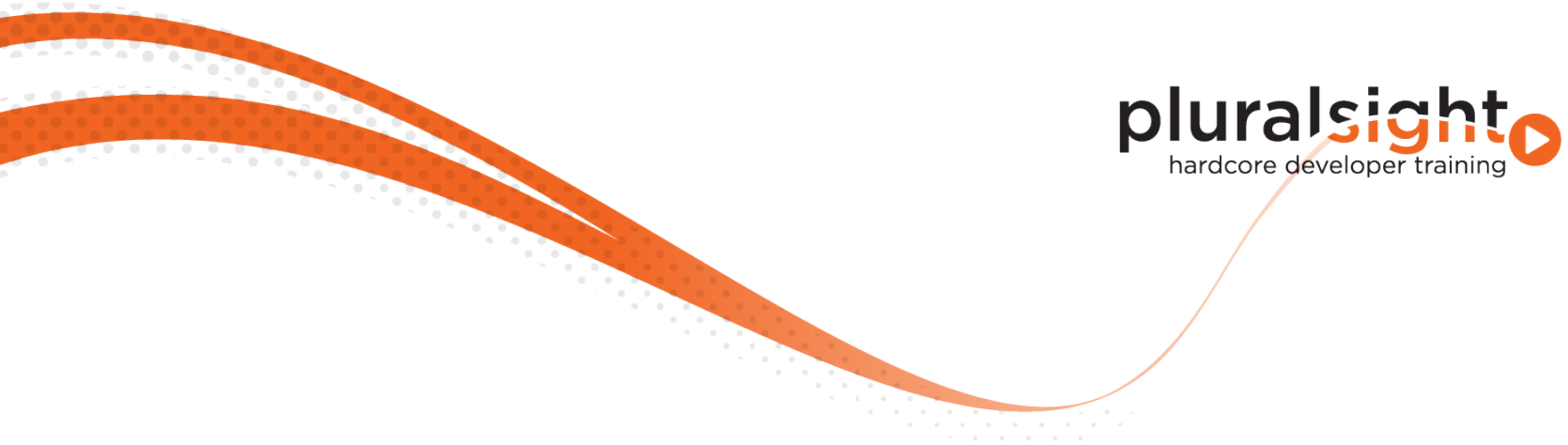
Designing RESTful Services

Howard Dierking

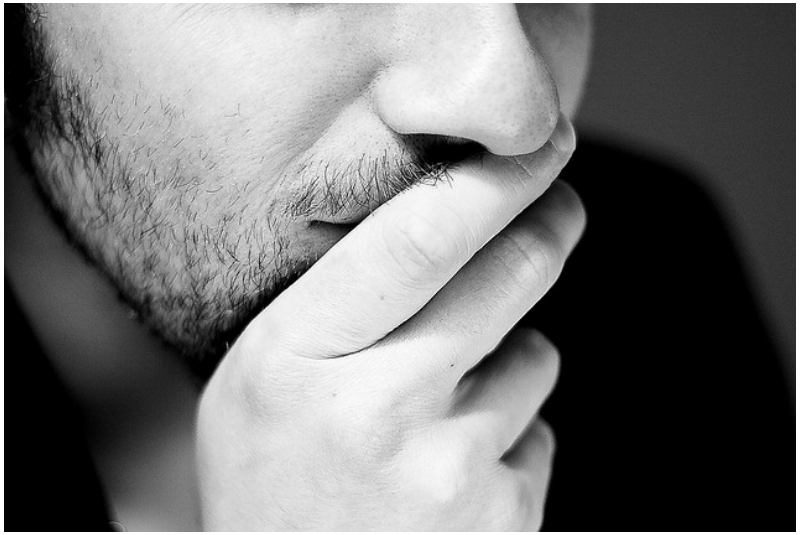
http://codebetter.com/howarddierking



Overview

- Thinking about RESTful System Design
- A Real World REST Example
- Designing the Bug Tracking Service
- State transitions and the uniform interface
- Establishing the “Contract”
- Evolving the Service

RESTful Thinking



- Developers approach REST with the metaphors and abstractions used to solve other programming problems
- While these are orthogonal to REST, they can bias a design towards RPC

To create a RESTful design, we need to start with a new metaphor...

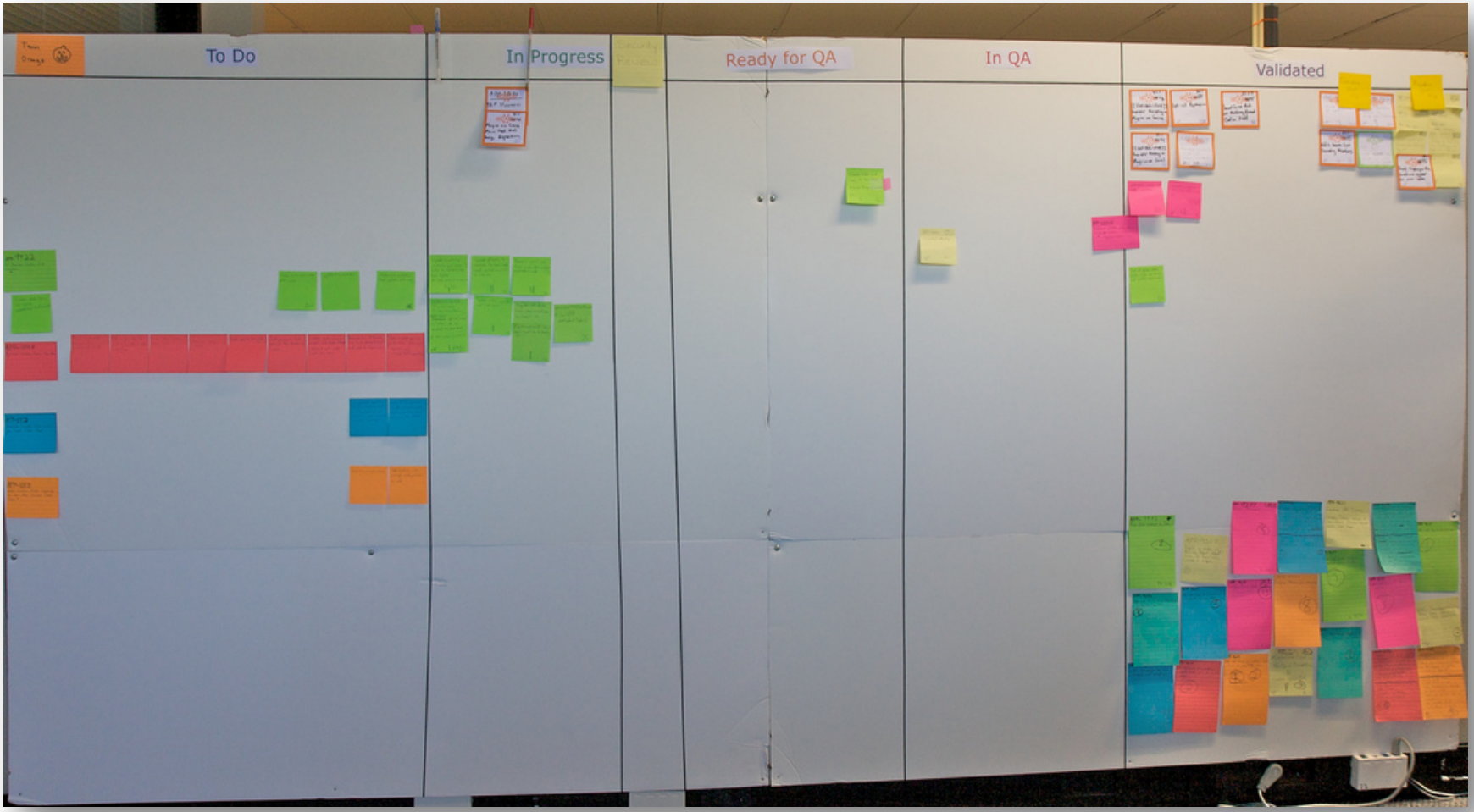
The Metaphor

**Imagine that you have no computers –
only paper and desk paper trays.**

How would you organize these to facilitate a workflow?

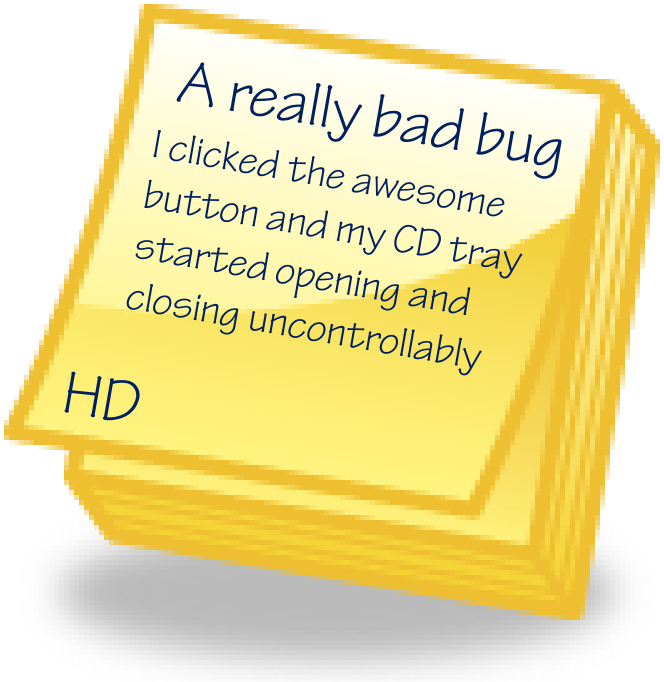
A Real World REST Example

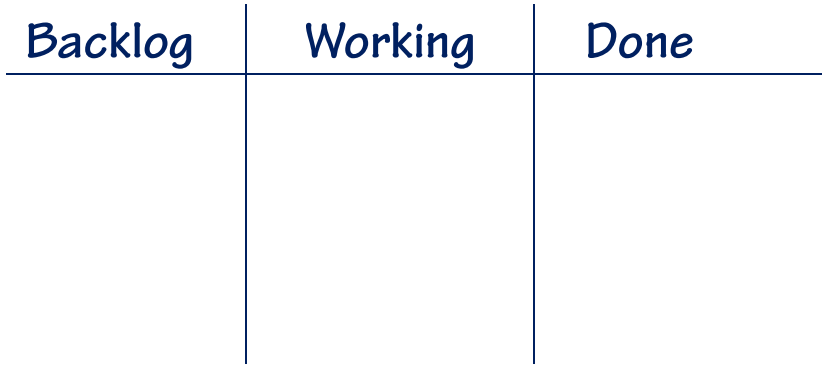
Looking at the physical world shows examples of REST all around us



A Bug Tracking Workflow

- What does the sticky note look like?
- What does the board look like?





Getting Rid of Implicit Understanding

Backlog	Working	Done

Changing the Workflow

Backlog	Working	QA	Done

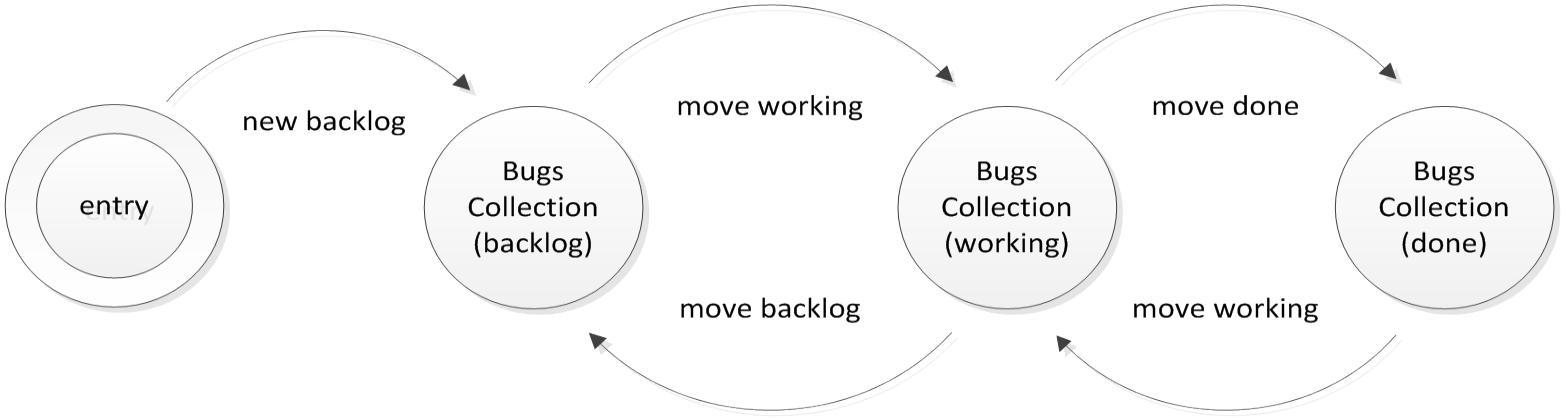
Designing the Bug Tracking Service

- List the requirements
- Identify the state transitions
- Identify the resources
- Design the media type

Bug Tracking Service Requirements

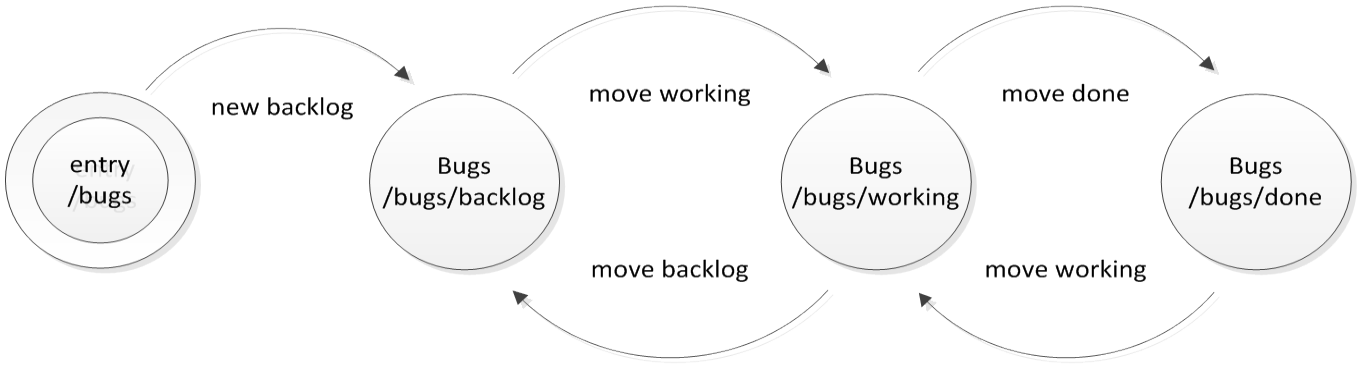
- **Discover bugs in various workflow states**
  - Backlog
  - Working
  - Done
- **Add a new bug to the backlog**
- **Activate a bug**
- **Complete a bug**

Application State Transitions



- Client should be able to transition to all of these states through starting state resource identifier
- Navigation and error transitions are not shown

Identify the Resources



- **/bugs – entry point**
  - GET: fetch hypermedia elements to navigate and to add a new bug to the backlog
- **/bugs/backlog – bugs that have not yet been activated**
  - GET: fetch the list of bugs in the backlog
  - POST: add a new bug to the backlog
- **/bugs/working – bugs that are being actively worked on**
  - GET: fetch the list of bugs being worked on
  - POST: activate a bug
- **/bugs/done – bugs that are finished**
  - GET: fetch the list of bugs that are done
  - POST: complete a bug

Design the Representation

- **Base format**
  - Can be anything (XML, JSON, HTML, etc.)
- **State transfer**
  - Read only – client does not transfer data to servers
  - Predefined – transfer bodies defined in the media type documentation that clients learn to use
  - Ad-hoc – details about valid transfer elements are sent to the client in the representation
- **Domain style**
  - Specific – type is tightly bound to the business domain (custom schema)
  - General – type is bound to a general domain, such as invoices or lists (ATOM)
  - Agnostic – type is unrelated to a specific domain (HTML)
- **Application Flow**
  - None – client does not have any flow identifiers
  - Intrinsic – identifiers are built into the media type design
  - Applied – identifiers are applied using decorators (HTML rel and class)

Bug Tracking Service Representation

- Base format = HTML
- State transfer = Ad-hoc
- Domain style = Agnostic
- Application Flow = Applied

Need Elements For

- List of bugs
- Link template for moving a bug to backlog
- Link template for moving a bug to working
- Link template for moving a bug to done
- Link template for adding a bug
- Navigation links

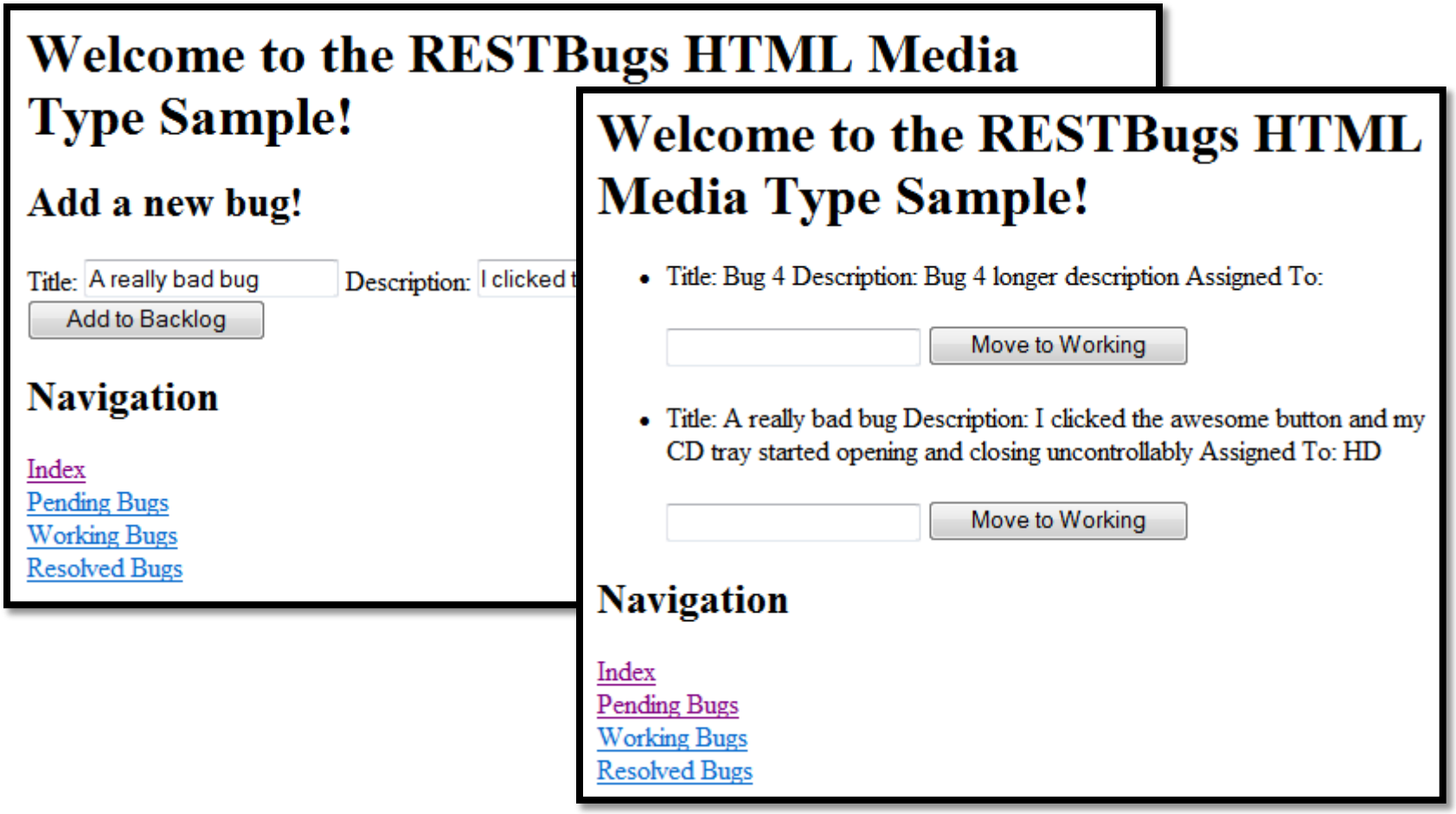
Mapping to Base Format

Attribute	Value	Applied To	Definition
id	bugs	DIV	Container for bugs state elements
name	id	INPUT[hidden]	The bug identity; found as a child of FORM.move
	title	INPUT[text]	Bug title state transfer element; found as a child of FORM.new
class	all	UL	List of bugs; can be a single bug
	title	SPAN	Bug title; found as a child of LI
	description	SPAN	Bug description; found as a child of LI
	new backlog	FORM	Application flow identifier for adding a new bug to the backlog
	next	FORM	Hints to client the ideal next state transition; should be used with FORM.move class
rel	index	A	Navigate to the index resource; should not be more than 1
	backlog	A	Navigate to the backlog resource; should not be more than 1

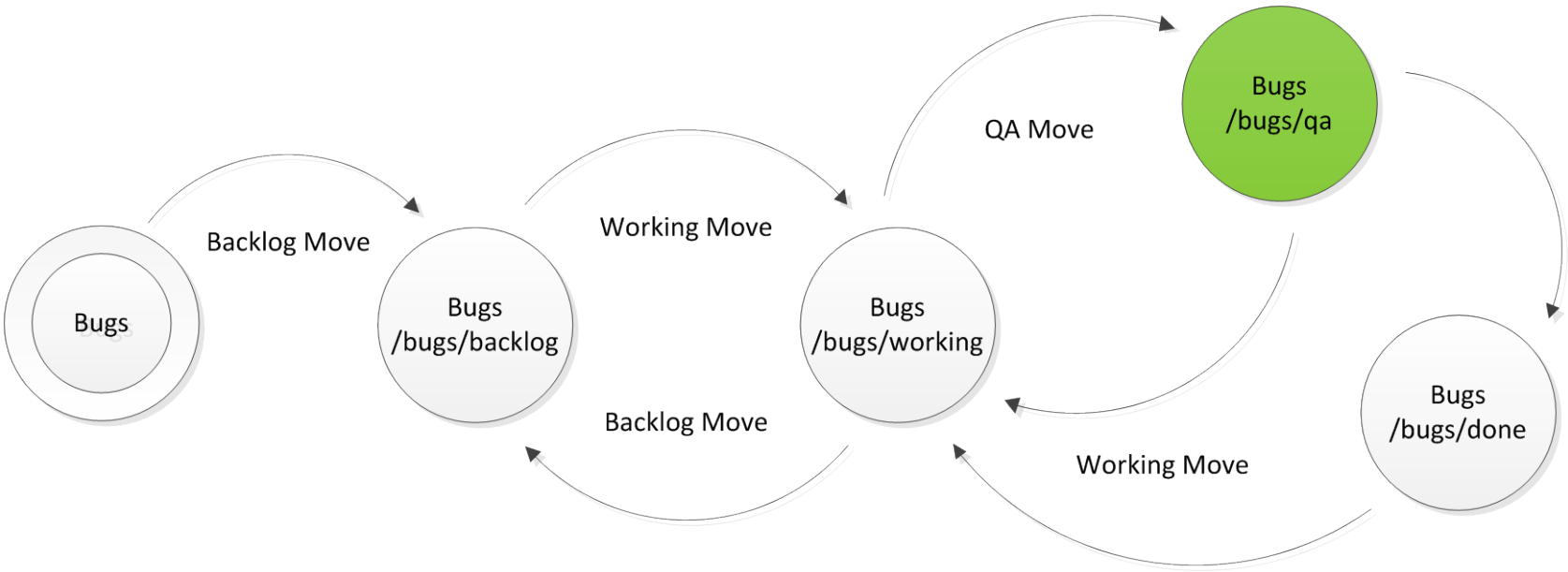
Sample Markup

```
<html>
   <div id="bugs">
      <ul class="all">
         <li>
            <span class="title"></span>
            <span class="description"></span>
            <span class="assigned-to"></span>
            <form class="move active next"
             action="..." method="POST">
               ...
            </form>
         </li>
      </ul>
   </div>
   <div id="links>
      <a rel="index" href="...">Index</a>
   </div>
   <div id="forms">
      <form class="new backlog" action="..." method="POST">
         ...
      </form>
   </div>
</html>
```

“Surfing” Your API



Dynamically Modify the Workflow



- Create the QA resource
- Add new FORM.class and A.rel values “qa” (SHOULD understand)
- Modify link generation logic

Modified Representation of “Working” Resource

```
<li>
  Title: <span class="title">A really bad bug</span>
  Description: <span class="description">I clicked the awesome
   button and my CD tray started opening and closing
   uncontrollably</span>
  Assigned To: <span class="assigned-to">HD</span>
  <form class="move backlog" action="/bugs/backlog" method="POST">
    <input name="id" type="hidden" value="6"/>
    <input name="comments" type="text" value=""/>
    <input name="submit" type="submit" value="Move to Backlog"/>
  </form>
  <form class="move qa next" action="/bugs/qa" method="POST">
    <input name="id" type="hidden" value="6"/>
    <input name="comments" type="text" value=""/>
    <input name="submit" type="submit" value="Move to QA"/>
  </form>
</li>
```

Versioning

- **With Web services, the unit of versioning is the contract==service description ==URI**
  - Versioned service URIs look like: http://localhost/services/v2/my.svc
- **With REST, the contract is the uniform interface, enabling:**
  - Versioning within the representation
  - Versioning the representation
  - Versioning the resource

Summary

- Thinking about RESTful System Design
- A Real World REST Example
- Designing the Bug Tracking Service
- Establishing the “Contract”
- Mapping the Domain to the Uniform Interface
- Evolving the Service

- **HTTP/1.1 Specification**
  - http://www.w3.org/Protocols/rfc2616/rfc2616.html
- **Fielding, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.**
- **Amundsen, Mike. Building Hypermedia APIs with HTML5 and Node. O'Reilly Media, 2011.**