Measuring REST

Subbu Allamaraju

http://www.subbu.org

http://twitter.com/sallamar

My (Im)Maturity

Level 1: What is REST?

Level 2: Wow! … What? … Aha!

Level 3: Explain, rationalize

Level 4: Build stuff

Solutions for Improving Scalability and Simplicity

RESTful Web Services Cookbook

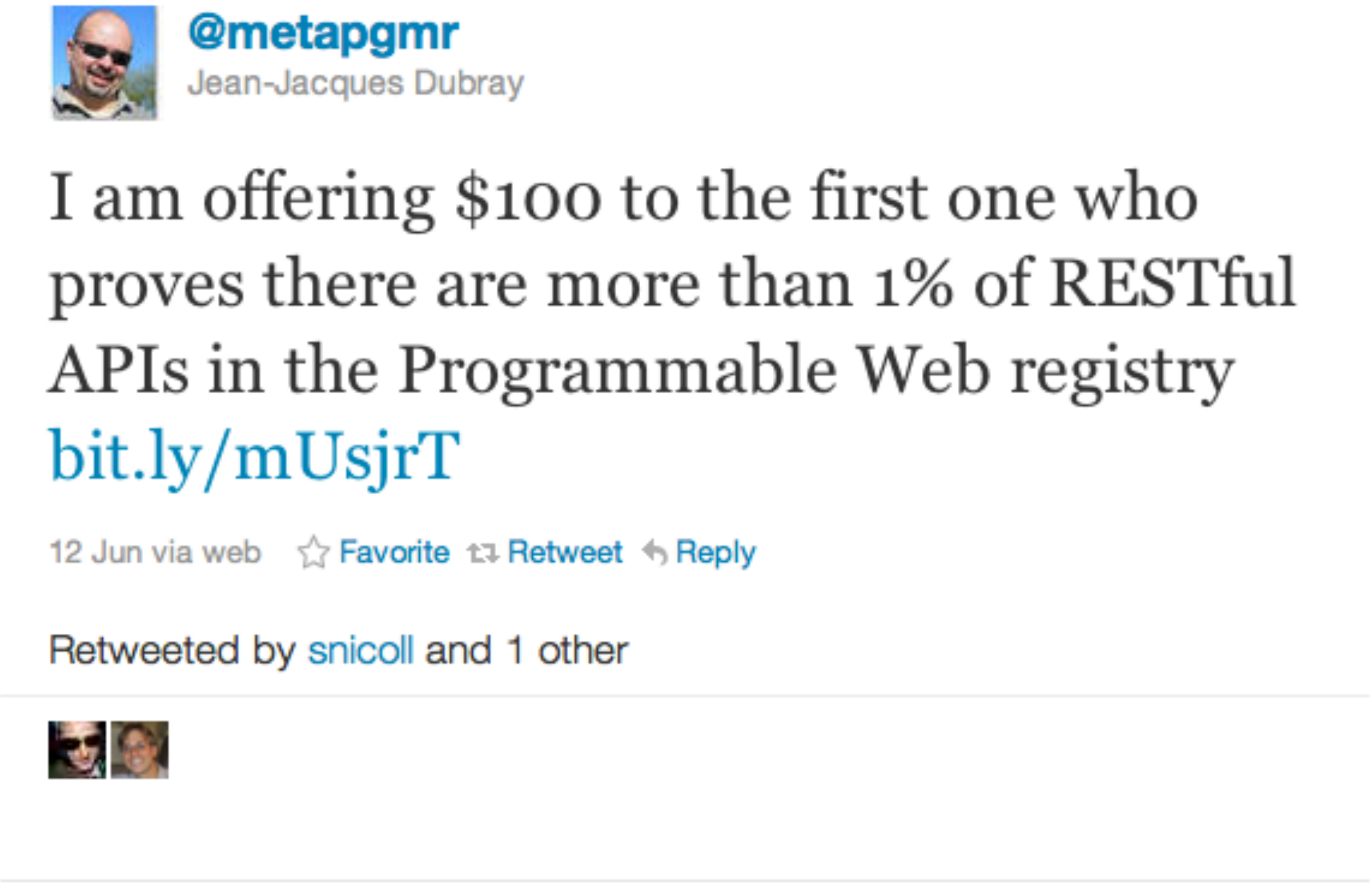
O'REILLY® | YAHOO!® PRESS

Subbu Allamaraju

Hypermedia and Media Types

~~How to measure REST?~~

What to measure?

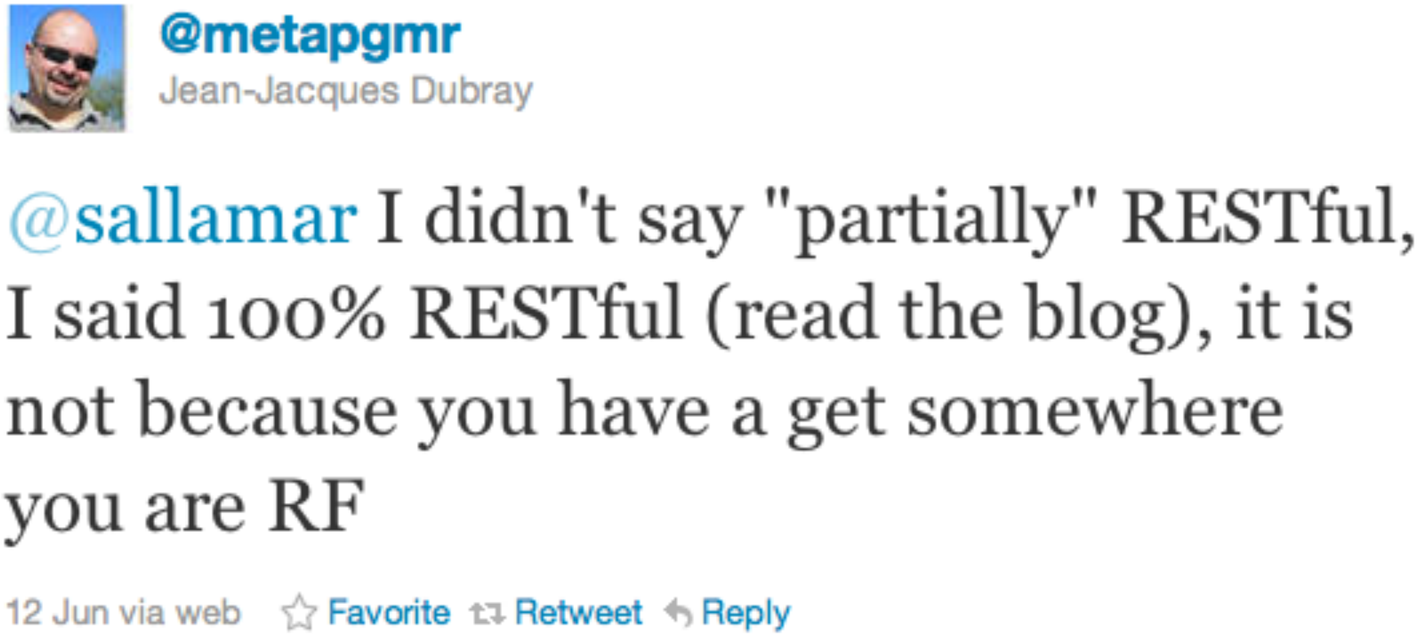


@metapgmr
Jean-Jacques Dubray

I am offering $100 to the first one who proves there are more than 1% of RESTful APIs in the Programmable Web registry bit.ly/mUsjrT

12 Jun via web

Retweeted by snicoll and 1 other

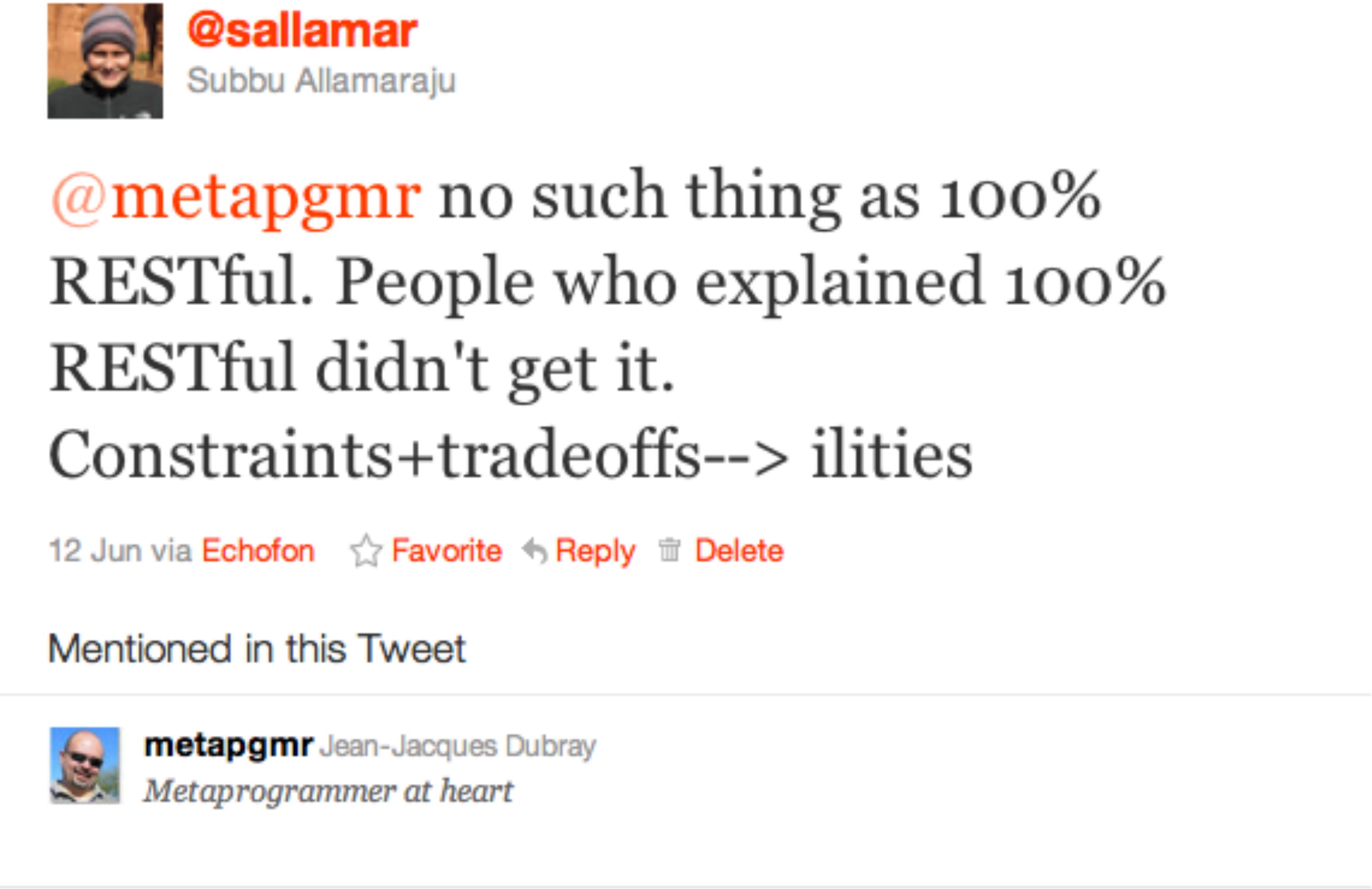


@metapgmr

Jean-Jacques Dubray

@sallamar I didn't say "partially" RESTful, I said 100% RESTful (read the blog), it is not because you have a get somewhere you are RF

12 Jun via web



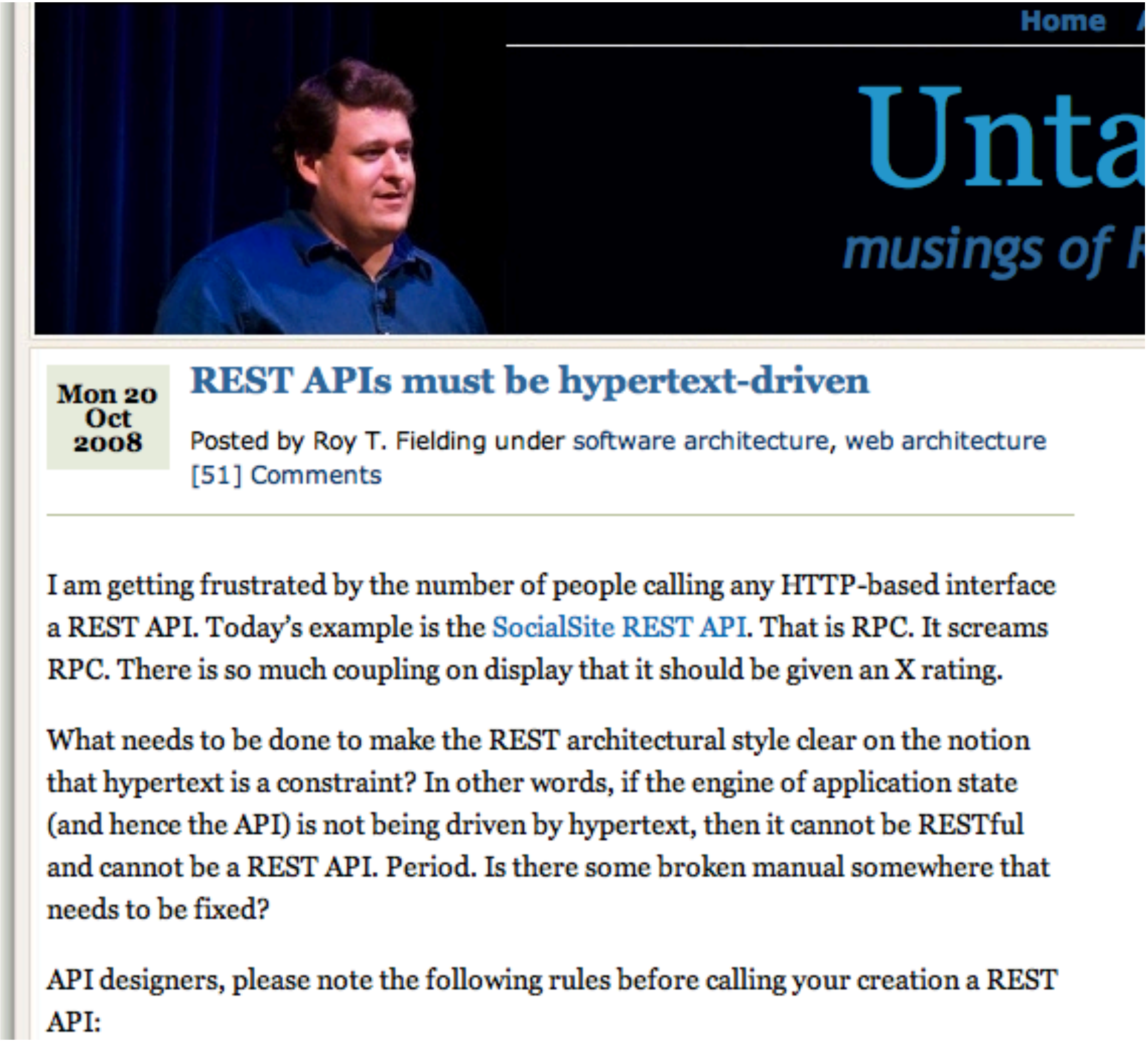
@sallamar
Subbu Allamaraju

@metapgmr no such thing as 100% RESTful. People who explained 100% RESTful didn't get it. Constraints+tradeoffs--> ilities

12 Jun via Echofon ☆ Favorite ↰ Reply 🗑 Delete

Mentioned in this Tweet

metapgmr Jean-Jacques Dubray
Metaprogrammer at heart



Mon 20 Oct 2008

REST APIs must be hypertext-driven

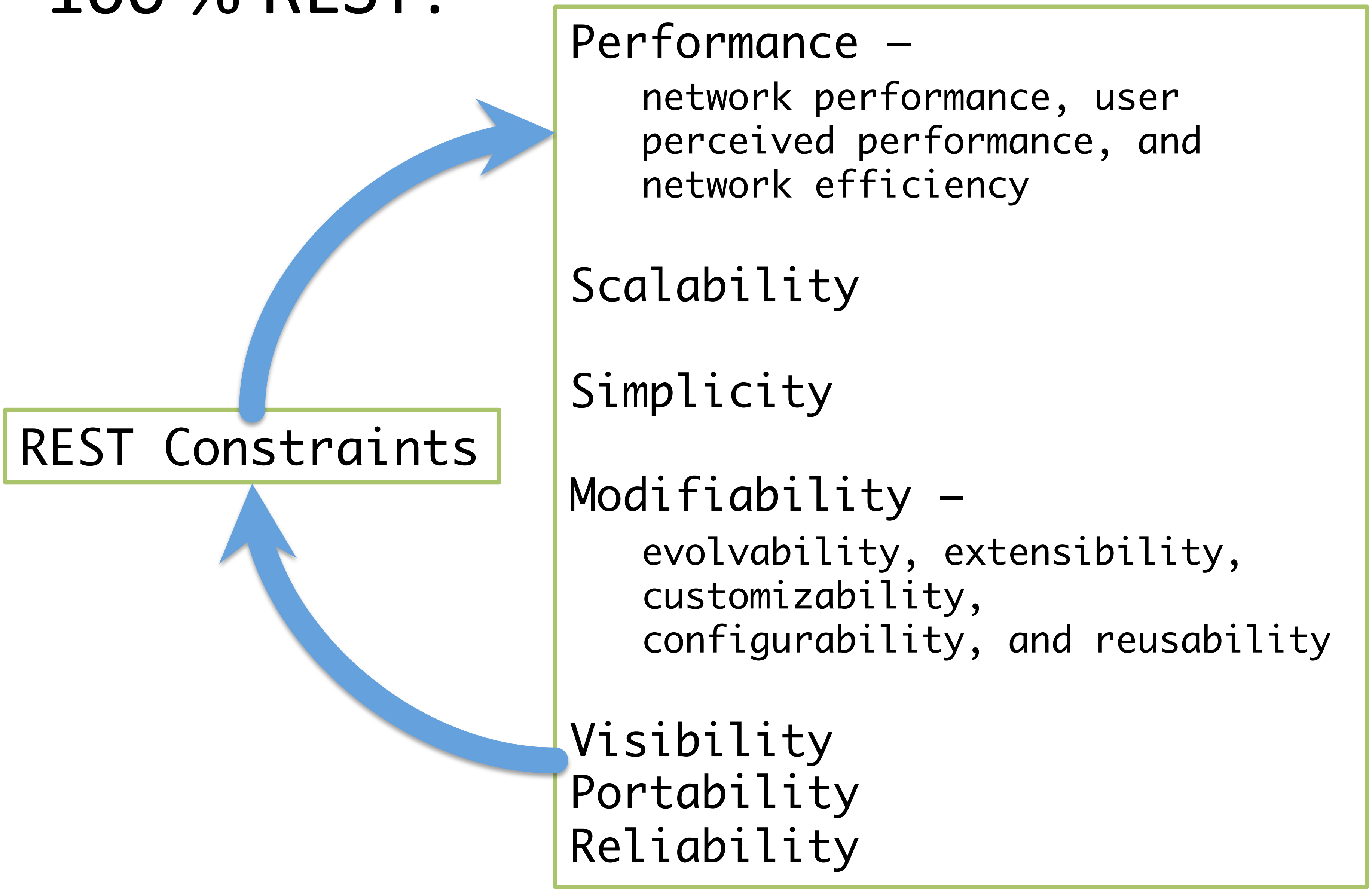
Posted by Roy T. Fielding under software architecture, web architecture
[51] Comments

I am getting frustrated by the number of people calling any HTTP-based interface a REST API. Today's example is the SocialSite REST API. That is RPC. It screams RPC. There is so much coupling on display that it should be given an X rating.

What needs to be done to make the REST architectural style clear on the notion that hypertext is a constraint? In other words, if the engine of application state (and hence the API) is not being driven by hypertext, then it cannot be RESTful and cannot be a REST API. Period. Is there some broken manual somewhere that needs to be fixed?

API designers, please note the following rules before calling your creation a REST API:

100 % REST?



REST Constraints

Performance –

network performance, user perceived performance, and network efficiency

Scalability

Simplicity

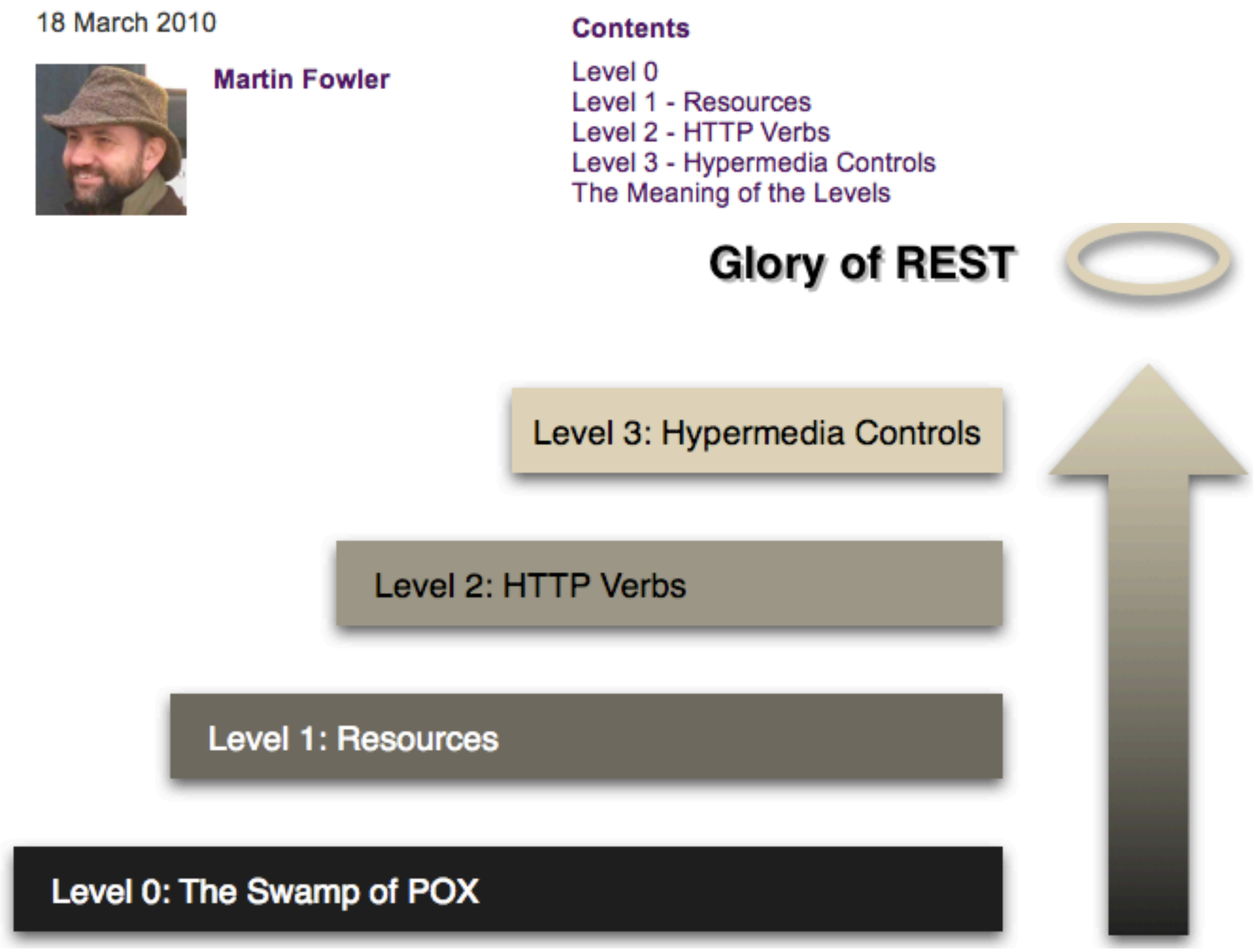
Modifiability –

evolvability, extensibility, customizability, configurability, and reusability

Visibility

Portability

Reliability

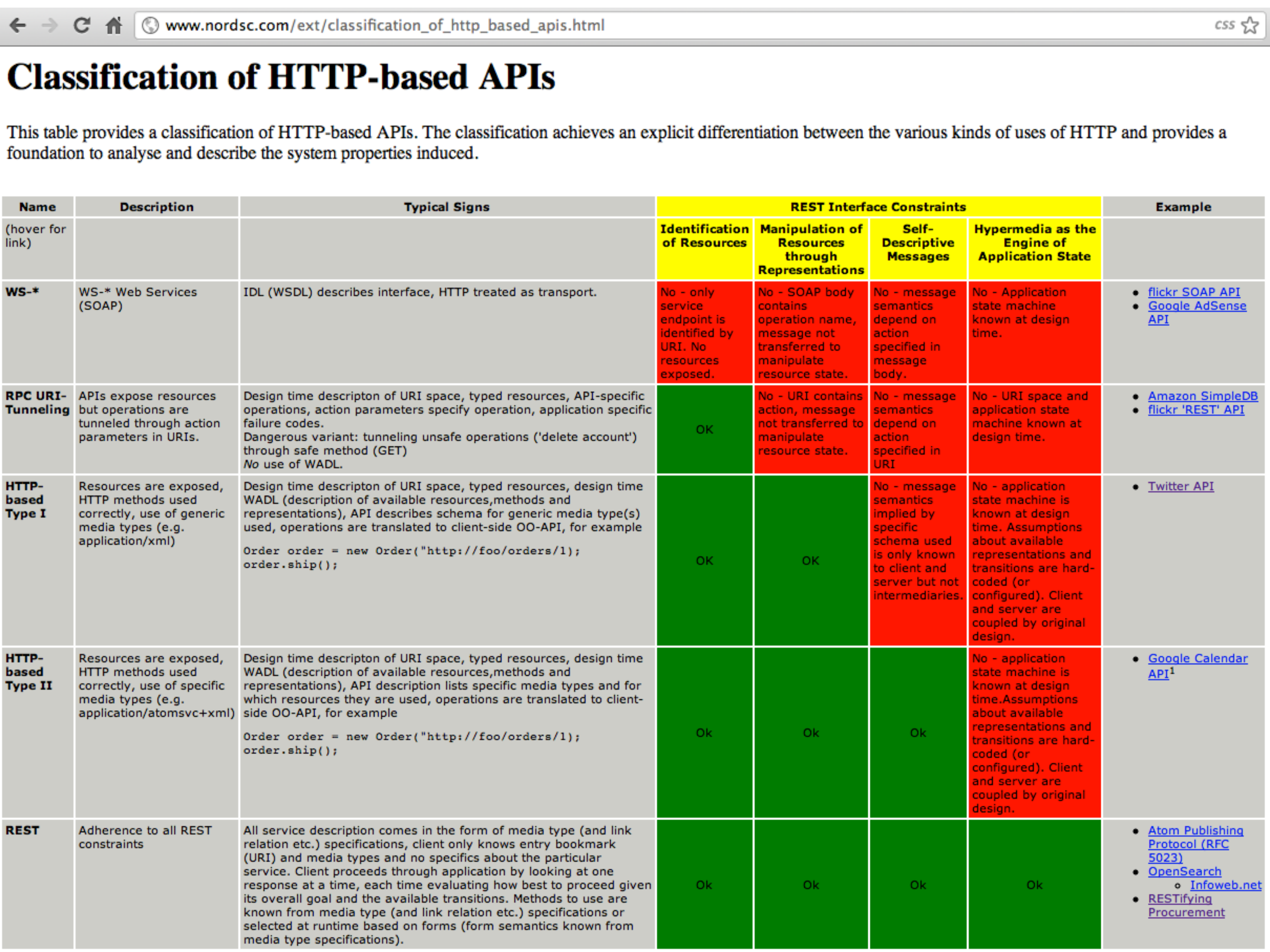


18 March 2010

Martin Fowler

Contents

- But “glory” is not a systemic quality.
- Glory is not measurable
- Does not lead to $$$



www.nordsc.com/ext/classification_of_http_based_apis.html

Classification of HTTP-based APIs

This table provides a classification of HTTP-based APIs. The classification achieves an explicit differentiation between the various kinds of uses of HTTP and provides a foundation to analyse and describe the system properties induced.

Name	Description	Typical Signs	REST Interface Constraints				Example
(hover for link)			Identification of Resources	Manipulation of Resources through Representations	Self-Descriptive Messages	Hypermedia as the Engine of Application State	
WS-*	WS-* Web Services (SOAP)	IDL (WSDL) describes interface, HTTP treated as transport.	No - only service endpoint is identified by URI. No resources exposed.	No - SOAP body contains operation name, message not transferred to manipulate resource state.	No - message semantics depend on action specified in message body.	No - Application state machine known at design time.	flickr SOAP API Google AdSense API
RPC URI-Tunneling	APIs expose resources but operations are tunneled through action parameters in URIs.	Design time descripton of URI space, typed resources, API-specific operations, action parameters specify operation, application specific failure codes. Dangerous variant: tunneling unsafe operations ('delete account') through safe method (GET) *No* use of WADL.	OK	No - URI contains action, message not transferred to manipulate resource state.	No - message semantics depend on action specified in URI	No - URI space and application state machine known at design time.	Amazon SimpleDB flickr 'REST' API

HTTP-based Type I	Resources are exposed, HTTP methods used correctly, use of generic media types (e.g. application/xml)	Design time descripton of URI space, typed resources, design time WADL (description of available resources,methods and representations), API describes schema for generic media type(s) used, operations are translated to client-side OO-API, for example
		`Order order = new Order("http://foo/orders/1);`
		`order.ship();`
			OK	OK	No - message semantics implied by specific schema used is only known to client and server but not intermediaries.	No - application state machine is known at design time. Assumptions about available representations and transitions are hard-coded (or configured). Client and server are coupled by original design.	Twitter API

HTTP-based Type II	Resources are exposed, HTTP methods used correctly, use of specific media types (e.g. application/atomsvc+xml)	Design time descripton of URI space, typed resources, design time WADL (description of available resources,methods and representations), API description lists specific media types and for which resources they are used, operations are translated to client-side OO-API, for example
		`Order order = new Order("http://foo/orders/1);`
		`order.ship();`
			Ok	Ok	Ok	No - application state machine is known at design time.Assumptions about available representations and transitions are hard-coded (or configured). Client and server are coupled by original design.	Google Calendar API[1]

REST	Adherence to all REST constraints	All service description comes in the form of media type (and link relation etc.) specifications, client only knows entry bookmark (URI) and media types and no specifics about the particular service. Client proceeds through application by looking at one response at a time, each time evaluating how best to proceed given its overall goal and the available transitions. Methods to use are known from media type (and link relation etc.) specifications or selected at runtime based on forms (form semantics known from media type specifications).	Ok	Ok	Ok	Ok	Atom Publishing Protocol (RFC 5023) OpenSearch ○ Infoweb.net RESTifying Procurement

Show me 100%
RESTful apps

Is REST a hoax then?

Question #1: Is this a marketing or messaging problem?

Question #2: Can you prove that sum(constraints) leads to sum(qualities)?

Hand-wave

Reason out and rationalize in the abstract

- Nah, we need to put some engineering back into software

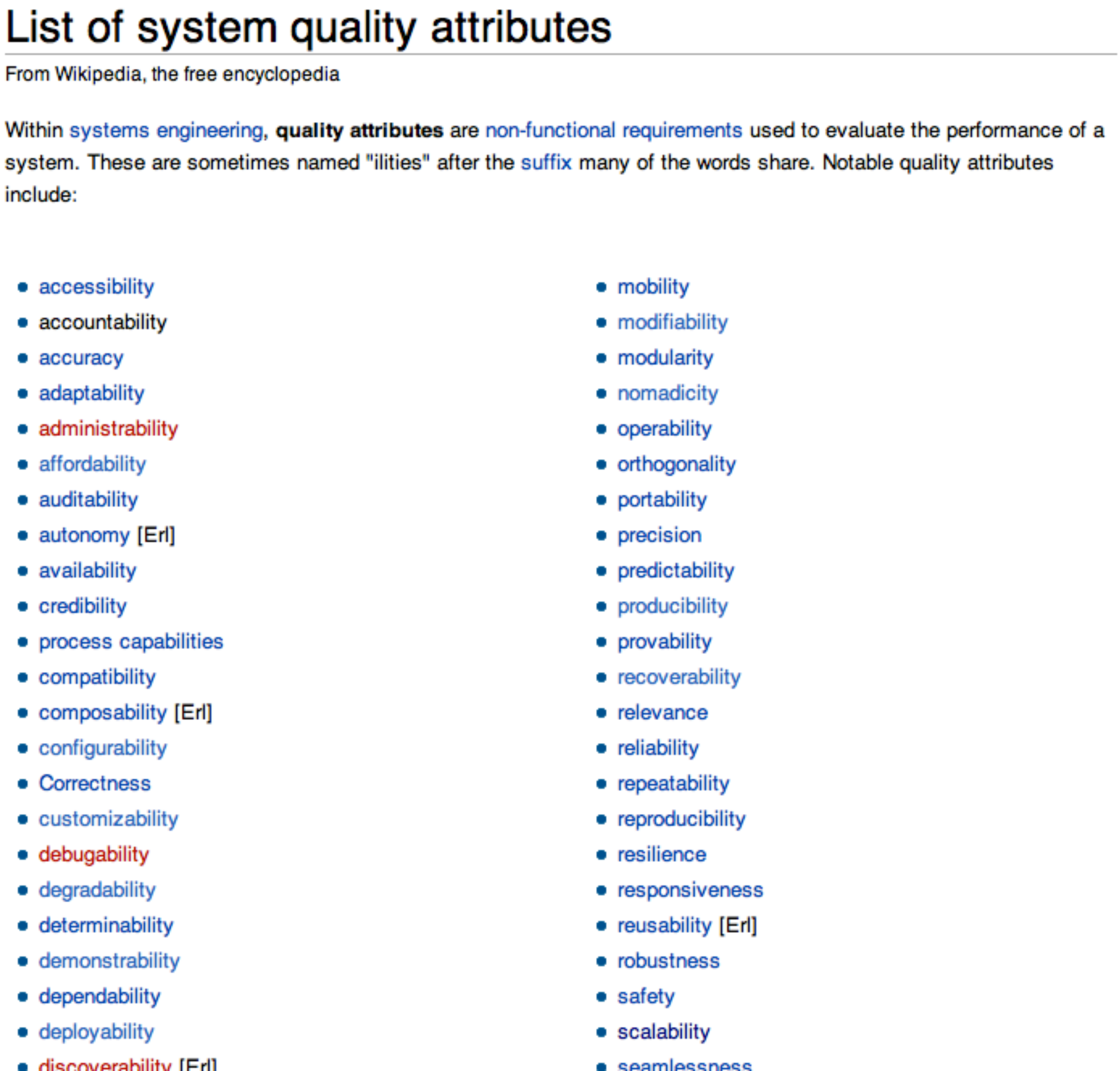
Question #3: Can you prove that any given constraint leads to any quality?

Hand-wave

Advocate: “Make judicious tradeoffs”

- But how do I know if I’m making *judicious* tradeoffs?

Step #1: Agree on a set of qualities that matter for your app



List of system quality attributes

From Wikipedia, the free encyclopedia

Within systems engineering, **quality attributes** are non-functional requirements used to evaluate the performance of a system. These are sometimes named "ilities" after the suffix many of the words share. Notable quality attributes include:

- accessibility
- accountability
- accuracy
- adaptability
- administrability
- affordability
- auditability
- autonomy [Erl]
- availability
- credibility
- process capabilities
- compatibility
- composability [Erl]
- configurability
- Correctness
- customizability
- debugability
- degradability
- determinability
- demonstrability
- dependability
- deployability
- discoverability [Erl]
- mobility
- modifiability
- modularity
- nomadicity
- operability
- orthogonality
- portability
- precision
- predictability
- producibility
- provability
- recoverability
- relevance
- reliability
- repeatability
- reproducibility
- resilience
- responsiveness
- reusability [Erl]
- robustness
- safety
- scalability
- seamlessness

Ex:

- Performance
- Ease of use
- Adoption

Roy’s list is not the super-set

Step #2: Contextualize the qualities into scenarios

- Performance
  - Serve a photo in *“t”* msec
  - Process *“n”* photos per hour
- Ease of use
  - A developer should be able to build a client app in 30 minutes
- Adoption
  - Must gain *“y00”* developers in 3 months after launch

Step #3: Prioritize qualities and scenarios

- Adoption
  - Must gain *“y00”* developers in 3 months after launch
- Performance
  - Serve a photo in *“t”* msec
  - Process *“n”* photos per hour
- Ease of use
  - A developer should be able to build a client app in 30 minutes

Step #4: Pick solutions among alternatives that you think meet the scenarios

- Adoption
  - Must gain *“y00”* developers in 3 months after launch
- Performance
  - Serve a photo in *“t”* msec
  - Process *“n”* photos per hour
- Ease of use
  - A developer should be able to build a client app in 30 minutes

What should you measure?

$$$
$$$,$$$
$$$,$$$,$$$

Make the case for REST constraints based on specific scenarios and priorities