

The logo features the word "talis" in a white, lowercase, sans-serif font, centered within a black, glossy, tilted oval. A small red dot is positioned above the letter 'i'. To the right of the word is a registered trademark symbol (®). The background is a light gray gradient with faint, overlapping circular lines in shades of blue and red.

talis<sup>®</sup>

*“That's not what you said yesterday!”  
(Evolving your Web API)*

Ian Davis  
Chief Technology Officer, Talis

*What To Expect*

**Introduction**

**Compatibility**

**Techniques**

**Examples**

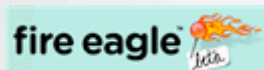
**Conclusions**

# Introduction

## *What is a Web API?*

a programmable interface to a computer system using Web protocols

as provided by a cast of thousands...

The Twitter logo, featuring the word "twitter" in a light blue, rounded, lowercase font with a white outline.The Bloglines logo, with the word "Bloglines" in a blue, sans-serif font.The Myspace.com logo, featuring a blue background with a white icon of three people and the text "myspace.com" and "a place for friends" below it.The last.fm logo, with the text "last.fm" in white on a red rectangular background.The Flickr logo, with the word "flickr" in blue and pink, and "BETA" in small letters above it.The Amazon Web Services logo, featuring a yellow cube icon and the text "amazon web services" in black.The Technorati logo, with a green speech bubble icon and the word "Technorati" in black.The Digg API logo, with the text "digg api" in white on a red rectangular background.The Tumblr logo, with the word "tumblr." in white on a black rectangular background.The Jaiku logo, with the word "jaiku" in white inside a green circle.The Fire Eagle logo, with the text "fire eagle" and a small orange eagle icon.The Google logo, with the word "Google" in its multi-colored font.The DOPPLR logo, with a horizontal bar of four colored squares (orange, grey, teal, blue) and the word "DOPPLR" in black.The Talis logo, with the word "talis" in white on a black oval background.

# *Kinds of Web API*

REST

SOAP

XML-RPC, POX, “GETsful”

Every URI is an “API”

## *Evolvability*

*“Because the components participating in a network-based application may be distributed across multiple organizational boundaries, the system must be prepared for gradual and fragmented change, where old and new implementations coexist, without preventing the new implementations from making use of their extended capabilities.”*

Roy Fielding

*Architectural Styles and the Design of Network-based Software Architectures.*

*Doctoral dissertation, University of California, Irvine, 2000.*

## *Why Evolve An API?*

**Fix bugs**

**Change behaviour**

**Change input/output formats**

**Remove functionality**

## *What's The Difficulty?*

Changes affect all users instantly

Often have client libraries

You often don't know all your users

You are a critical dependency

## *Considerations*

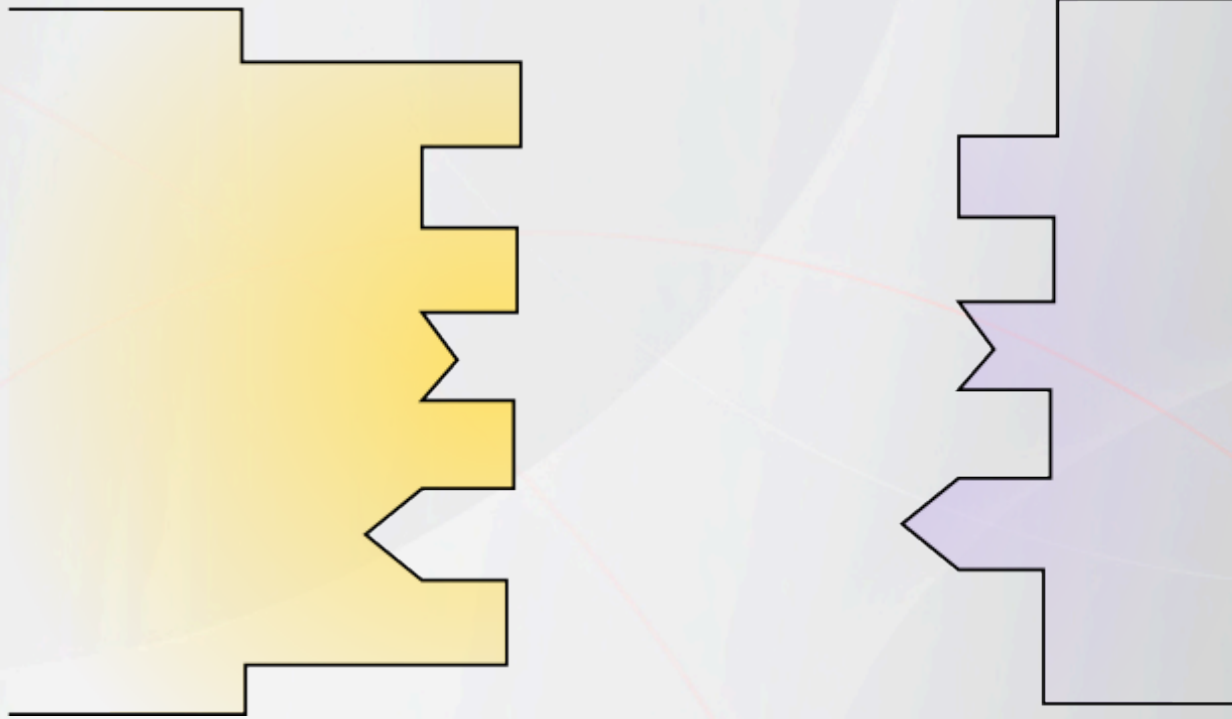
Your API is a contract with the rest of the world

Be cautious, consider whether you would support every addition for all time

Costs associated with maintaining compatibility

# Compatibility

# *Client and Service*



*Kinds of Compatibilty*

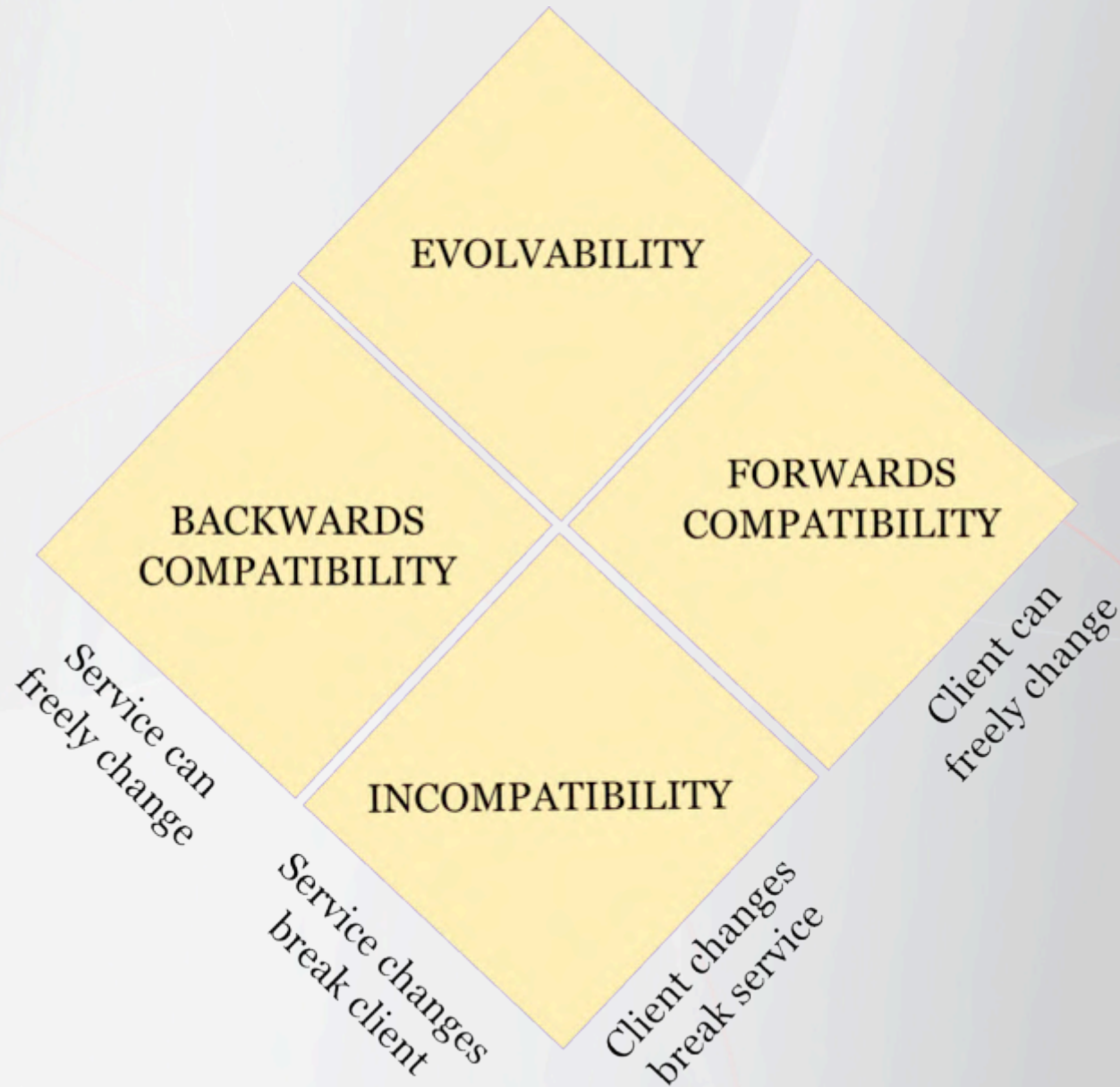
Incompatible

Backwards compatible

Forwards compatible

Evolvable

# *Coupling Between Client and Service*



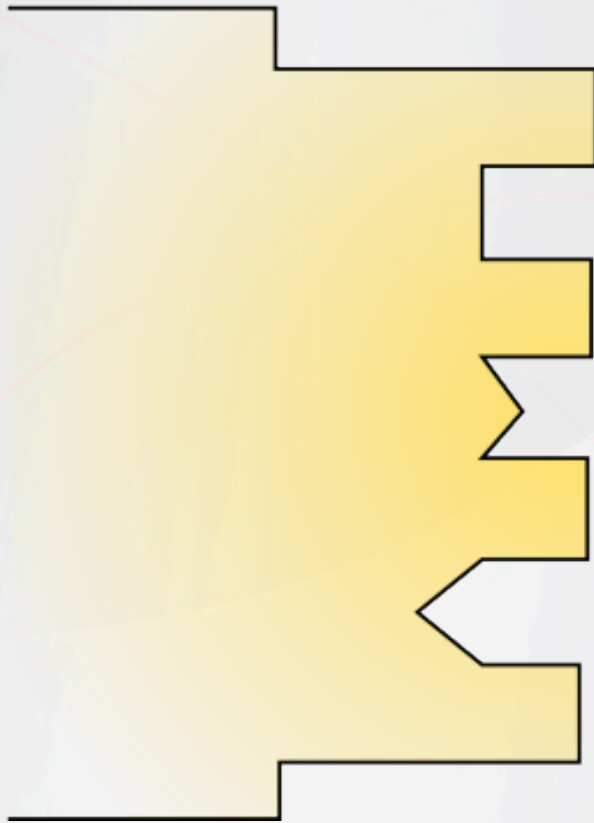
*Incompatible*

When service evolves, client  
is forced to change also

Client and service  
tightly coupled

Brittle

*Incompatible*

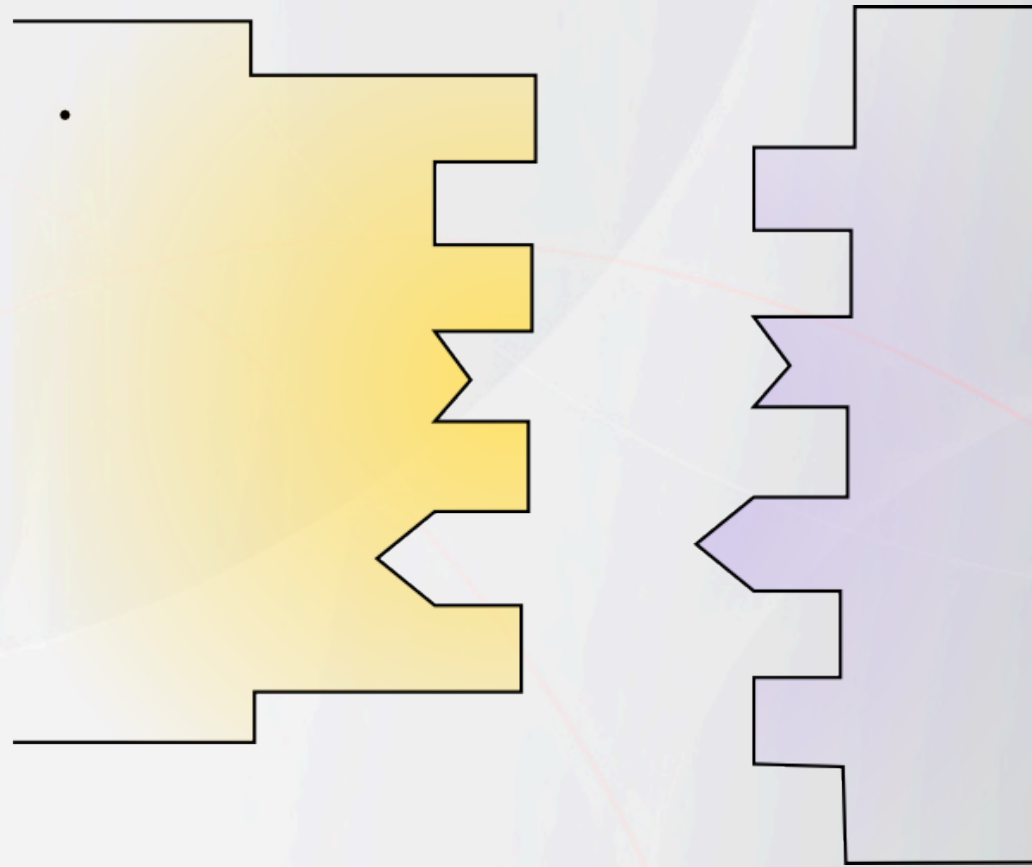


## *Backwards Compatibility*

Older clients understand  
interactions with newer services

Services can evolve without  
breaking clients

# *Backwards Compatibility*



## *Backwards Compatible Changes*

Adding a new independent service

Adding a new optional parameter

Adding a new optional format

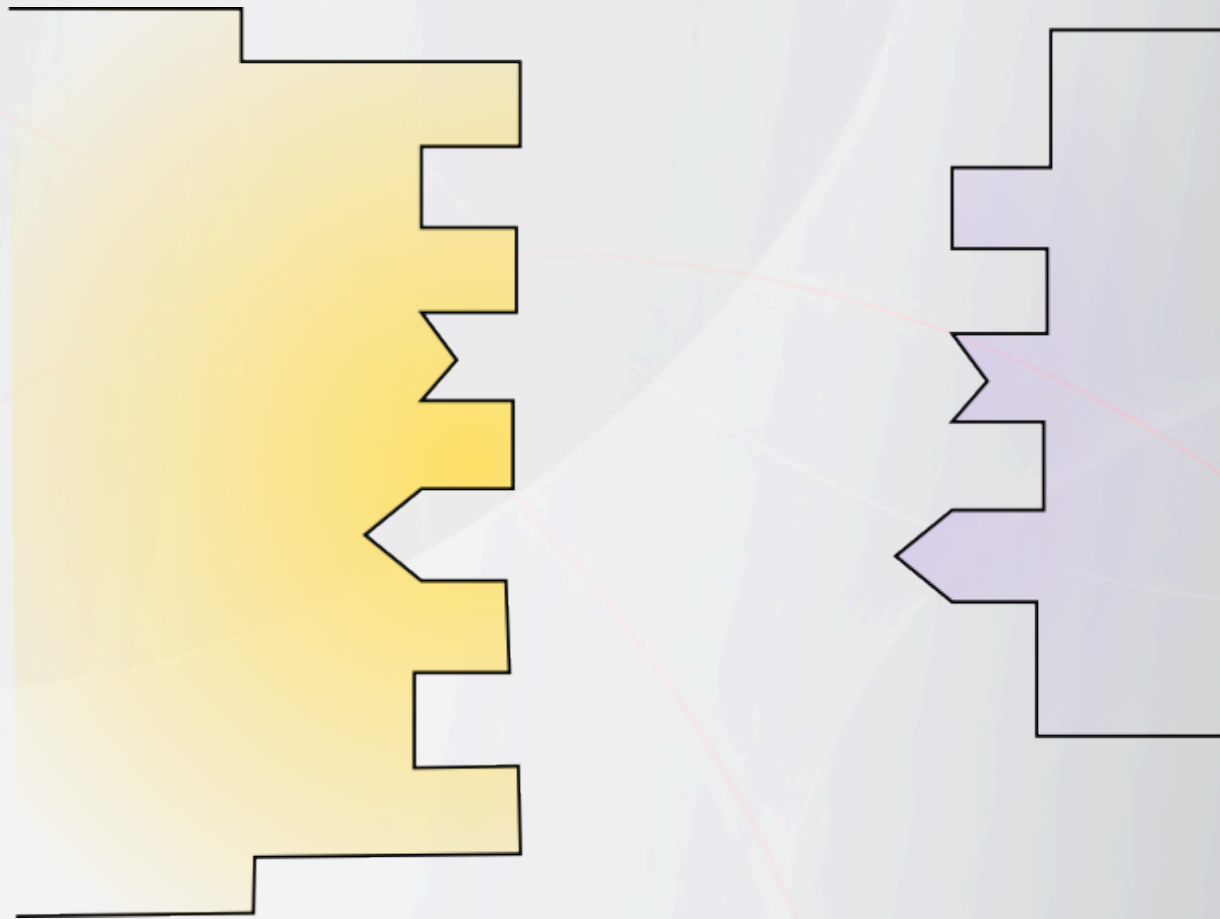
Non-behavioural bug fixing

## *Forwards Compatibility*

Older services understand  
interactions with newer clients

Services remain useful  
and relevant for longer

# *Forwards Compatibility*



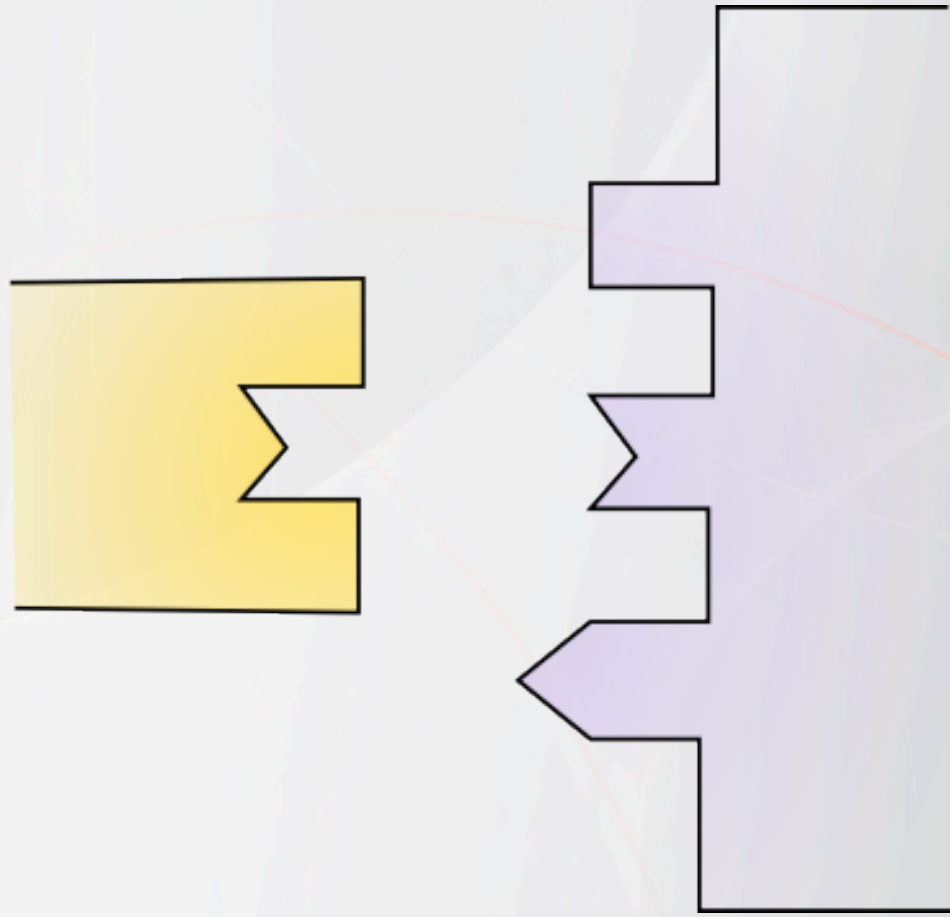
## *Evolvability*

Clients are tolerant of  
changes to services

Services are tolerant of  
changes to clients

Loose coupling

# *Clients Don't Consume The Whole API*



The background features a light blue and white gradient with several overlapping, semi-transparent circles in shades of blue and white. A thin, curved red line arches across the middle of the frame, passing behind the word 'Techniques'.

# Techniques

## *Must Ignore*

Principle is that service  
should ignore what it does  
not understand

Part of the robustness principle

Ignore means “do not process”  
not “do not respond”

Continue if you have enough information

## *Forwards Compatibility Via Must Ignore*

### Using query string

Ignore parameters you do not understand

Query string parameters should not change  
meaning of other parameters

## *Backwards Compatibility Via Media Types*

If data format changes between versions

Use a new media type

Still accept old format but client can indicate  
new behaviour via media type

## *Use HTTP Responses*

Redirect improved services with 301

Deprecate services with 410

Deprecate data formats with 406

*Use HTTP Redirection*

HTTP Redirects from  
old service URI to new

Service can rewrite entire request to new  
format and issue as a permanent redirect

Trickier with POST

## *Being Strict*

Don't silently ignore bad input

Give appropriate errors

So clients don't rely on things that aren't  
public

*WSDL and Service Description Languages*

Don't actually help with evolvability

May hinder it

They make it easier to consume the service  
but at a cost of static dependencies

# Examples

# *Examples of API Evolution*

Amazon

Yahoo

Google

del.icio.us

Talis

Namespaces

## *Amazon*

April 2008, Amazon introduced versioning of APIs

Requests now require version parameter  
(`&Version=2007-01-17`)

It's Optional – omitting it means use latest schema and API

*del.icio.us*

May 2006 changed from

<http://del.icio.us/api/posts/get>

To

<https://api.del.icio.us/v1/posts/get>

6 week notice period

Used 301 redirect

## *PayPal*

From Jan 2008 require a new parameter called VERSION to be set

VERSION is equal to PayPal's internal release number

New services require the version to be set to the release number they were first issued under

## *Google AdWords*

SOAP service, dependent on WSDL

Currently on version 12

Previous version is supported for 4 months  
and then turned off

*Talis Example 1: removal of /bf*

Previously we had URIs like:

<http://api.talis.com/bf/stores/xyz>

now

<http://api.talis.com/stores/xyz>

## *Talis Example 1*

**No deprecation of previous URI**

**Use HTTP redirects**

**Client problems...**

## *Talis Example 2*

First services had /1/ in them:

<http://api.talis.com/1/dir/collections>

Unclear about versioning policy... are all services in lockstep?

Can /1/ services work with /2/ ones?

## *Talis Example 2*

Left /1/ as historical artifact

New services don't include it

## *Namespaces*

Terms evolve independently of namespaces

Changing URIs is a forwards-incompatible change

FOAF has 0.1 in its URI and will forever now

<http://xmlns.com/foaf/0.1/>

## *Dublin Core*

<http://purl.org/dc/elements/1.0/>

<http://purl.org/dc/elements/1.1/>

Mirrored into <http://purl.org/dc/terms/>

Editorial errata do not change URIs

Semantic changes to terms result in new URIs

Addition of terms use same namespace

# Conclusions

*What Is Really Going On Here?*

Building dependencies on  
URI structure into our apps

Not using hypermedia as  
engine of app state

These APIs should be  
discoverable at runtime

## *Pragmatics*

It is possible to use  
Web standards to help  
evolvability

But many users don't care  
or understand

So some pragmatism is in order

## *Communication*

Define your policy

Deprecation?

Notice period?

Set expectations

Educate

Thank you and  
any questions?

shared innovation™

[www.talis.com/platform](http://www.talis.com/platform)