June 2019

Practical API Design

Ronnie Mitra

ronnie.mitra@publicissapient.com
@mitraman

publicis sapient

Publicis Sapient | Digital Business Transformation

As a digital business transformation partner of choice, we've spent nearly three decades utilising the disruptive power of technology and ingenuity to help digitally enable our clients' business in their pursuit of next

Our scale

20,000

passionate people

50+

offices globally connect

30

years of digital pioneering and customer innovation

Our clients

Nationwide Building Society · AVIVA · LLOYDS BANKING GROUP · HSBC · T-Mobile · Goldman Sachs · Marriott · SAMSUNG

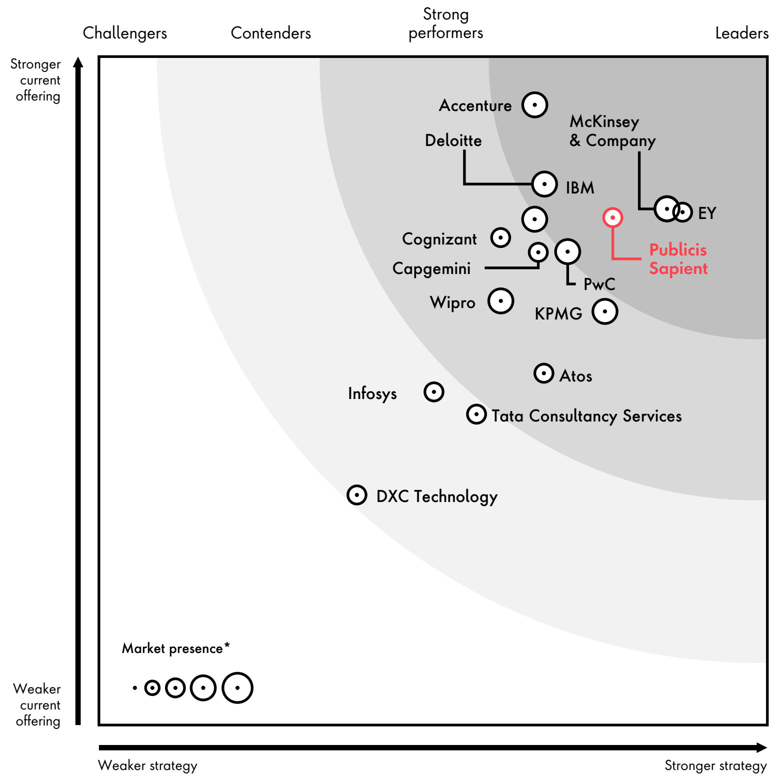
Serving you

A startup mindset and agile methods to unlock value in ways that delight your customers and improve their operational effectiveness

A transformation approach that is grounded in a view of both the company and the customer simultaneously

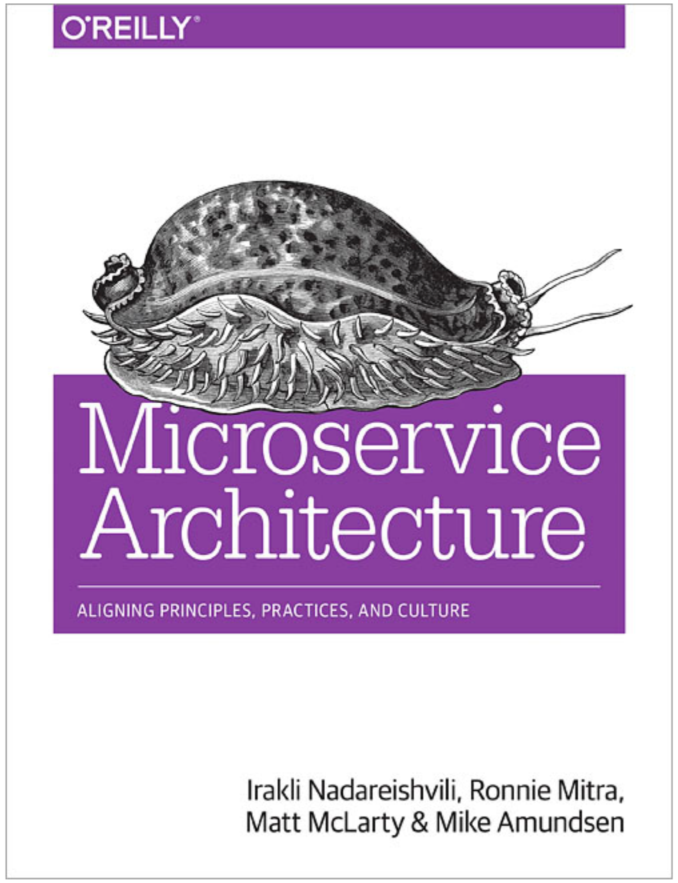
A unique fusing of strategy and consulting, experience and engineering with an enduring culture of problem-solving creativity

Industry Recognition



Forrester Global Digital Business Transformation Accelerators – Q1 2019

Ronnie Mitra

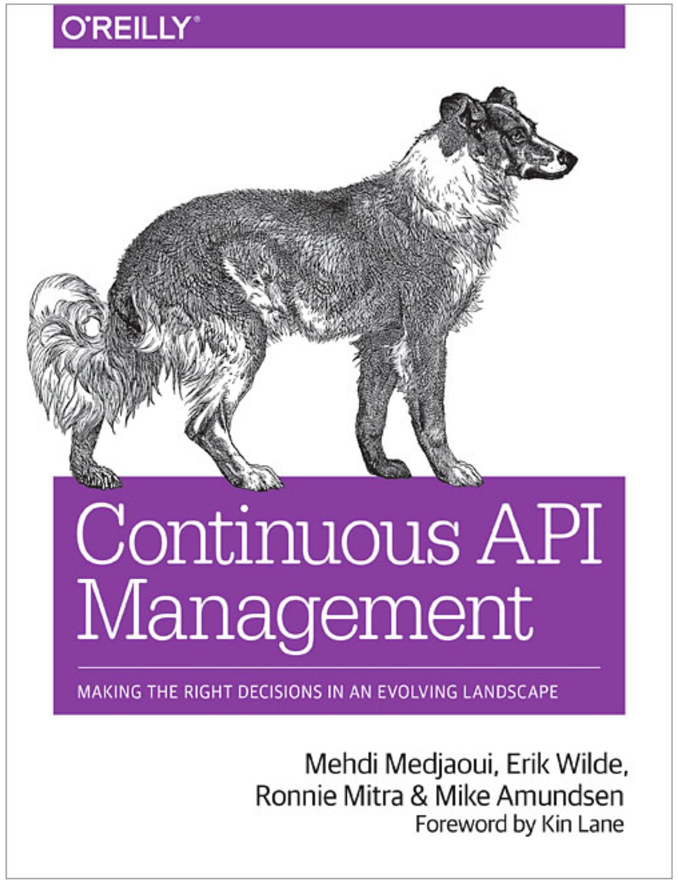


O'REILLY®

Microservice Architecture

ALIGNING PRINCIPLES, PRACTICES, AND CULTURE

Irakli Nadareishvili, Ronnie Mitra, Matt McLarty & Mike Amundsen



O'REILLY®

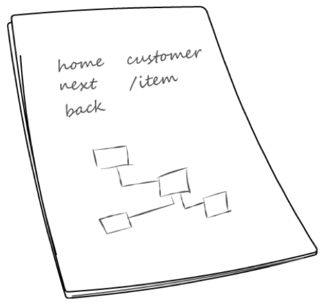
Continuous API Management

MAKING THE RIGHT DECISIONS IN AN EVOLVING LANDSCAPE

Mehdi Medjaoui, Erik Wilde, Ronnie Mitra & Mike Amundsen
Foreword by Kin Lane

The Scope of API Design

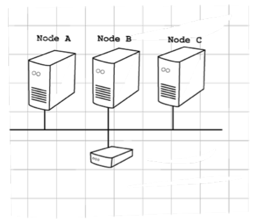
Interface Model



Implementation

```
// HTTP Request Handler
onRequest(request, response) {
  // Make sure the database connection is up
  db = initDb();

  // Check the path and call
  if(request.URI === '/customer') {
    customerHandler(request, response);
  }else if( request.URI === '/products') {
    productHandler(request, response);
  }
}
```

Instance



Supporting Assets & Tools

/customer

The customer collection resource.

id	identifier
page_count	number of response pages

```
GET /customer

200 OK
{
 "name":"Al",
 "id":"3"
}
```

API

Significant API Design Costs

Interface Design Cost

API Engineering Cost

Client Engineering Cost

After Publishing

Change Cost

Safety Cost

7 Practical Techniques For API Design

Technique #1
Set The Right Design Goals

Typical API Design Goals

Access to Data & Services

Typical API Design Goals

Reduced Reliance on Staff

Reduced Learning Time

Increased Developer Productivity

Access to Data & Services

Typical API Design Goals

Increased Conversion Rate

Talent Retention

Brand Credibility

Reduced Reliance on Staff

Reduced Learning Time

Increased Developer Productivity

Access to Data & Services

Typical API Design Goals

Experience Focus

- Increased Conversion Rate
- Talent Retention
- Brand Credibility

Usability Focus

- Reduced Reliance on Staff
- Reduced Learning Time
- Increased Developer Productivity

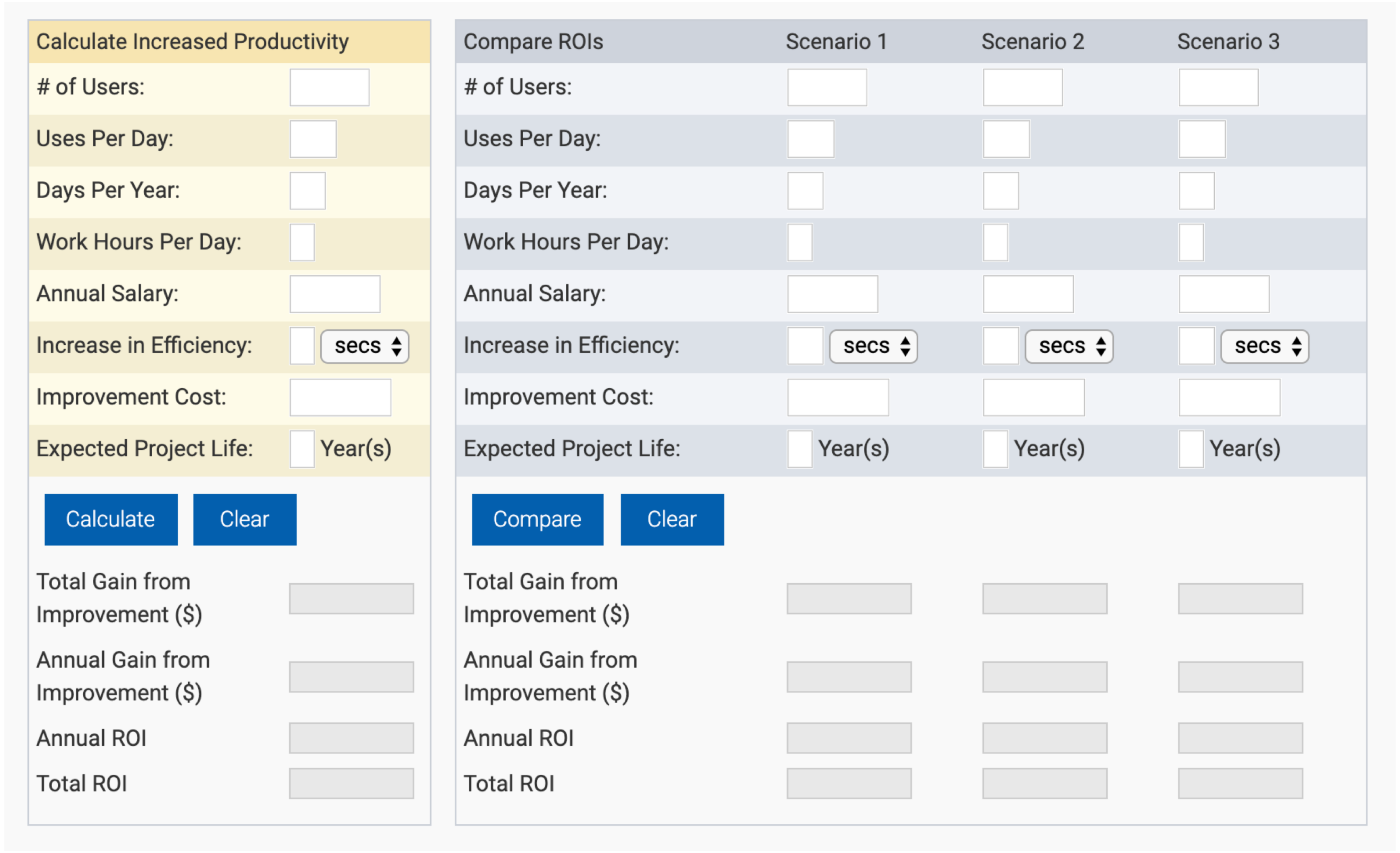
Functional Focus

- Access to Data & Services

Design costs increases as we move up

Calculating The Cost-Benefit of API Design

https://humanfactors.com/coolstuff/roi_increase_productivity.asp



Calculate Increased Productivity	
# of Users:	
Uses Per Day:	
Days Per Year:	
Work Hours Per Day:	
Annual Salary:	
Increase in Efficiency:	secs
Improvement Cost:	
Expected Project Life:	Year(s)

Calculate Clear

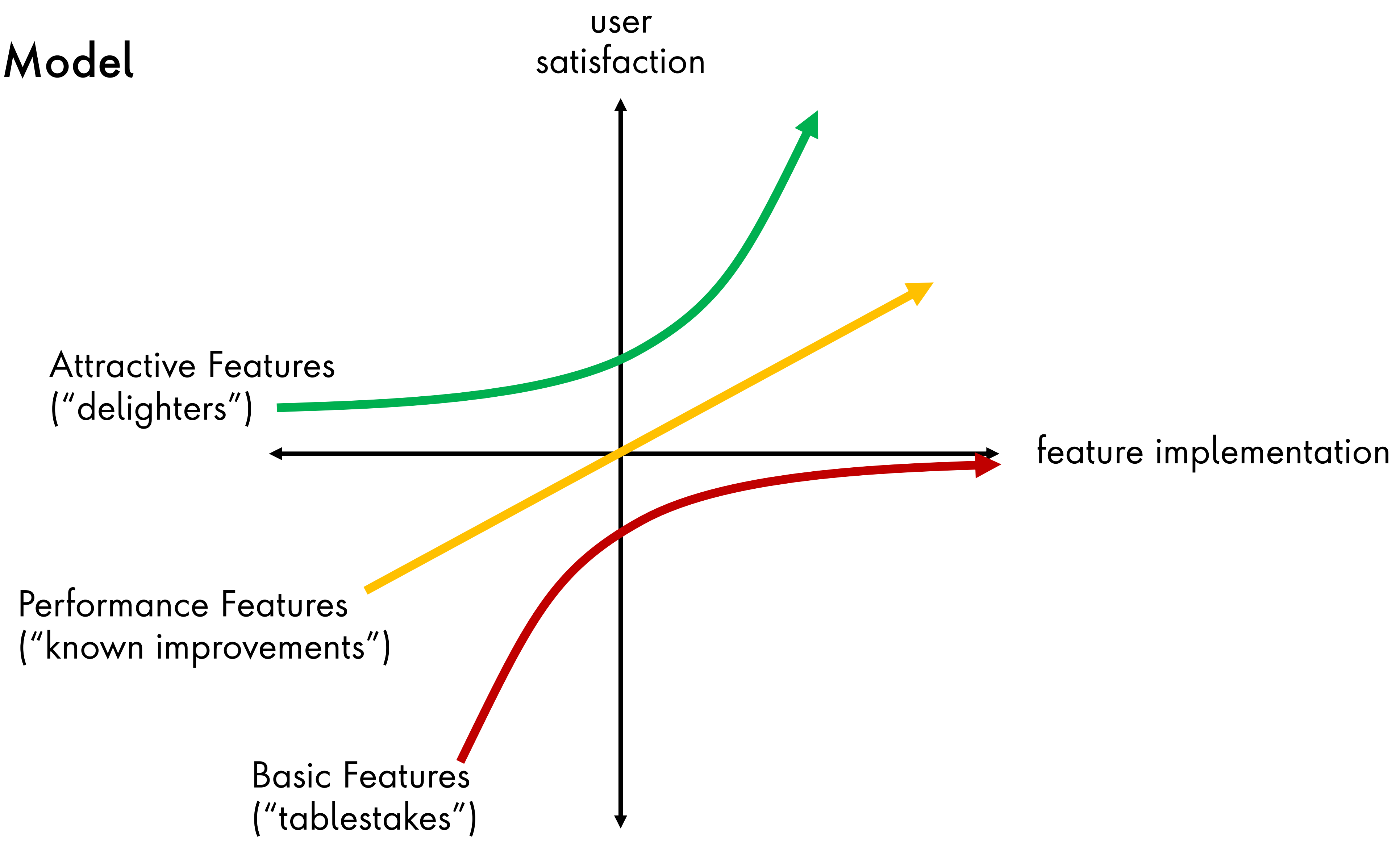
Total Gain from Improvement ($)	
Annual Gain from Improvement ($)	
Annual ROI	
Total ROI	

Compare ROIs	Scenario 1	Scenario 2	Scenario 3
# of Users:			
Uses Per Day:			
Days Per Year:			
Work Hours Per Day:			
Annual Salary:			
Increase in Efficiency:	secs	secs	secs
Improvement Cost:			
Expected Project Life:	Year(s)	Year(s)	Year(s)

Compare Clear

Total Gain from Improvement ($)			
Annual Gain from Improvement ($)			
Annual ROI			
Total ROI			

The Kano Model

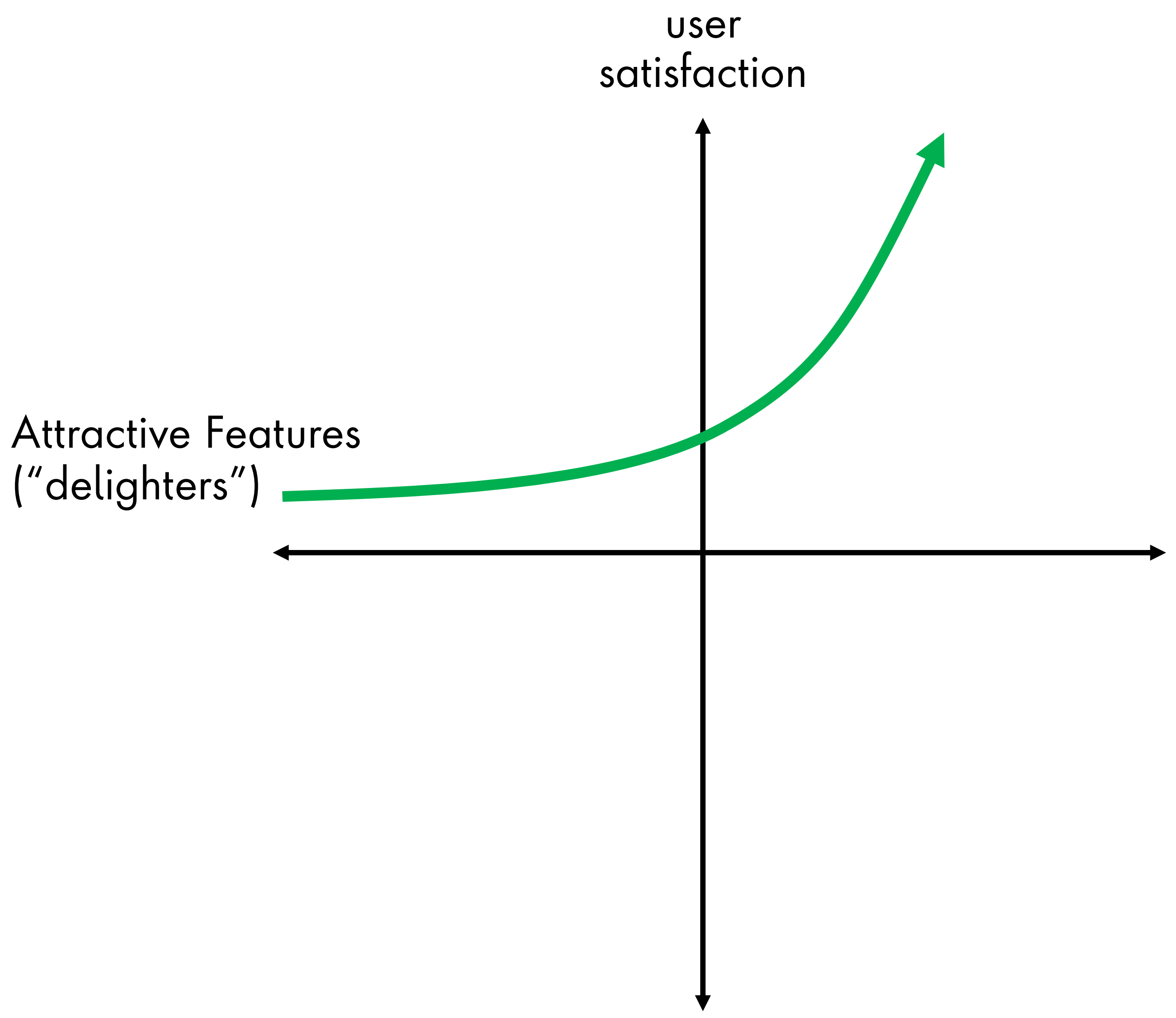

user satisfaction
feature implementation
Attractive Features ("delighters")
Performance Features ("known improvements")
Basic Features ("tablestakes")

Invest in Attractive Features when API-X is a key success factor

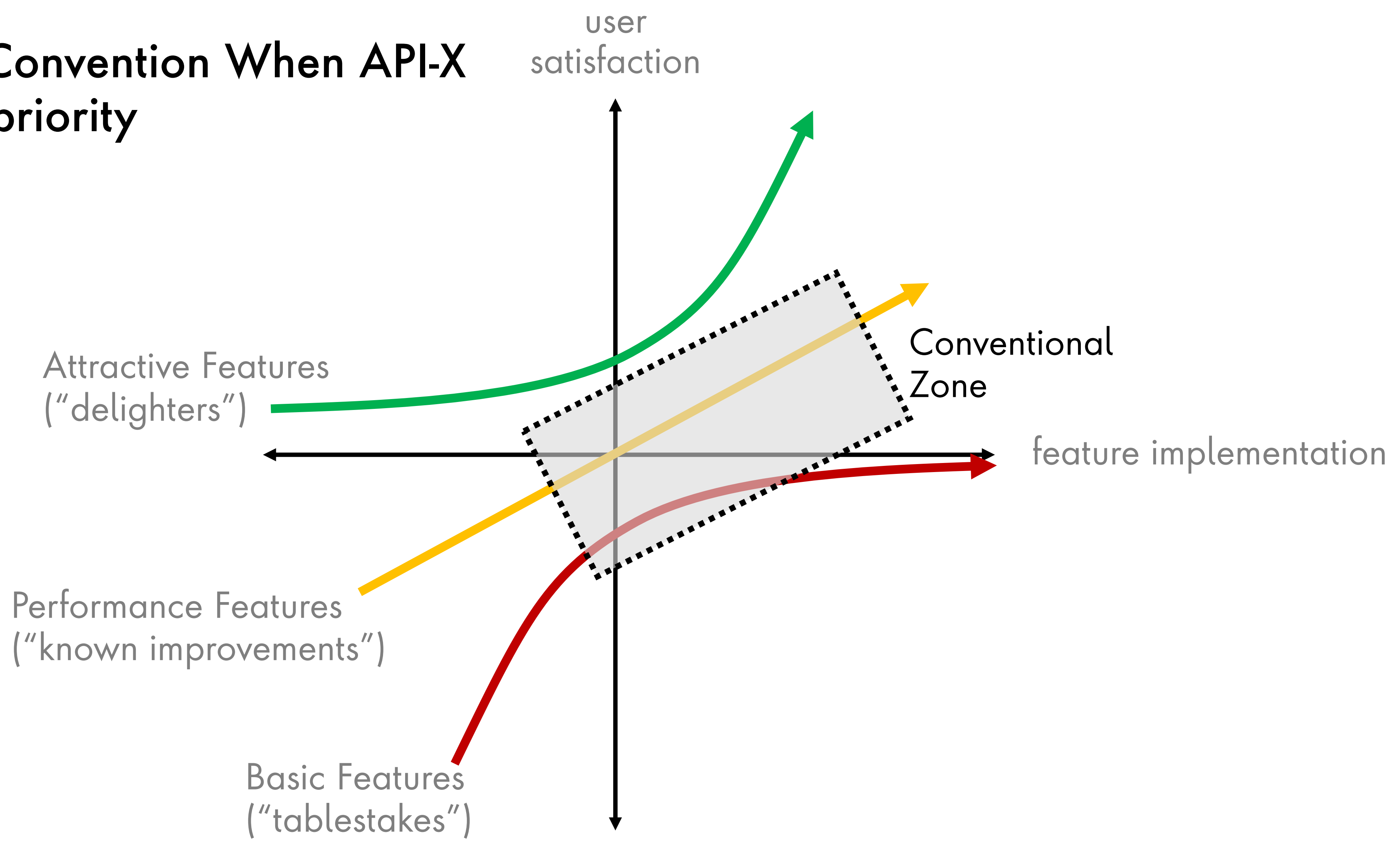
Increased Conversion Rates

Brand Marketability

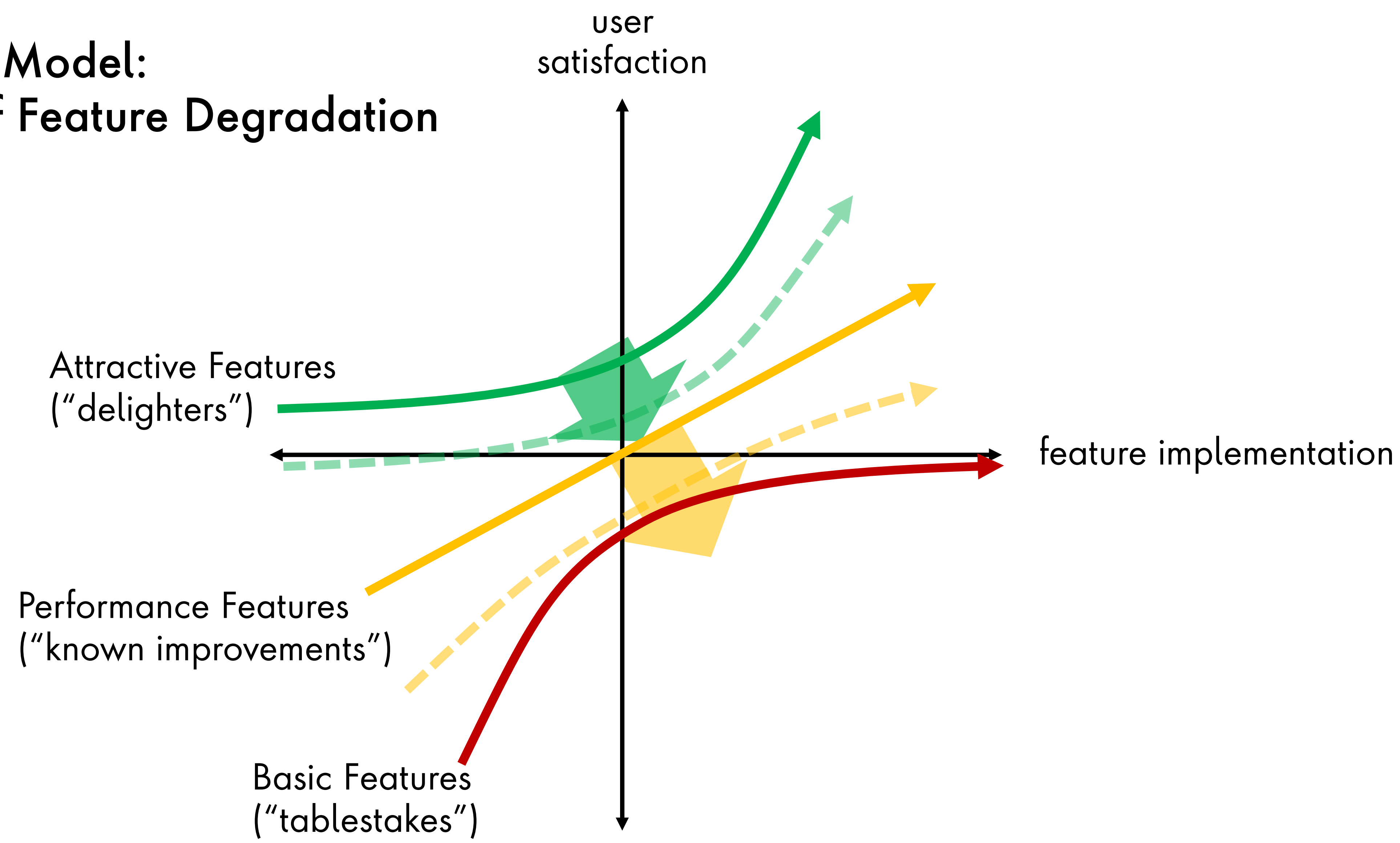
Talent Retention



Focus on Convention When API-X is not the priority



The Kano Model: Beware of Feature Degradation

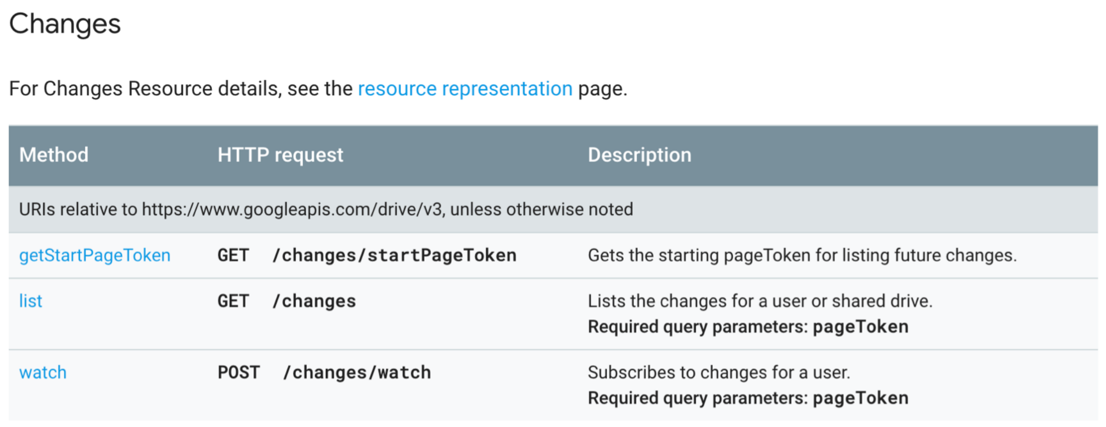

user satisfaction
feature implementation
Attractive Features ("delighters")
Performance Features ("known improvements")
Basic Features ("tablestakes")

Use Imitation as a Shortcut to a Conventional API

Save time by using another API design as inspiration

Considerations:

- Who are the API's users?
- What domain does it operate in?
- What is it like to use?



Changes

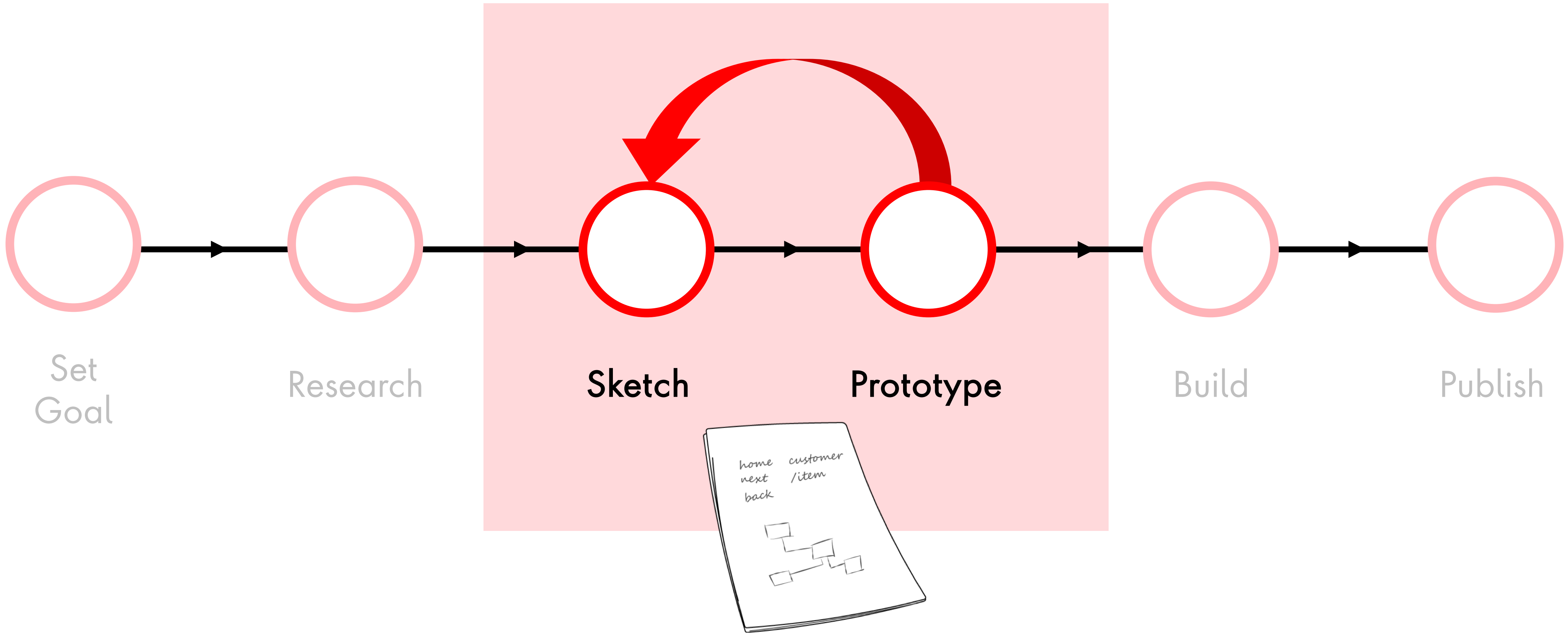
For Changes Resource details, see the resource representation page.

Method	HTTP request	Description
URIs relative to https://www.googleapis.com/drive/v3, unless otherwise noted		
getStartPageToken	`GET /changes/startPageToken`	Gets the starting pageToken for listing future changes.
list	`GET /changes`	Lists the changes for a user or shared drive. **Required query parameters: `pageToken`**
watch	`POST /changes/watch`	Subscribes to changes for a user. **Required query parameters: `pageToken`**

GET /changes
GET /changes/watch

Technique #2
Sketch & Prototype Iteratively

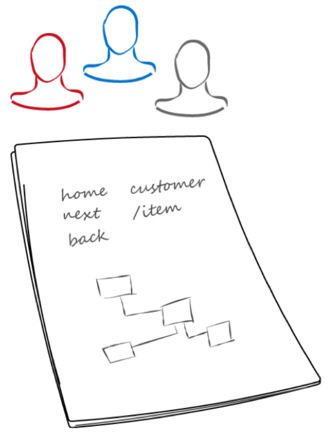
Sketch & Prototype



Technique #3
Heuristic Evaluation

API Design Reviews

Just like a code review, your API design can benefit from evaluation by other experts and your peers.

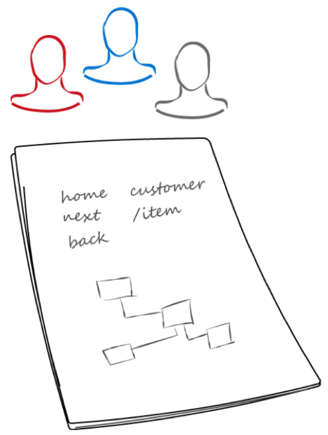


API Design Reviews

Just like a code review, your API design can benefit from evaluation by other experts and your peers.

Practical Challenges:

- Access to API design experts
- Getting comprehensive feedback
- Collating analysis from multiple experts



Jakob Neilsen and Rolf Molich: 10 Usability Heuristics for User Interface Design

1. Visibility of System Status
2. Match Between System and the Real World
3. User Control and Freedom
4. Consistency and Standards
5. Error Prevention
6. Recognition rather than recall
7. Flexibility and Efficiency of Use
8. Aesthetic and Minimalist Design
9. Help Users Recognize, Diagnose, and Recover from Errors
10. Help and Documentation

7 Usability Heuristics for API Design

1. Visibility of System Status
2. Match Between System and the Real World

   User Control and Freedom

3. Consistency and Standards
4. Error Prevention

   Recognition rather than recall

5. Flexibility and Efficiency of Use

   Aesthetic and Minimalist Design

6. Help Users Recognize, Diagnose, and Recover from Errors
7. Help and Documentation

5 Usability Heuristics for Machine Interface Design

1. Visibility of System Status

   Match Between System and the Real World

   User Control and Freedom

2. Consistency and Standards
3. Error Prevention

   Recognition rather than recall

4. Flexibility and Efficiency of Use

   Aesthetic and Minimalist Design

5. Help Users Recognize, Diagnose, and Recover from Errors

   Help and Documentation

5 Usability Heuristics for Machine Interface Design

1. Visibility of System Status
2. Consistency and Standards
3. Error Prevention
4. Flexibility and Efficiency of Use
5. Help Users Recognize, Diagnose, and Recover from Errors

How easy is it to understand what is happening?

5 Usability Heuristics for Machine Interface Design

1. Visibility of System Status
2. **Consistency and Standards**
3. Error Prevention
4. Flexibility and Efficiency of Use
5. Help Users Recognize, Diagnose, and Recover from Errors

Are interface and data models internally consistent?

Does the API adhere to specifications and organizational standards?

5 Usability Heuristics for Machine Interface Design

1. Visibility of System Status
2. Consistency and Standards
3. **Error Prevention**
4. Flexibility and Efficiency of Use
5. Help Users Recognize, Diagnose, and Recover from Errors

Are the interface model and data model overly complicated?

Is there avoidable tight coupling that will cause errors when things change?

5 Usability Heuristics for Machine Interface Design

1. Visibility of System Status
2. Consistency and Standards
3. Error Prevention
4. **Flexibility and Efficiency of Use**
5. Help Users Recognize, Diagnose, and Recover from Errors

Does the interface model support both beginner and advanced use cases?

Are their optimizations and accelerators available?

5 Usability Heuristics for Machine Interface Design

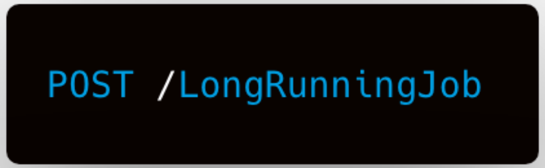
1. Visibility of System Status
2. Consistency and Standards
3. Error Prevention
4. Flexibility and Efficiency of Use
5. **Help Users Recognize, Diagnose, and Recover from Errors**

Is error information accurate and helpful?

Does it address both human and machine concerns?

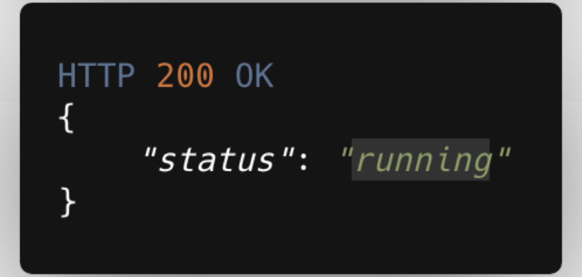
Example of a Heuristic Analysis

REQUEST



```
POST /LongRunningJob
```

RESPONSE



```
HTTP 200 OK
{
    "status": "running"
}
```

Visibility:

- “Use 202 instead”
- “Provide a link where client can check job status and add some info about job length”

Consistency & Standards:

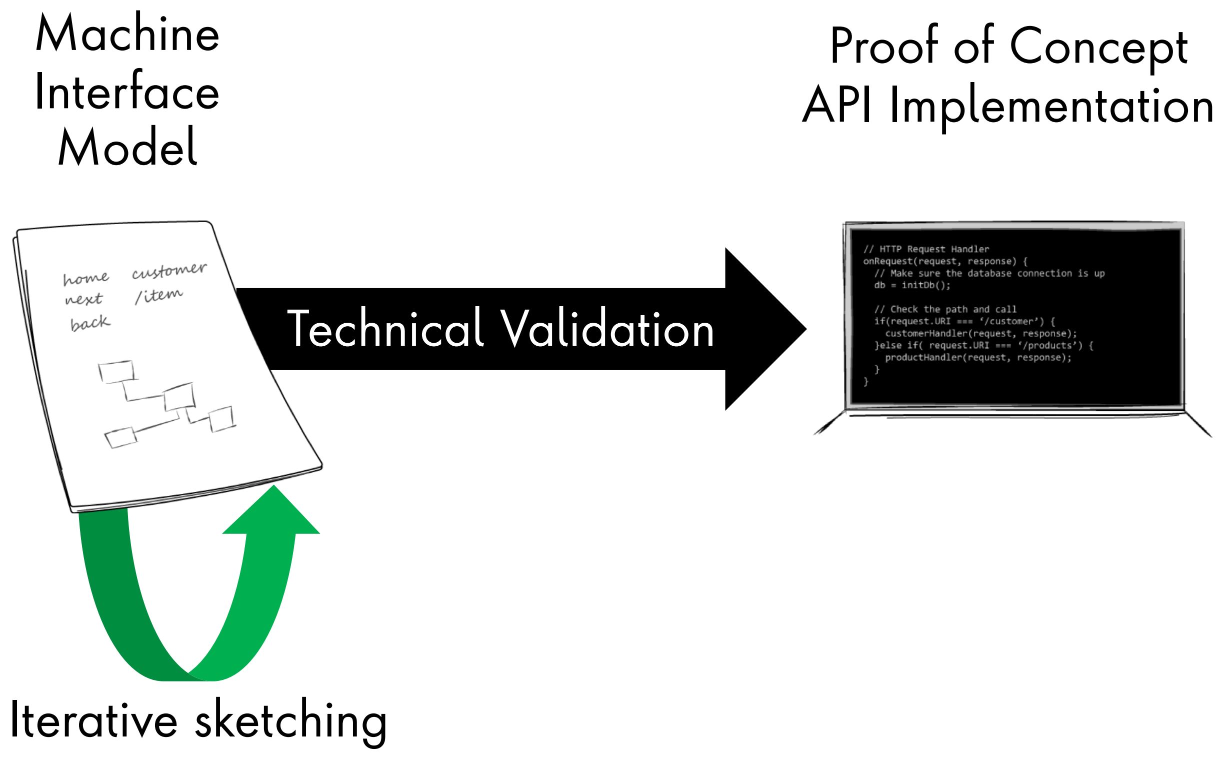
- “Use our standardized words for job status (“in-progress”)”

Find Usability Problems by Combining Results

	Reviewer A	Reviewer B	Reviewer C	Reviewer D
Visibility of System Status			🟥	
Consistency & Standards	🟥	🟨	🟥	🟨
Error Prevention			🟨	
Flexibility & Efficiency of Use				
Help Users Recognize, Diagnose an Recover	🟨		🟥	

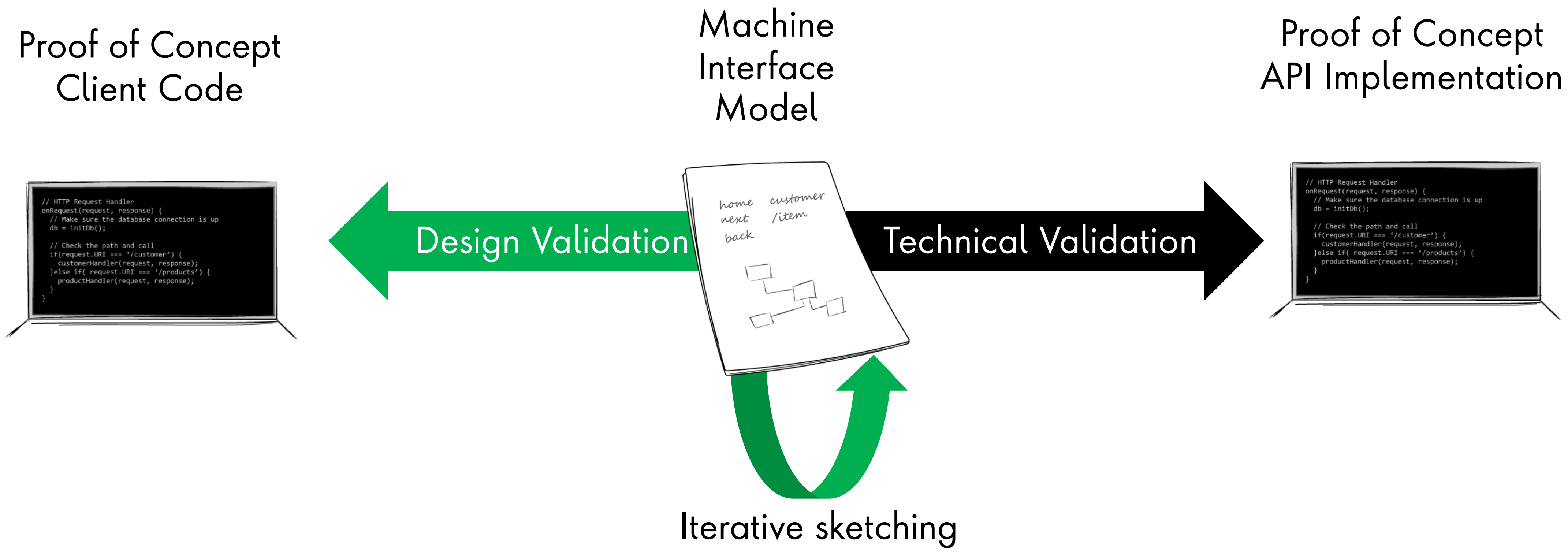
Technique #4
Write Code

Writing Code in the Design Phase



```
// HTTP Request Handler
onRequest(request, response) {
  // Make sure the database connection is up
  db = initDb();

  // Check the path and call
  if(request.URI === '/customer') {
    customerHandler(request, response);
  }else if( request.URI === '/products') {
    productHandler(request, response);
  }
}
```

Writing Code in the Design Phase



```
// HTTP Request Handler
onRequest(request, response) {
  // Make sure the database connection is up
  db = initDb();

  // Check the path and call
  if(request.URI === '/customer') {
    customerHandler(request, response);
  }else if( request.URI === '/products') {
    productHandler(request, response);
  }
}
```

```
// HTTP Request Handler
onRequest(request, response) {
  // Make sure the database connection is up
  db = initDb();

  // Check the path and call
  if(request.URI === '/customer') {
    customerHandler(request, response);
  }else if( request.URI === '/products') {
    productHandler(request, response);
  }
}
```

Technique: Write Code

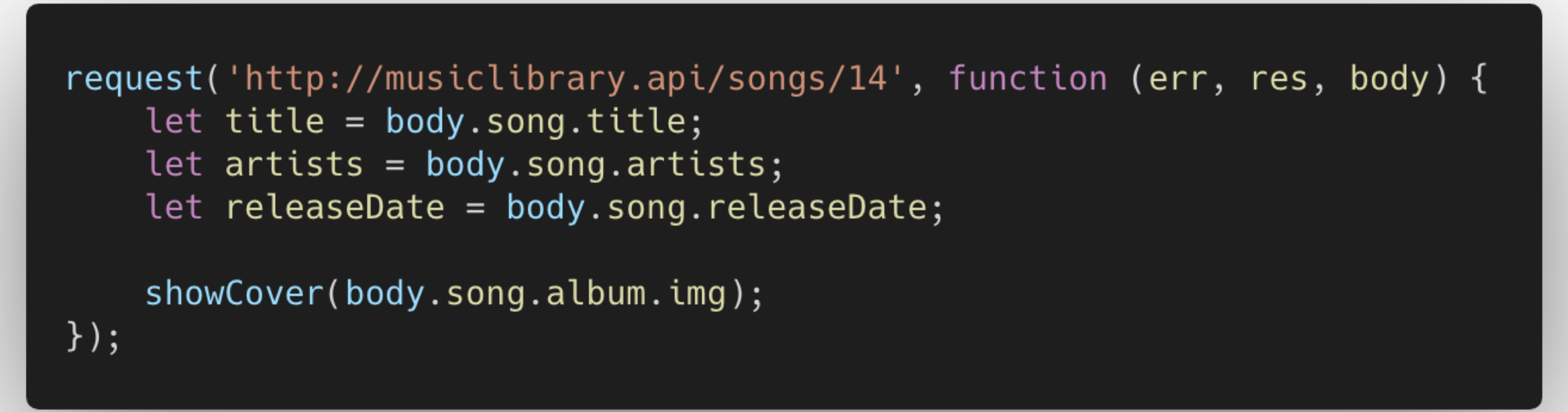
"Code the use-cases against your API before you implement it, even before you specify it properly"

– Joshua Bloch

https://github.com/jbloch

Technique: Write Client Code

Write code from the perspective of your users early in the API design cycle.



```
request('http://musiclibrary.api/songs/14', function (err, res, body) {
    let title = body.song.title;
    let artists = body.song.artists;
    let releaseDate = body.song.releaseDate;

    showCover(body.song.album.img);
});
```

Tips for Using Client Code Effectively

Be your user

Utilize languages, frameworks and techniques that you think your users would use.

Unit tests aren’t enough

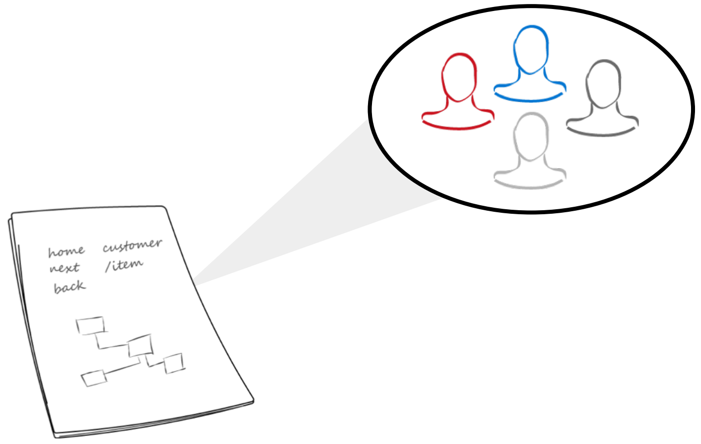
Write code that accomplishes a goal from a user perspective – not code that tests a spec.

Focus on insight not syntax

Don’t get caught investing too much time making code compile or worrying about code completeness.

Technique #5
Participatory Design

Participatory Design
High Fidelity – Co-Design Team



Frontend Developer

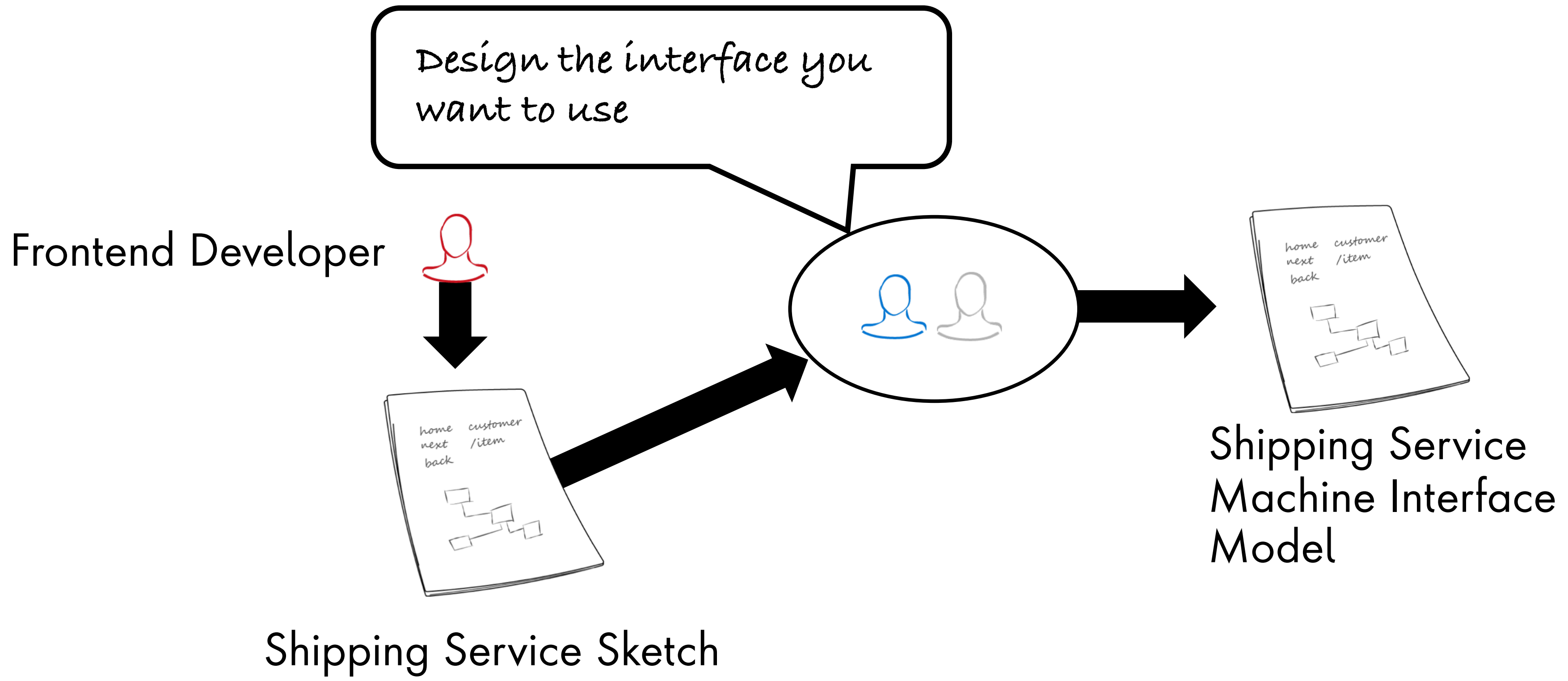
API Designer

UX Designer

Backend Developer

Shipping Service
Machine Interface Model

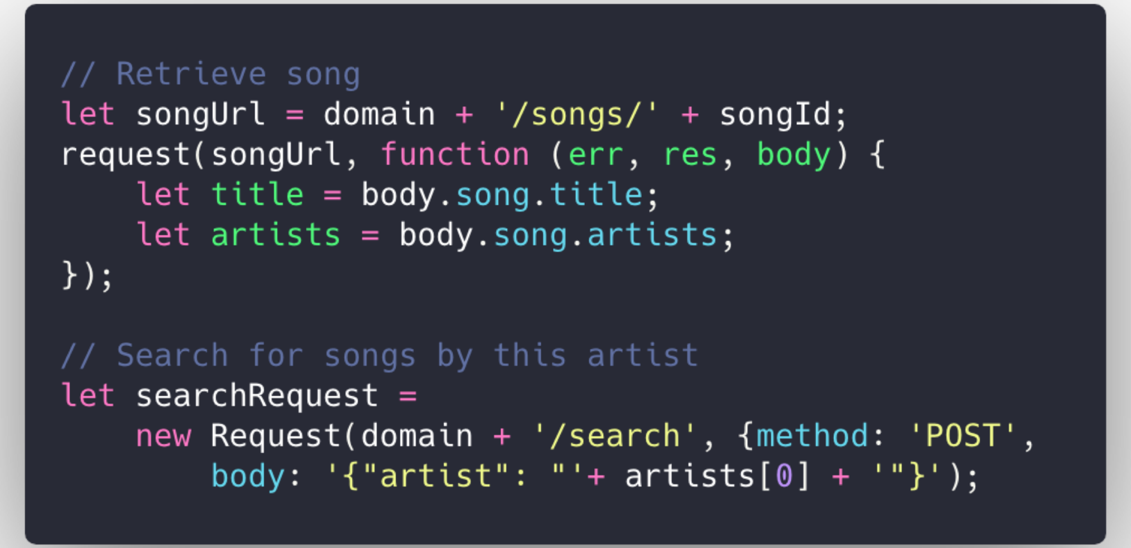
Participatory Design
Low Fidelity – Blank Paper Exercise


Design the interface you want to use
Frontend Developer
home
customer
next
/item
back
Shipping Service Sketch
Shipping Service Machine Interface Model

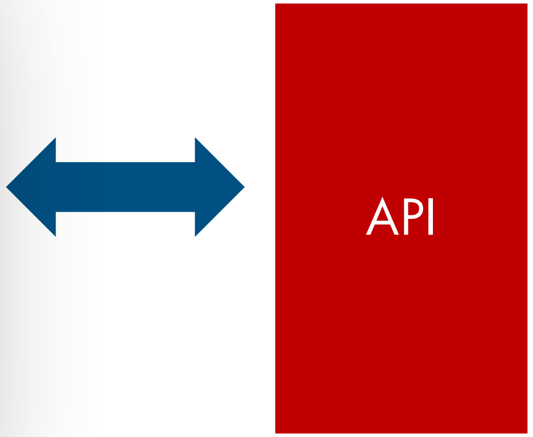
Technique #6
Choose a Style That Fits

"REST" (The CRUD Style)

GET
PUT
POST
DELETE



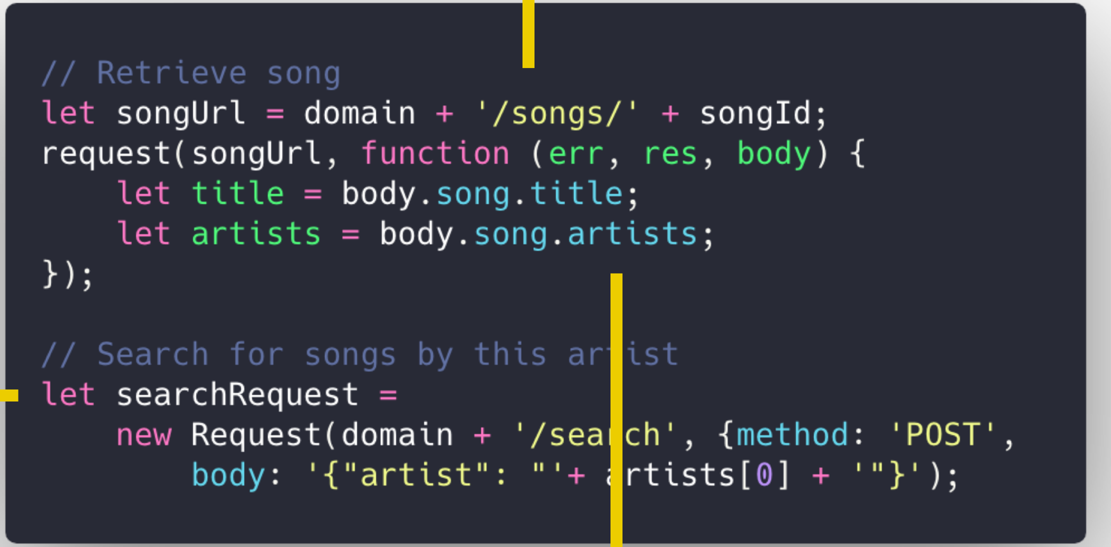
```
// Retrieve song
let songUrl = domain + '/songs/' + songId;
request(songUrl, function (err, res, body) {
    let title = body.song.title;
    let artists = body.song.artists;
});

// Search for songs by this artist
let searchRequest =
    new Request(domain + '/search', {method: 'POST',
        body: '{"artist": "'+ artists[0] + '"}');
```



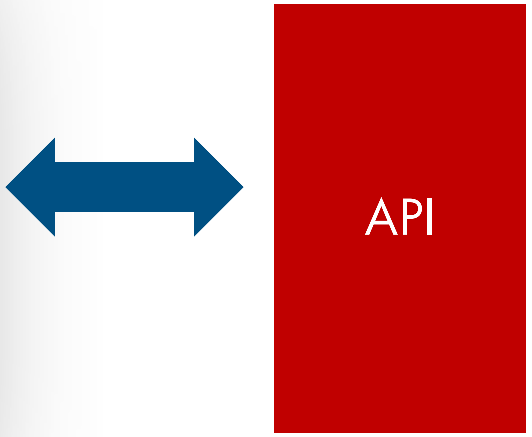
"REST" (The CRUD Style)

Shared understanding of object address space

Some RPC endpoints



```
// Retrieve song
let songUrl = domain + '/songs/' + songId;
request(songUrl, function (err, res, body) {
    let title = body.song.title;
    let artists = body.song.artists;
});

// Search for songs by this artist
let searchRequest =
    new Request(domain + '/search', {method: 'POST',
        body: '{"artist": "'+ artists[0] + '"}');
```



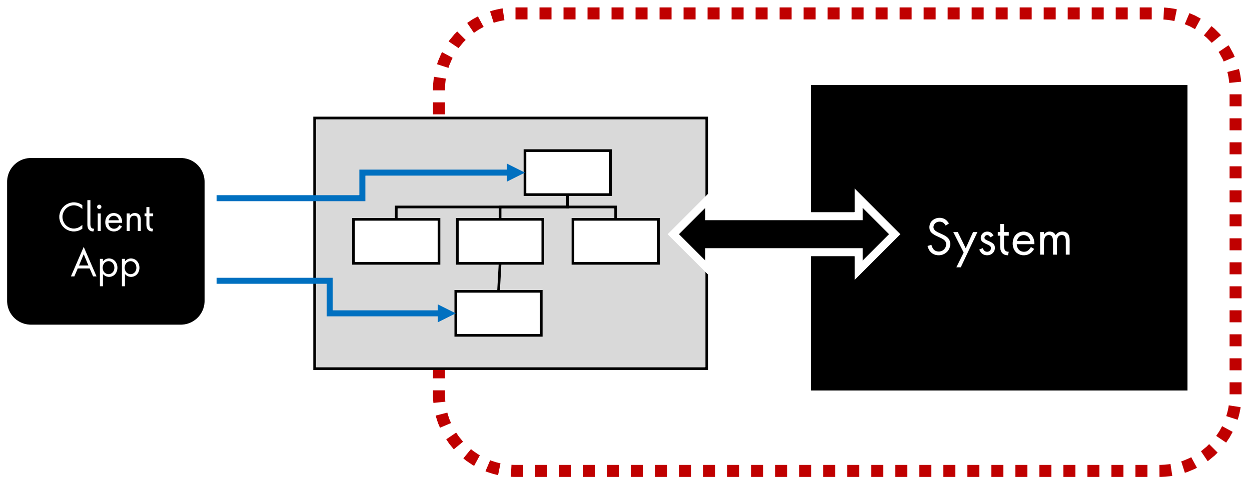
Shared understanding of data model

"REST" (The CRUD Style)

The API is a nested set of "CRUD"able objects

Interface design is "crafted"

You design the objects, relationships and query model

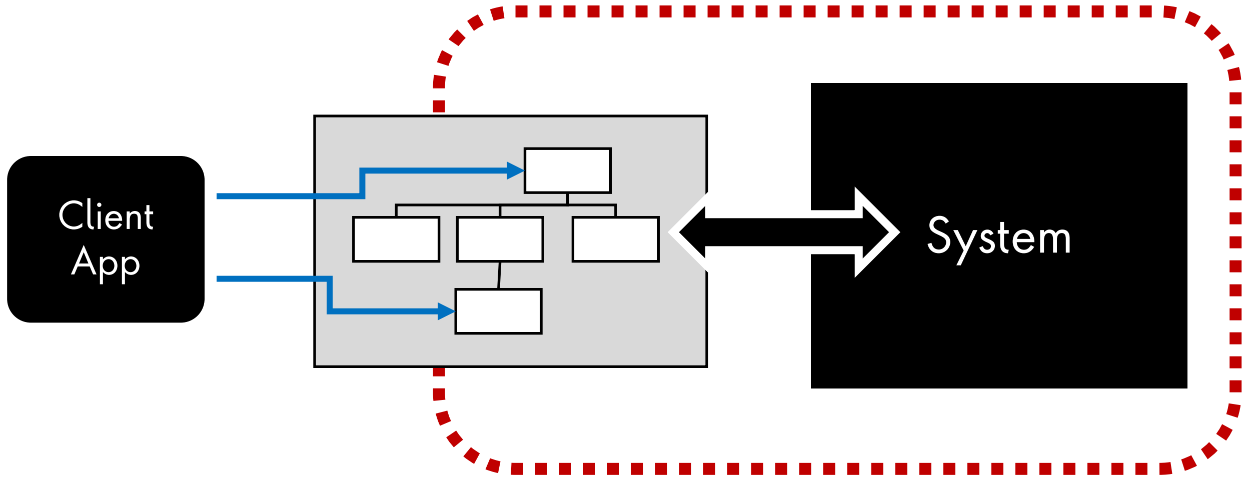


"REST" (The CRUD Style): Cost Impacts

Increases user learning costs
(crafted API)

Increases design costs
(crafted API)

Increases cost of future changes
(coupling to data model and address space)



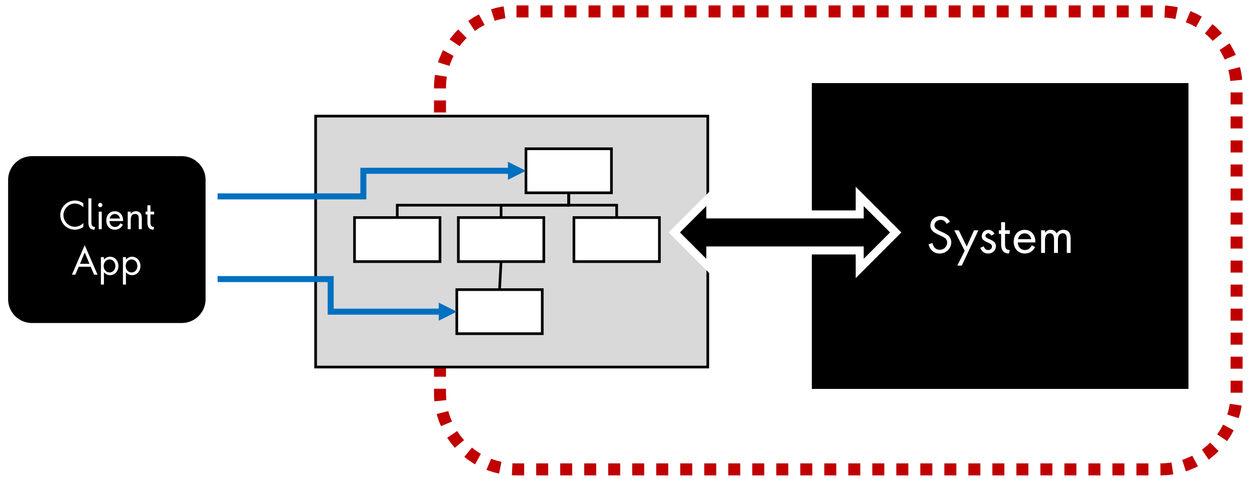
"REST" (The CRUD Style): When I Like To Use It

GET
PUT
POST
DELETE

When I want to deliver a conventional API experience

When I need to provide an easily usable interface

When I'm targeting client developers who are not in our team/organization



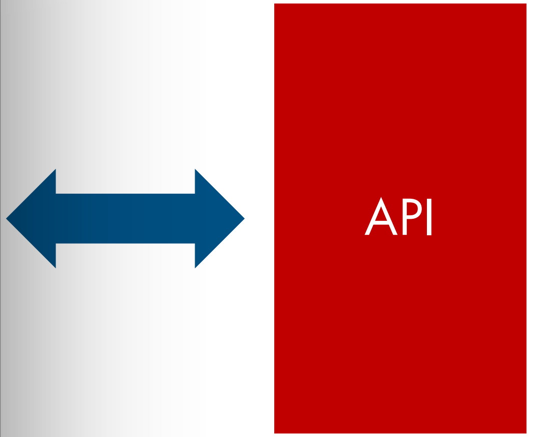
GraphQL (The Query Style)

?

Fixed
RPC
endpoint

Shared
Data Model

```
const queryEndpoint = domain + "/graphql";

const query = "{
    Customer(id: $id) {
        accountInformation
        balance
        address {
            postcode
        }
    }
}"

const body = '{
    "query: "' + query +
    "variables: { "id": "' + customerID + '"}
}

let QueryRequest =
  new Request(queryEndpoint, {method: 'POST', body: body});
```



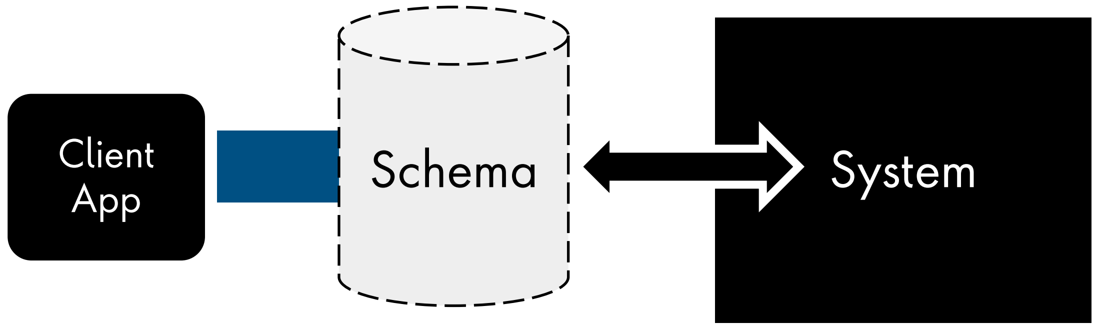
GraphQL (The Query Style)

?

The API is a data source

Interface design is standardized

You design the data model and the RPC endpoints



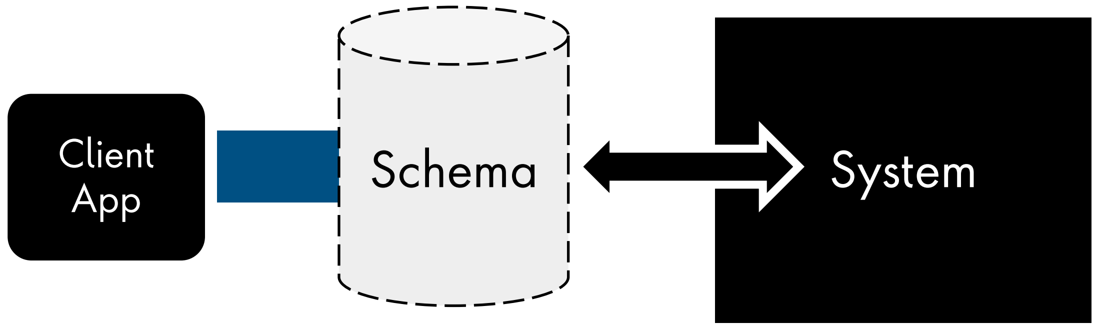
GraphQL (The Query Style): Cost Impacts

?

Increases learning costs
(understand data model)

Increases engineering costs
(data pipe architecture)

Increases cost of future changes
(coupling to data model)

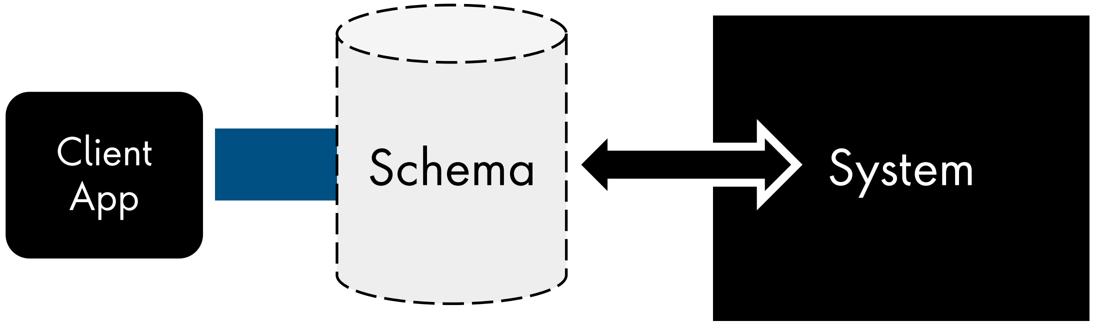


GraphQL (The Query Style): When I Like To Use It

When the client developers are in in my team

When my client developers need greater flexibility and autonomy

When I want to present something new to my users



API Styles – User Metaphors

Tunnel-RPC Style	The API is a local library
CRUD Style	The API is a set of data objects
Hypermedia Style	The API is a website
Query Style	The API is a database
Event Driven Style	The API is a notification message

Technique #7
Make Practical Design Decisions

Example

“What should we return when `GET /songs?genre=classical` doesn’t produce a match?”

Resolving API Design Decisions

1. How <u>reversible</u> is this design decision?

If its easy to reverse we can afford to make a less optimal decision and improve it later.
This is debt that is easy to pay back.

"Once we decide on this, it's going to be difficult to change.
We'd have to release a new version."

Resolving API Design Decisions

2. **What do the specifications and standards say?**

If there are clear rules, endeavor to follow them.

"We've read RFC 7231, now we are starting to think 404 is the way to go."

Resolving API Design Decisions

3. **What would the client code look like?**

Write client code to test your hypothesis and gain insight

"Actually, now it seems like a 200 with an empty collection makes the most sense!"

Seven API Design Techniques

1. Manage Your Debt
2. Build a Conventional Product (when it makes sense)
3. Perform Heuristic Evaluations
4. Write Code
5. Use Participatory Design
6. Choose a style that fits
7. Make Practical Design Decisions

Bill Moggride on Design

"If there's a simple, easy design principle that binds everything together, it's probably about starting with the people"

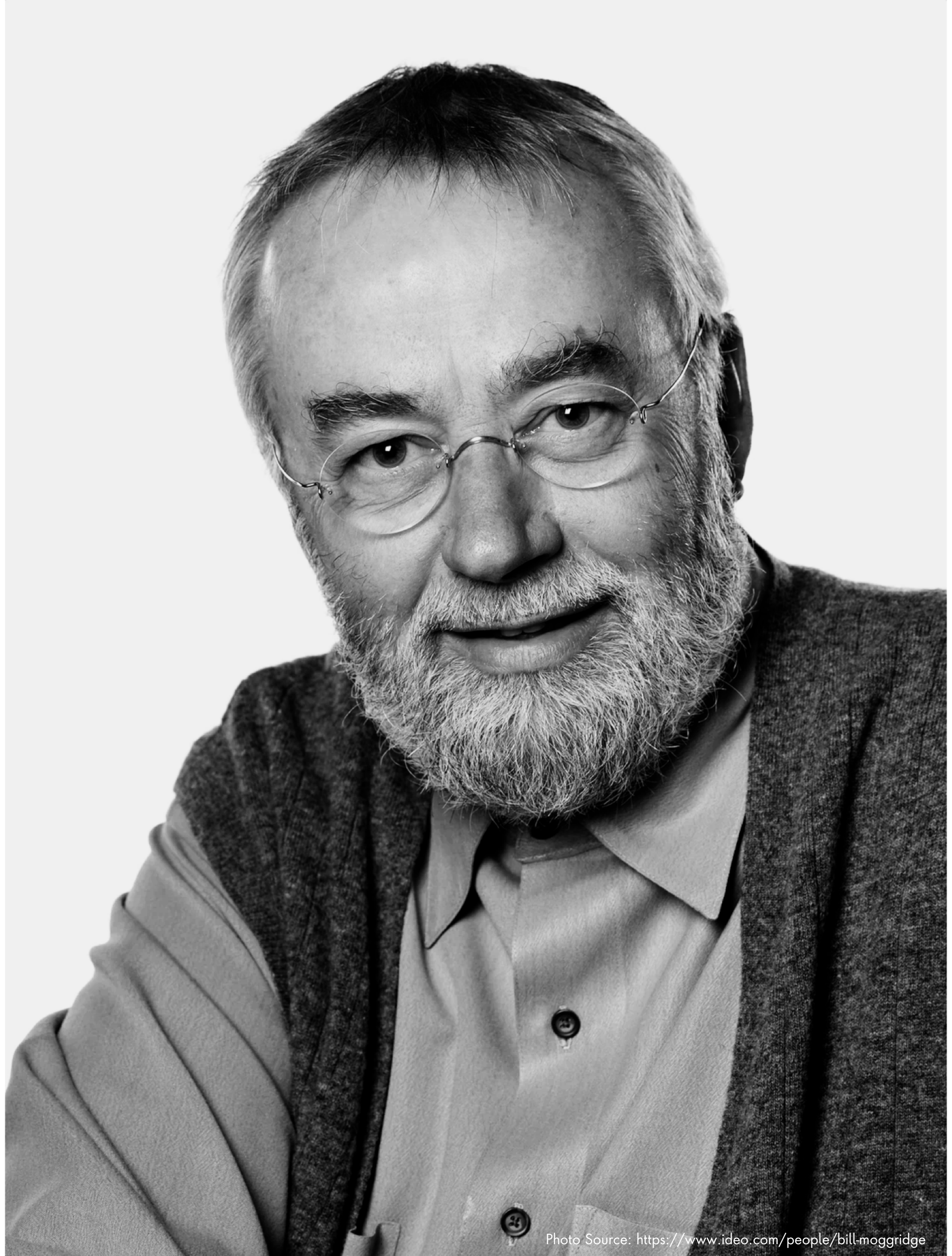


Photo Source: https://www.ideo.com/people/bill-moggridge

June 2019

Practical API Design

Ronnie Mitra

ronnie.mitra@publicissapient.com
@mitraman