goto; conference aarhus

USABILITY LESSONS FOR APIS

Ian Cooper
Huddle

Who are you?

- Software Developer for 20 years
  - Worked mainly for Software Companies
    - Reuters, SunGard, Misys, Huddle
  - Worked for a couple of MIS departments
    - DTI, Beazley
- Microsoft MVP for C#
  - Interested in OO design
  - Interested in Agile methodologies and practices
- No smart guys
  - Just the guys in this room

huddle™

Huddle is an online collaboration app for businesses, enabling you to manage projects, files and people in the cloud

Agenda

- Why should you care?
- Personas, Goals, and Scenarios
- Documentation Driven Design
- Cognitive Dimensions
- Usability Testing

The Importance of Design

WHY SHOULD YOU CARE?

Design Matters!

Project Origami

The iPad

What happens when we develop for end users

Vision
Feature Sets
Personas
Goals
Scenarios
Prototyping
Stories
Scheduling
Specifications
UI Design
UI Code
Usability Testing
Further Code
Performance Testing
Acceptance Testing
Packaging
Documentation
Training
Support
A/B Testing

What happens when we design for developers

Vision

Feature Sets

Personas

Goals

Stories

Scheduling

Scenarios

Prototyping

UI Design

UI Code

Further Code

Performance Testing

Acceptance Testing

Packaging

Documentation

Training

Support

A Contract with the World

Upstream Teams

People are dependent on them, but they don't depend on anyone else.

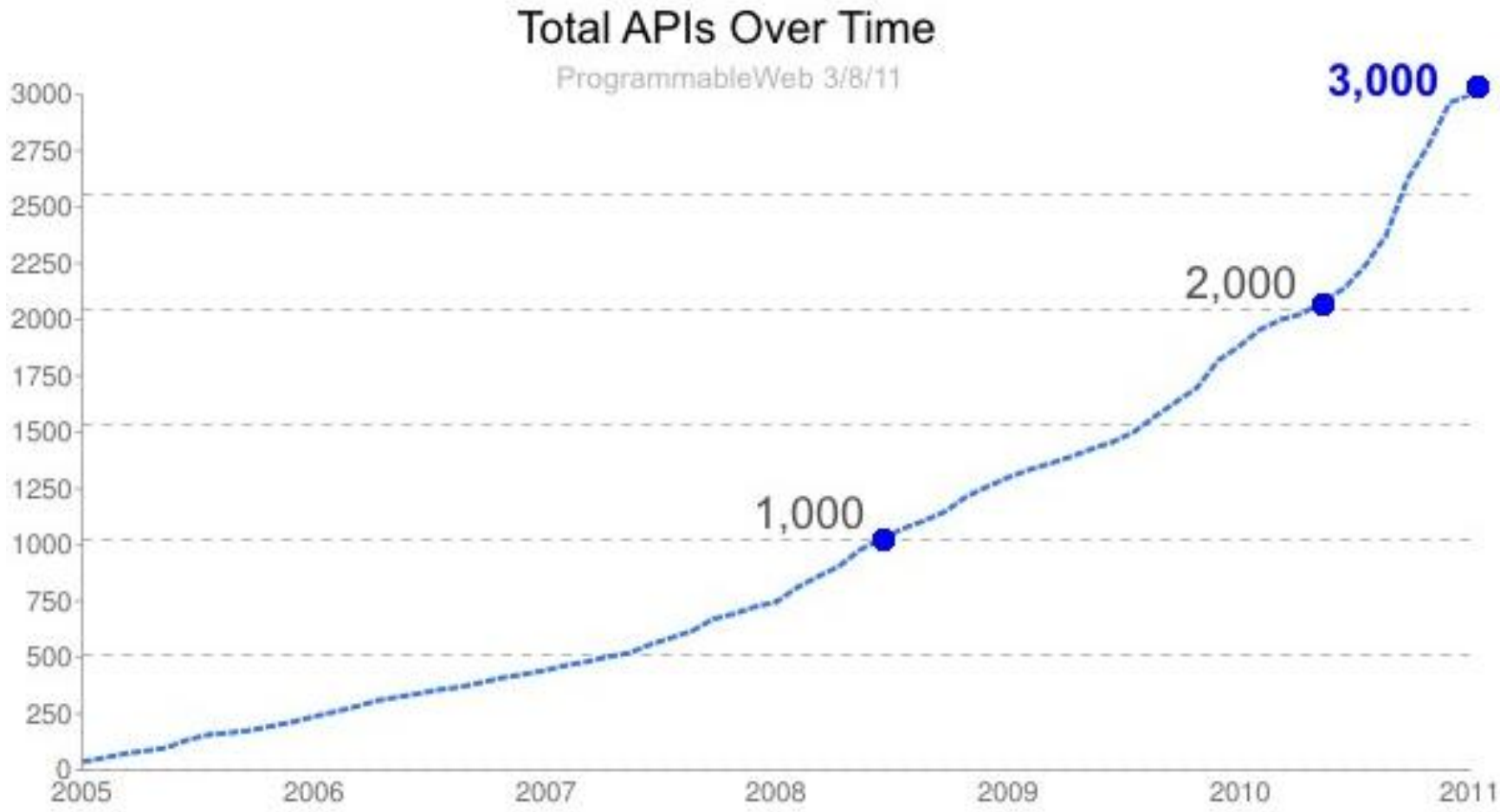
Downstream Teams

They are dependent on an upstream team, they may or may not have folks dependent on them.

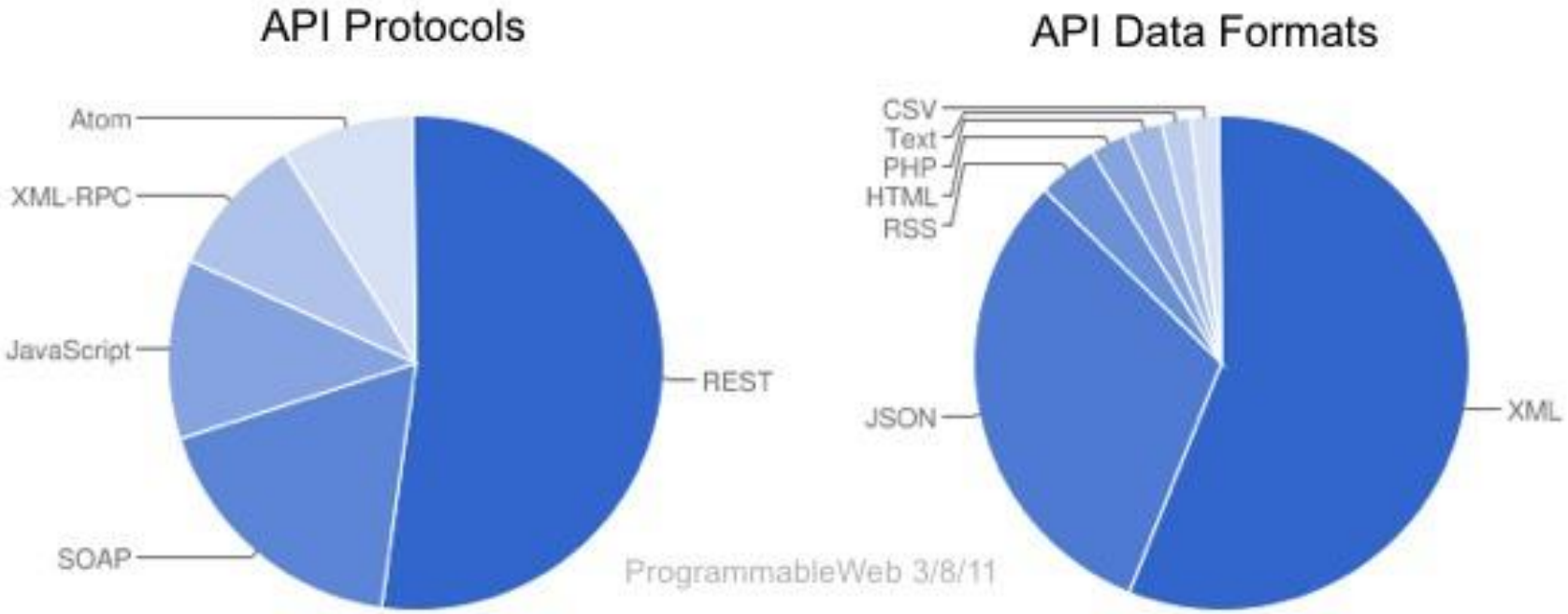
An upstream team has no reason not to pollute the river, forcing the downstream team to drink their pollution.

An API is a contract: it says we won't pollute the river, we will stick to these environmental regulations, and you have comeback if we don't

The Importance of APIs



On the Web, REST dominates



Learning from the designers

PERSONAS, GOALS, SCENARIOS

Use Personas

Personas are archetypal users. They ‘stand-in’ for real users and help guide our decisions

Persons identify user motivations, expectations and goals that drive behavior

The more specific we make our personas, the more effective they are as design tools. That's because we reduce the ‘debate’ around a personas goals as they become more specific.

John is 35 years old and married with a 5 year old son, Joshua. His wife works as a PA, for his last company where they met. This is John's third job since leaving university, where he did a degree in Mechanical Engineering. His first job after leaving university was as an Access and Visual Basic programmer, for a small software house building accounting solutions. He moved from there after 5 years to work at a pharmaceutical company as a Visual Basic programmer building internal MIS solutions. John was disappointed with MS for releasing .NET, because ihe liked working in VB6 and did not want to learn VB.NET.. Recently he has become worried that Microsoft is eroding his skills again with announcements ending Silverlight, which he codes in day-to-day, for Windows 8.

John does not attend conferences, or user groups; he only rarely reads blogs and then it is always MSDN blogs. He did attend Tech Ed 2005. He is definitely not Alt.NET, and never uses TDD. He thinks of that group as 'ivory-tower' technologists who don't focus on delivering to users. Mostly this is fear - fear that he might have to give up evenings or weekends to improve his skills. There is also a fear of appearing stupid by admitting there are things he does not know, that are worth knowing. He used to be on Experts Exchange and Code Project but now gets most of his technical help from Stack Exchange. John is knowledgeable about SQL server. His preferred development approach is to design the schema and stored procedures to access it, expose that through a web service and then write a Silverlight UI to call that.

If anything gives John pride its getting work done quickly. His users love his 'can do' attitude, his project managers the speed with which he delivers to the requirements. John tries to write as little code as possible, code generation is his favourite productivity secret, and he has authored his own CodeSmith and T4 templates to generate stored procedures for data access. His templates are used throughout the team.

John is not a web developer. Most of his experience has been client-server. He can't understand why anyone would write in JavaScript when they can code in C# using Silverlight. He has written SOAP web services, which were simple wrappers to get data out of a Database. He has used WCF, but the configuration file is just voodoo to him, and he makes it work by trial and error. He has never heard of REST. He has also written some SharePoint and Dynamics CRM code before.

Recently a lot of his work has been integration projects with third-party products. These all used SOAP APIs and John used the Visual Studio Wizards to generate the proxies and then called the external APIs.

How can les than a dozen personas represent the user base?

Traditional techniques, asking a broad cross-section of the user base generates a lot of noise, and a lot less signal.

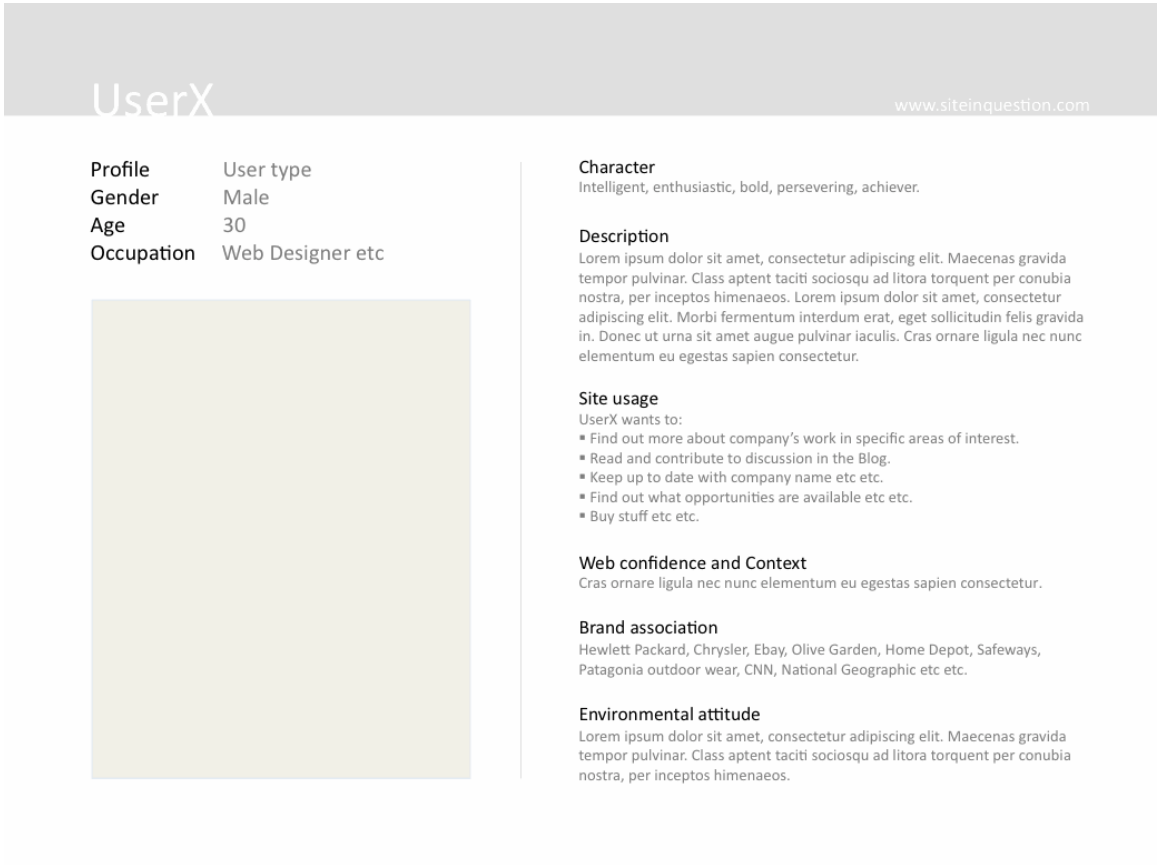
PERSONAS IMPROVE THE SIGNAL TO NOISE RATIO

It created designs that try to be everything to everyone. Trying to please everyone yet pleasing no one.

If you react to users you become a service company not a product company

Your product begins to mutate from one release to another, not follow a vision

Personas cut through this to represent the user base with archetypical types focused around the goals of a similar users



Finding Personas

You are looking to build a cast of characters

How Many Personas?

Build a cast of characters

Every cast has at least one primary persona

The primary persona is the person who must be satisfied.

Each primary persona implies a separate interface

Goals

Personas are defined by their goals; goals are defined by personas

Developers, by training, tend to approach design by asking: “What are the tasks?”

We want them to ask: “What are the goals?”

Only some task sequences will satisfy the user’s goals

Personal, Corporate, Practical

Personal Goals

Not feel stupid; Not make mistakes; Get an adequate amount of work done; Have fun (or at least not be too bored)

Corporate Goals

Increase Revenue; Protect Revenue; Reduce Costs

Practical Goals

Avoid meetings; Handle the client's demands; Record the client's order; Create a numerical model of the business

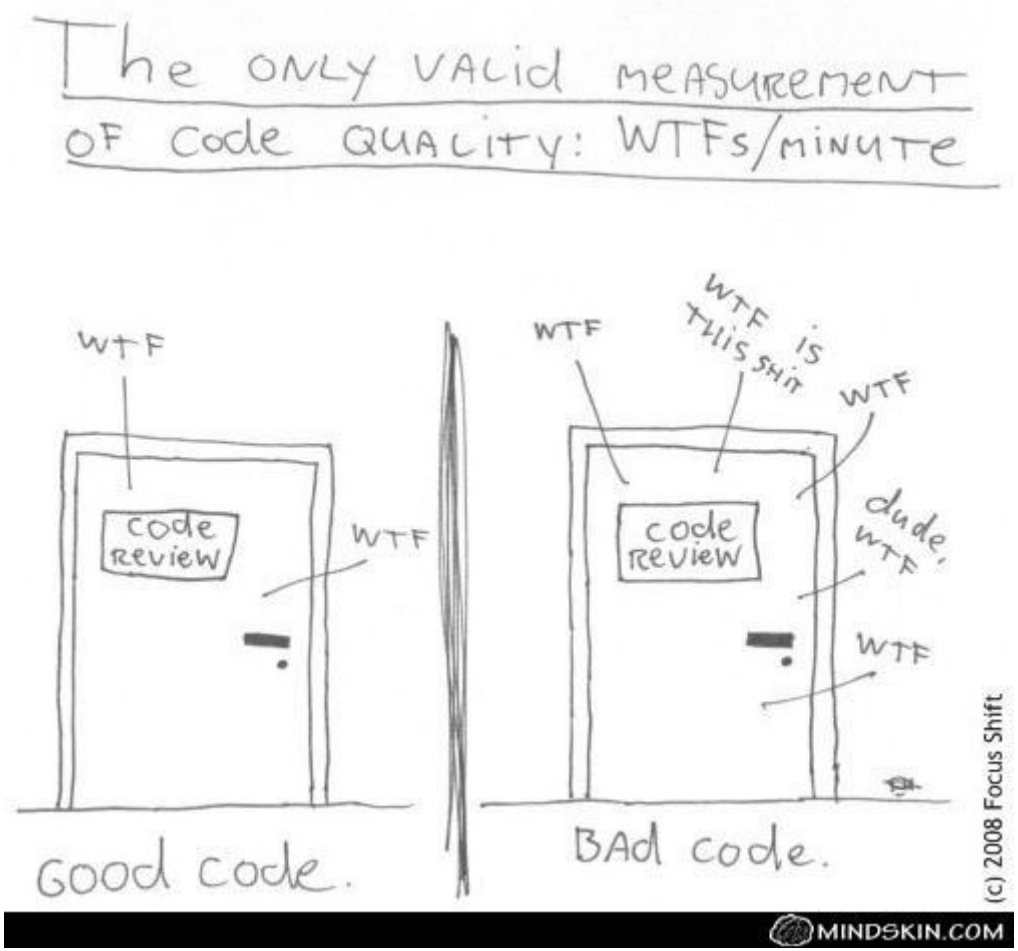
Scenarios

A scenario is a concise description of a persona trying to achieve a goal

Pretotyping for your API

DOCUMENTATION DRIVEN DESIGN

Quality measured by WTFs/minute



Write Code

Write Documentation

Re-write Code

Why does this work?

Programmers suffer from (and succeed because of):

Laziness, ineptitude, hubris

Programmers are task-focused

Culture of breaking problems down, solve parts

This results in functional, but not useable, code

Write Documentation First

Write the documentation for the API you want to build

Produce real documentation and ask for feedback

Find about what we should build, not how we will build it

Don’t’ document code, code to documentation

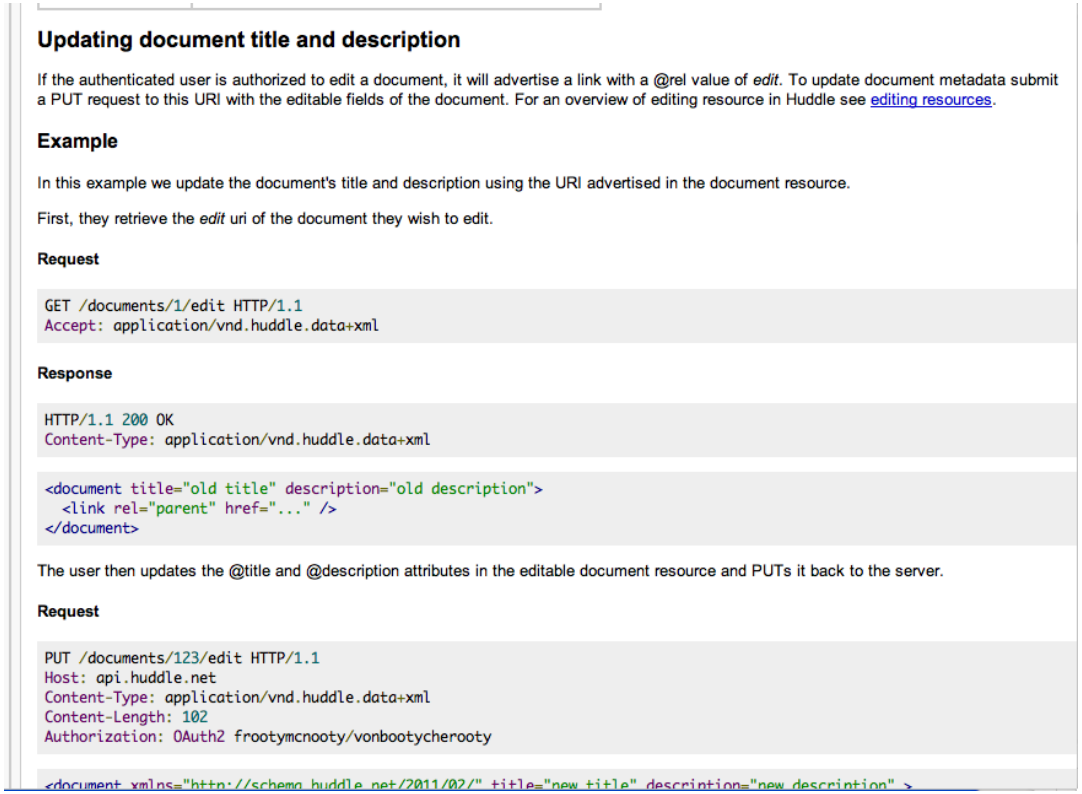
Pretotyping *[pree-tuh-tahy-ping], verb:* Testing the initial appeal *and* actual usage of a potential new product by simulating its core experience with the smallest possible investment of time and money.

Less formally, pretotyping is a way to test a product idea quickly and inexpensively by creating extremely simplified versions of that product to help validate the premise that *"If we build it, they will use it.”*

www.pretotyping.org

The Stairway to Heaven

1. Pull feature from backlog
2. What goals do personas have?
3. Break out scenarios.
4. Write initial piece of documentation
5. Review
6. Repeat



Updating document title and description

If the authenticated user is authorized to edit a document, it will advertise a link with a @rel value of *edit*. To update document metadata submit a PUT request to this URI with the editable fields of the document. For an overview of editing resource in Huddle see editing resources.

Example

In this example we update the document's title and description using the URI advertised in the document resource.

First, they retrieve the *edit* uri of the document they wish to edit.

Request

```
GET /documents/1/edit HTTP/1.1
Accept: application/vnd.huddle.data+xml
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/vnd.huddle.data+xml
```

```
<document title="old title" description="old description">
  <link rel="parent" href="..." />
</document>
```

The user then updates the @title and @description attributes in the editable document resource and PUTs it back to the server.

Request

```
PUT /documents/123/edit HTTP/1.1
Host: api.huddle.net
Content-Type: application/vnd.huddle.data+xml
Content-Length: 102
Authorization: OAuth2 frootymcnooty/vonbootycherooty
```

```
<document xmlns="http://schema.huddle.net/2011/02/" title="new title" description="new description" >
```

Using Cognitive Dimensions

MEASURING API QUALITY

The Cognitive Dimensions

1. Abstraction level.
2. Learning style.
3. Working framework.
4. Work-step unit.
5. Progressive evaluation..
6. Premature commitment.
7. Penetrability.
8. API elaboration.
9. API viscosity.
10. Consistency.
11. Role expressiveness.
12. Domain correspondence.

Role Expressiveness

```
GET ?prefix=photos/2006/&delimiter=/ HTTP/1.1
Host: example-bucket.s3.amazonaws.com
Date: Wed, 01 Mar  2009 12:00:00 GMT
Authorization: AWS 15B4D3461F177624206A:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

In response, Amazon S3 returns only the keys that start with the specified prefix. Further, it uses the *delimiter* character to group keys that contain the same substring until the first occurrence of the *delimiter* character after the specified prefix. For each such key group Amazon S3 returns one <CommonPrefixes> element in the response. The keys grouped under this *CommonPrefixes* element are not returned elsewhere in the response. The value returned in the CommonPrefixes element is a substring from the beginning of the key to the first occurrence of the specified delimiter after the prefix.

```
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Name>example-bucket</Name>
  <Prefix>photos/2006/</Prefix>
  <Marker></Marker>
  <MaxKeys>1000</MaxKeys>
  <Delimiter>/</Delimiter>
  <IsTruncated>false</IsTruncated>

  <CommonPrefixes>
    <Prefix>photos/2006/feb/</Prefix>
  </CommonPrefixes>
  <CommonPrefixes>
    <Prefix>photos/2006/jan/</Prefix>
```

Domain Correspondence

Request

```
GET /folders/12345 HTTP/1.1
Accept: application.vnd.huddle.data+xml
Authorization: OAuth2 frootymcnooty/vonbootycherooty
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/vnd.huddle.data+xml
```

```
<folder xmlns="https://schema.huddle.net/2011/02/" title="My folder" description="Folder description">

  <link rel="self" href="/folders/12345" />
  <link rel="parent" href="..." />
  <link rel="delete" href="..." />
  <link rel="edit" href="..." />
  <link rel="create-folder" href="..." />
  <link rel="create-document" href="..." />

  <actor name="Ian Cooper" rel="owner">
    <link rel="self" href="..." />
    <link rel="avatar" type="image/jpg" href="..." />
    <link rel="alternate" href="..." type="text/html" />
  </actor>

  <folders>
    <folder title="sub folder" description="my subfolder">
        <link rel="self" href="..." />
        <link rel="parent" href="..." />
        <link rel="create-folder" href="..." />
        <link rel="create-document" href="..." />
        <link rel="delete" href="..." />
    </folder>
  </folders>

  <documents>
    <document title="document title" description="document description">
      <link rel="self" href="http://api.huddle.net/documents/83847" />
      <link rel="parent" href="http://api.huddle.net/folders/45647" />
```

Consistency

Create a lock

When retrieving a document, the response will advertise a link with a @rel value of *lock*, which you can use to POST the new lock to the document.

Example

In this example a document is locked.

Request

```
POST /documents/123/lock HTTP/1.1
Content-Type: application/vnd.huddle.data+xml
Authorization: OAuth2 frootymcnooty/vonbootycherooty
```

Response

If successful, this operation returns a 201 Created with a representation of the lock. The response will contain a *delete* link which can be used to unlock the document.

If the document is already locked, we will return a 409 Conflict with a Location header giving the URI of the locked document.

This response uses the standard error codes and returns standard response headers.

```
HTTP/1.1 201 Created
Content-Type: application/vnd.huddle.data+xml
```

```
<lock xmlns="http://schema.huddle.net/2011/02/">
  <link rel="self" href="/documents/123/lock" />
  <link rel="delete" href="..." />
  <link rel="parent" href="..." />

  <actor name="Peter Gibson" rel="owner">
    <link rel="self" href="..." />
    <link rel="avatar" href="..." type="image/jpg" />
    <link rel="alternate" href="..." type="text/html" />
  </actor>
</lock>
```

Request

```
GET /folders/12345/changes?since=2011-02-14T13:17:42Z HTTP/1.1
Accept: application.vnd.huddle.data+xml
Authorization: OAuth2 frootymcnooty/vonbootycherooty
```

Response

This response uses the standard error codes and returns standard response headers.

```
HTTP/1.1 200 OK
Content-Type: application.vnd.huddle.data+xml
```

```
<changes xmlns="http://schema.huddle.net/2011/02/">
  <link rel="self" href="/folders/12345/changes?since=2011-02-14T13:17:42" />

  <change>
    <link rel="subject" href="/documents/123" />
    <type>DocumentLocked</type>
    <actor name="Mrs. Teasdale" rel="owner">
        <link rel="self" href="..." />
        <link rel="avatar" href="..." type="image/jpg" />
        <link rel="alternate" href="..." type="text/html" />
    </actor>
    <created>2011-02-01T13:18:42Z</created>
  </change>

  <change>
    <link rel="subject" href="/documents/345" />
    <type>DocumentCreated</type>
    <actor name="Rufus T. Firefly" rel="owner">
        <link rel="self" href="..." />
        <link rel="avatar" href="..." type="image/jpg" />
        <link rel="alternate" href="..." type="text/html" />
    </actor>
    <created>2011-02-01T13:17:42Z</created>
  </change>
</changes>
```

Learning Style

A link element represents an action that can be performed against a resource, or a related resource.

The URI for the action or relation is given by the @href property. The relationship between the linked resource and the current resource is given by the @rel property.

Optional @title, @type and @description properties may give further information about the linked resource.

The values of the @rel property will depend on the object which contains the link. See, for example, Actor#Link_Rel_Values

Example

```
<link href="http://example.org" rel="self" />
```

Properties

Name	Description	Usage
@href	A uri pointing to the linked resource	Required
@rel	A uri enumeration describing the relationship between this link and its containing resource	Required
@type	The content type of the linked resource	Optional
@title	The title of the linked resource	Optional
@description	The description of the linked resource	Optional

The @type attribute may refer either to

- the media-type that is expected in the body of request (eg. the @type attribute of a Folder#Syntax folder create link)

Working Framework

Response

```
HTTP/1.1 200 OK
Content-Type: application/vnd.huddle.data+xml
```

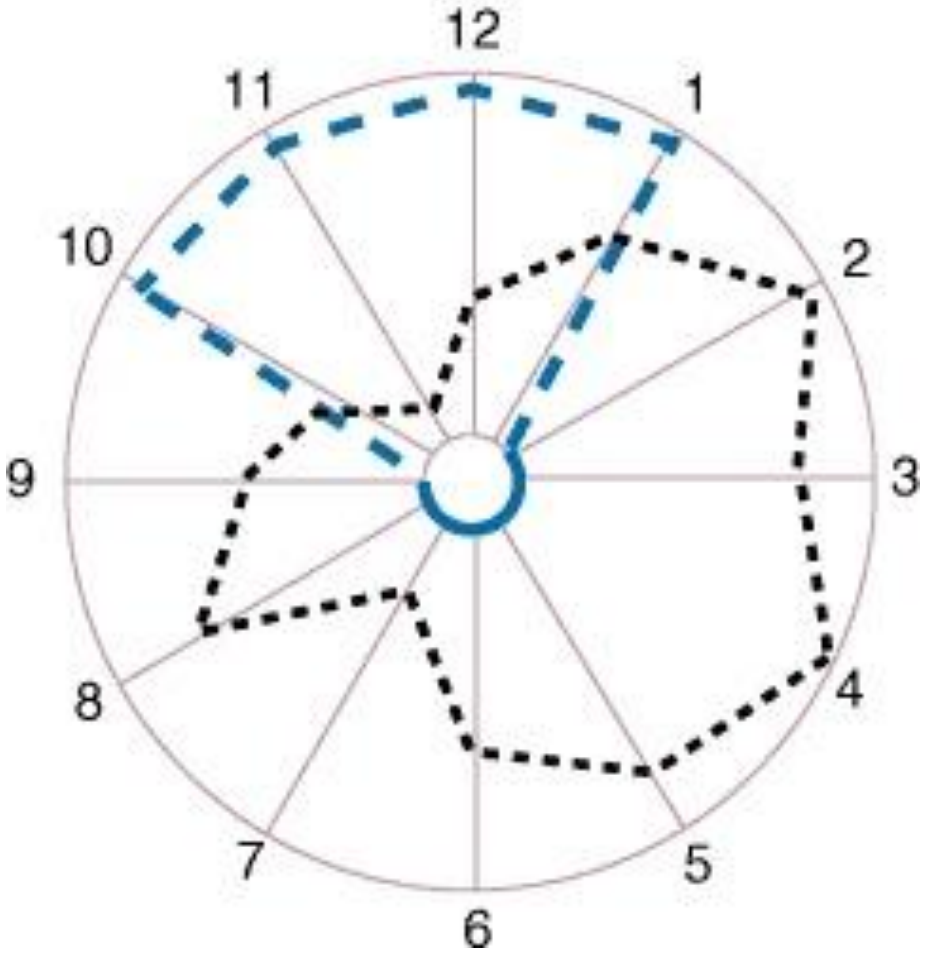
```
<folder xmlns="https://schema.huddle.net/2011/02/" title="My folder" description="Folder description">

  <link rel="self" href="/folders/12345" />
  <link rel="parent" href="..." />
  <link rel="delete" href="..." />
  <link rel="edit" href="..." />
  <link rel="create-folder" href="..." />
  <link rel="create-document" href="..." />

  <actor name="Ian Cooper" rel="owner">
    <link rel="self" href="..." />
    <link rel="avatar" type="image/jpg" href="..." />
    <link rel="alternate" href="..." type="text/html" />
  </actor>

  <folders>
    <folder title="sub folder" description="my subfolder">
        <link rel="self" href="..." />
        <link rel="parent" href="..." />
        <link rel="create-folder" href="..." />
        <link rel="create-document" href="..." />
        <link rel="delete" href="..." />
    </folder>
```

Personas and Cognitive Dimensions



USABILITY TESTING

Preparing for a Study

- Decide when
  - Enough of the API needs to be done
  - But you don't want to be so late you can't change
- Design the task list for the study
  - Think about the scenarios
  - Consider the personas – what assumptions will you prove?
- Work on the wording of the tasks in the lab
  - Guide them on what, avoid telling how

Running a Study

You don’t need an expensive lab

Record the interaction to share. You can do this by recording what the user does, using

- Camcorder
- Software (Camtasia, Hypercam)

Set up the tools on the machine. Use a VM to control and replicate environment. Remove orthogonal issues

Make the participant comfortable and put them at ease before you begin. Help them practice ‘working out loud’

Try to avoid guiding

Listen and take notes

Developer watching videotape of usability test.

Who do you test?

- Testing one user better than none
- Testing one user early better than 50 after
- Don’t worry about being representative

Dealing with the feedback

- Triage
  - Ignore ‘Kayak’ problems
  - Resist the temptation to ‘add’ something
  - ‘New feature’ requests may be preferences
  - Don’t thrown the baby out with the bath water
- Prioritize

Q&A

- @ICooper on Twitter
- ian@huddle.net
- http://codebetter.com/iancooper/