GEONOVUM

Kennisplatform API's speciale editie
https://www.geonovum.nl/agenda/kennisplatform-apis-speciale-editie

API First with
"Patterns for API Design": Your Personal API Design Process?

Dr. Olaf Zimmermann, Dr. Daniel Lübke

olaf.zimmermann@unisg.ch
daniel.luebke@digital-solution-architecture.com

Slides

Digital Solution **Architecture**

University of St.Gallen

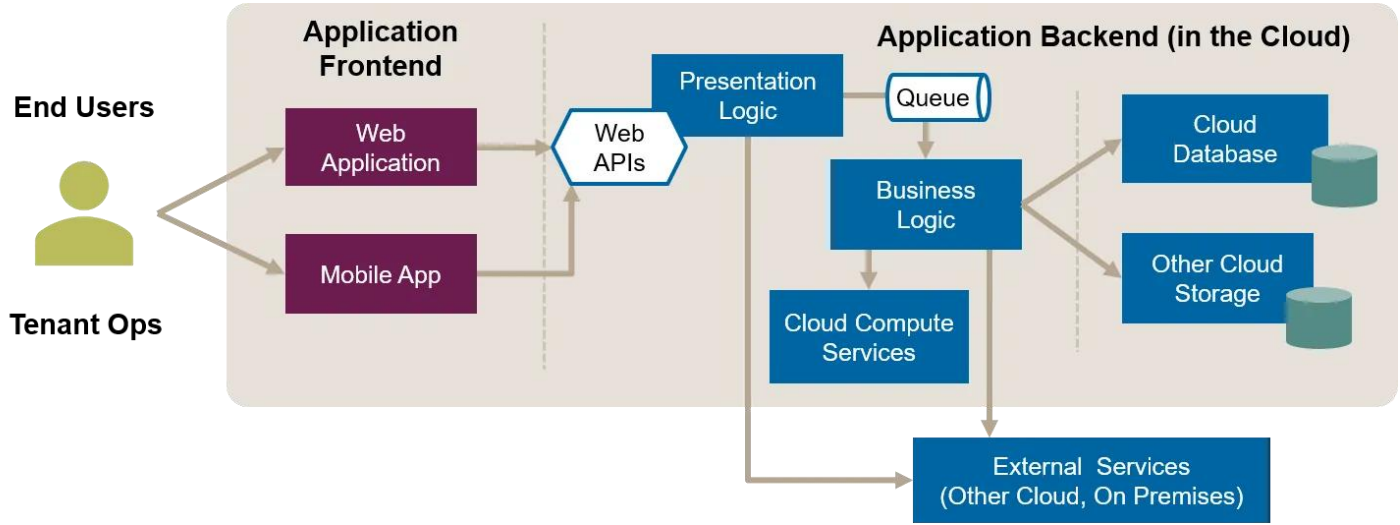
Key Messages

- **API design** is hard… problems and concepts stay while technologies come and go
- Patterns make design **knowledge** timeless – but **concrete and actionable** still
- We mined 44 API design **patterns** from personal experience and community insights
- These **"Patterns for API Design"** support API first practices nicely
- They complement **API-first design processes** such as ADDR

- *Agenda for afternoon sessions:*
  - Top 5 patterns deep dive, interest-driven and interactive
  - Design space discussion on Asynchronous APIs: Questions, Options, Criteria (QOC)

API Design is Hard…
a Wicked Problem

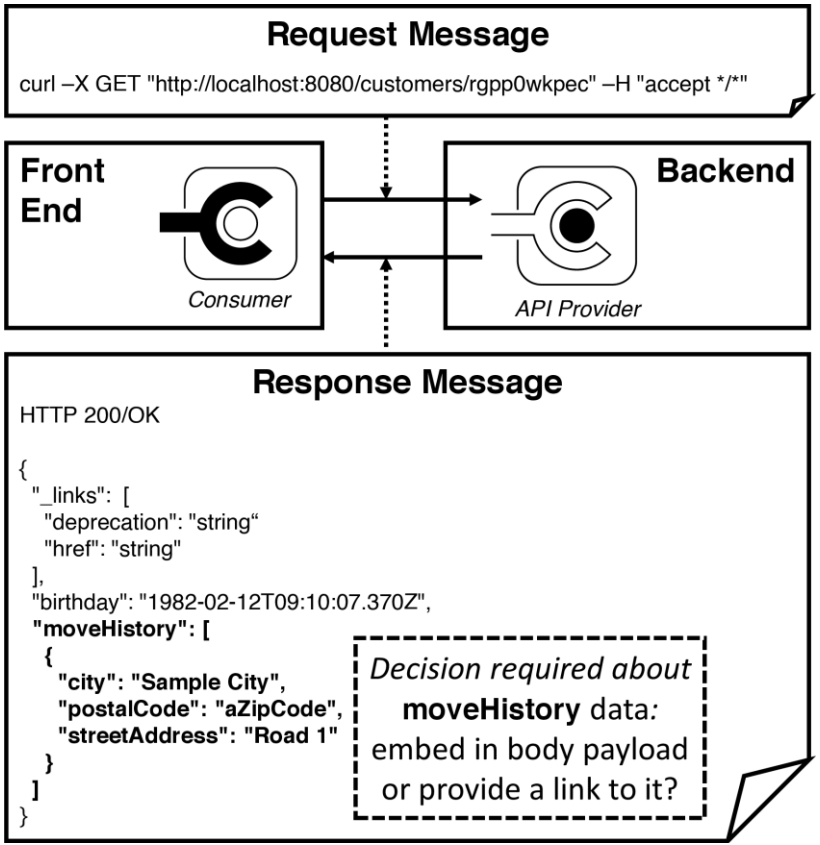
Context/Example: Frontend-Backend Integration



https://medium.com/olzzio/what-is-a-cloud-native-application-anyway-part-1-8241e9c71a62

Web API Example

- API clients send requests and API providers send responses to them
  - Often JSON over HTTP
  - Many other protocols and formats (old and new)
- Data contracts matter
  - Often part of OpenAPI Specification (OAS)
  - JSON Schema
  - Protocol Buffers



Reference: Figure 4.1 in Patterns for API Design, Addison Wesley 2022

Help, I need to design an API!

What are the important questions and driving forces?

Existing and New Capabilities

Complexity of systems

Emerging Technologies

Quality goals, security needs

Design Forces (aka Daily Business)

Project pressure, legacy system constraints

Feature Change Requests

Change dynamics

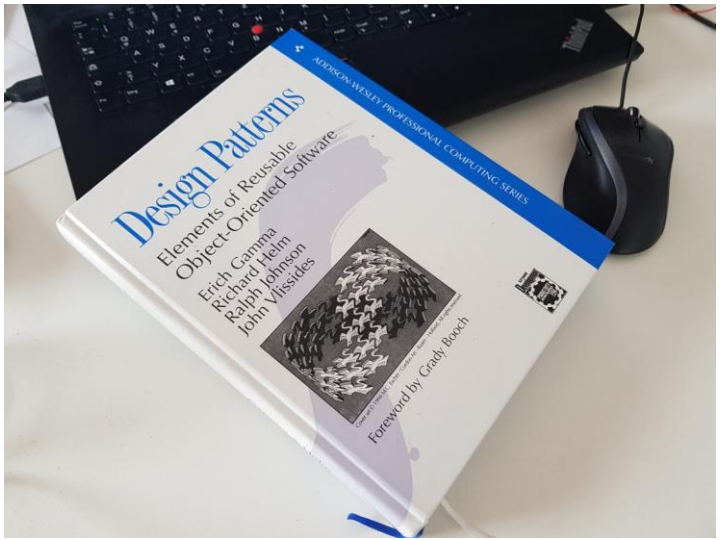
Guidelines (Corporate, Community)

“Patterns for API Design”: Motivation and Overview

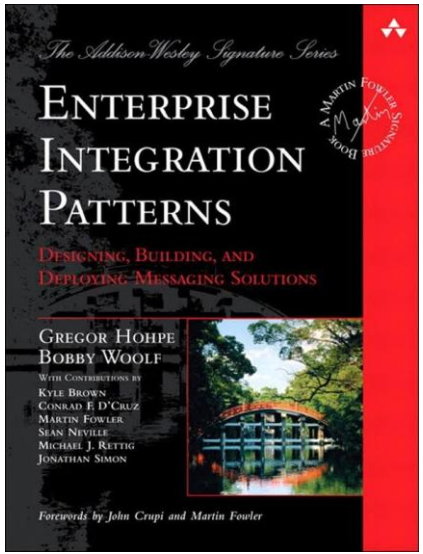
Why Patterns?

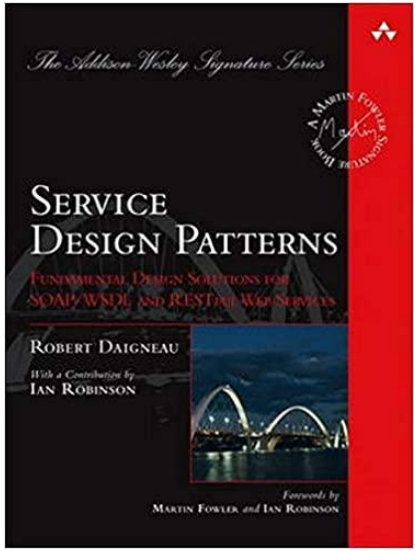
Patterns collect and document community experience – proven solutions to common, recurring problems

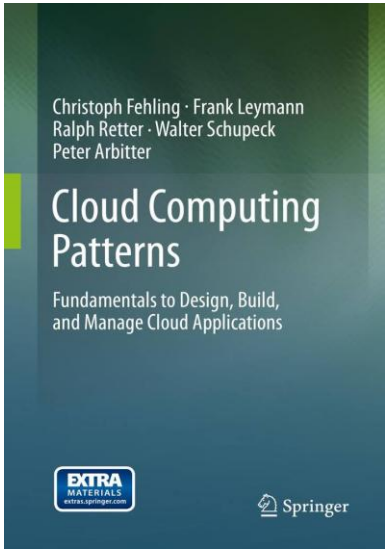
- Different pattern templates, themes:
  - *Name*, Icon
  - *Context*: Intent, motivation and applicability
  - *Solution* structure and its *forces*
  - *Consequences*: Benefits and liabilities
  - *Examples* and implementation hints
  - Pointers to *related patterns*
  - *Known uses*
- Community processes/practices:
  - Shepherding (coaching)
  - Writers' workshops
  - Ruel of three (known uses)

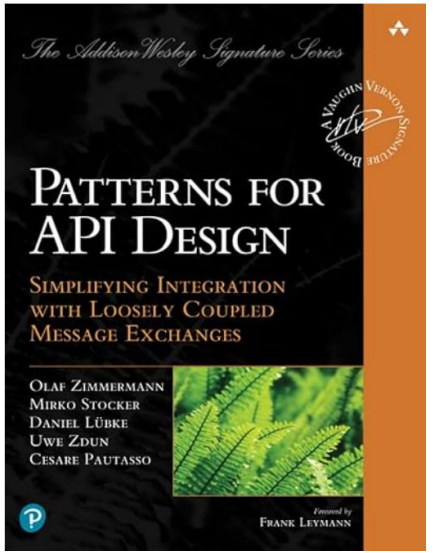


Existing Patterns Relevant for Remote API Design







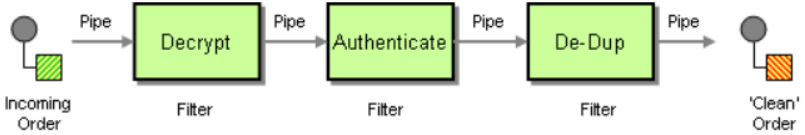


Patterns abstract and generalize to sustain; they are mined, not invented

Pattern Example 1: Pipes and Filters

https://www.enterpriseintegrationpatterns.com/patterns/messaging/PipesAndFilters.html

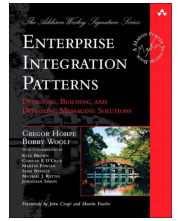
How can we perform complex processing on a message while maintaining independence and flexibility?



Use the *Pipes and Filters* architectural style to divide a larger processing task into a sequence of smaller, independent processing steps (Filters) that are connected by channels (Pipes).

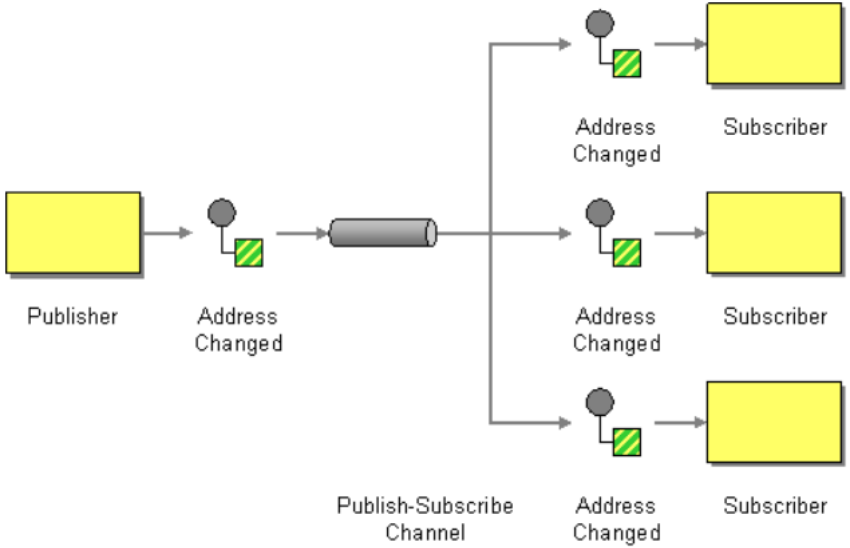
- **Forces:**
  - Separation of concerns, reuse and (de-)composition, concurrency
- **Known uses:**
  - UNIX, application integration flows, Extract-Transform-Load (ETL)
  - Many products and open source projects for these and other usage scnearios

Pattern Example 2: Publish-Subscribe



An application is using *Messaging* to announce events.

How can the sender broadcast an event to all interested receivers?

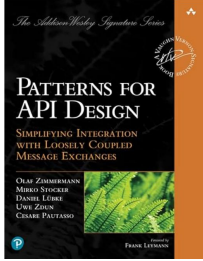


- **Forces:**
  - Coupling dimensions
  - Message semantics
- **Known uses:**
  - Messaging systems such as ActiveMQ and RabbitMQ
  - Apache Kafka
  - Managed cloud services (AWS, Azure, …)

Send the event on a *Publish-Subscribe Channel*, which delivers a copy of a particular event to each receiver.

https://www.enterpriseintegrationpatterns.com/patterns/messaging/PublishSubscribeChannel.html

Our Pattern Categories and Questions Answered



Foundation Patterns

- Which types of (sub-)systems and components are integrated?
- From where should an API be accessible?
- How should it be documented?

Responsibility Patterns

- What is the architectural role played by each API endpoint?
- How do these roles and the operation responsibilities realizing them impact (micro-)service cuts and granularity?

Structure Patterns

- What is an adequate number of representation elements for request and response messages?
- How should these elements be structured?
- How can they be grouped and annotated with usage instructions?

Quality Patterns

- How can an API provider achieve a certain level of quality of the offered API, while using its available resources in a cost-effective way?
- How can the quality tradeoffs be communicated and accounted for?

Evolution Patterns

- How to deal with lifecycle management concerns such as support periods and versioning?
- How to promote backward compatibility and communicate breaking changes?

https://api-patterns.org/categories

Meet the Authors

(and the Book Content)

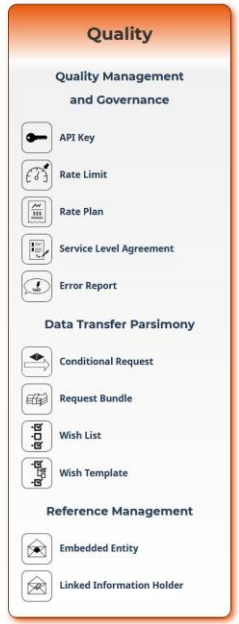
Olaf Zimmermann

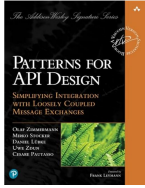
Mirko Stocker

Daniel Lübke

Uwe Zdun

Cesare Pautasso

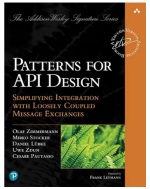




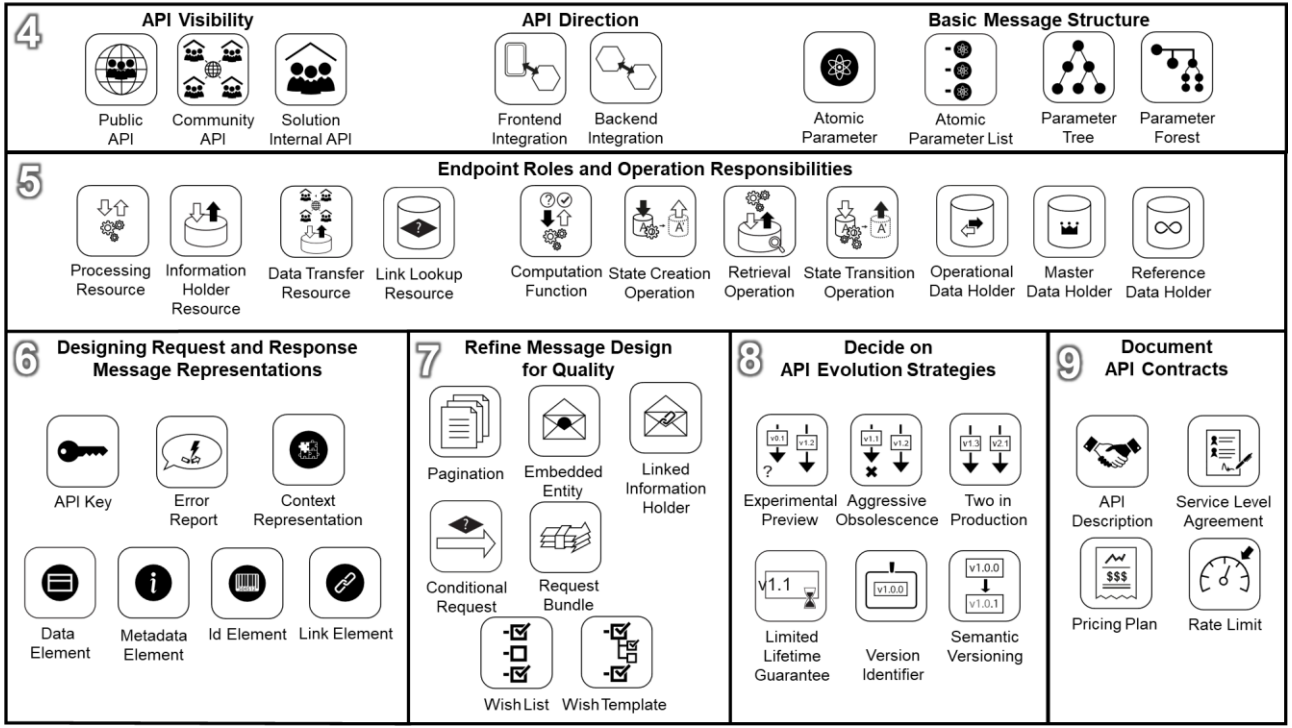
https://api-patterns.org

https://api-patterns.org/

Selected Patterns from the Book (and the Web)



Patterns as a Design Guide and/or Checklist

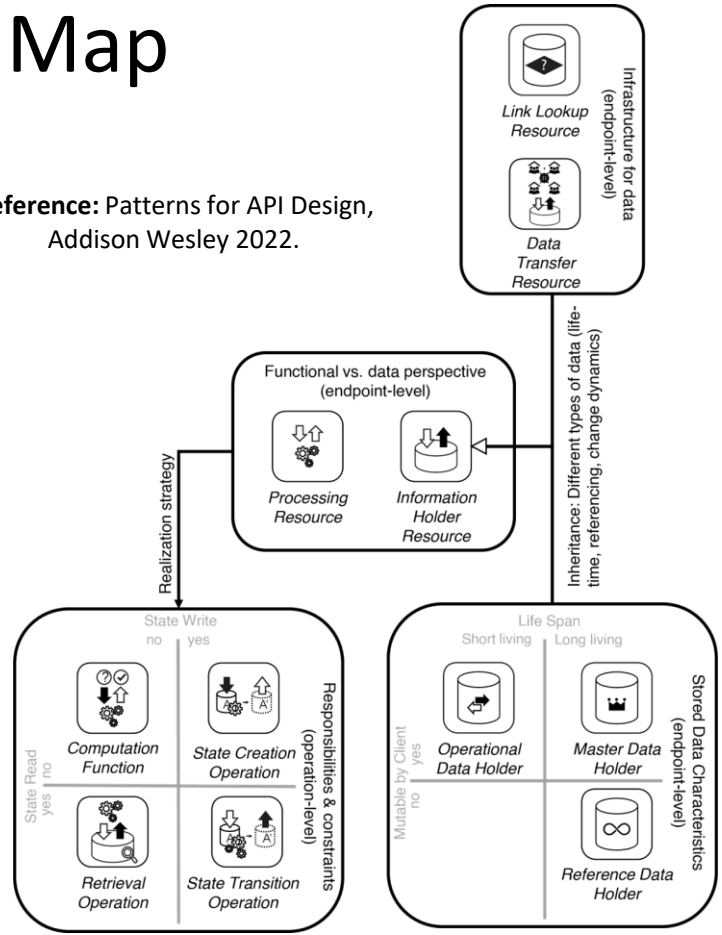


Responsibility Patterns Map

- Which architectural role should an API endpoint play?
- What is the responsibility of each API operation?

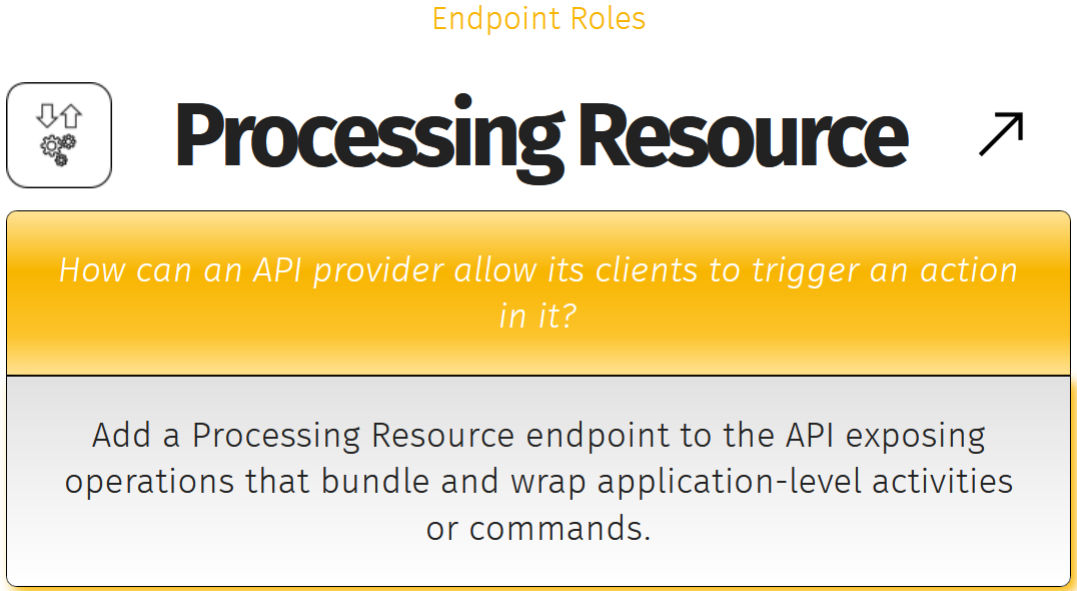
https://eprints.cs.univie.ac.at/6520/1/MAP-EuroPlop2020aPaper.pdf

Reference: Patterns for API Design, Addison Wesley 2022.



Processing Resource

- **Context**
  - Activity-oriented semantics of story
- **Forces**
  - Contract expressiveness and service granularity (and their impact on coupling)
  - Learnability and manageability
  - Semantic interoperability
  - Response time
  - Security and privacy
  - Compatibility and evolvability

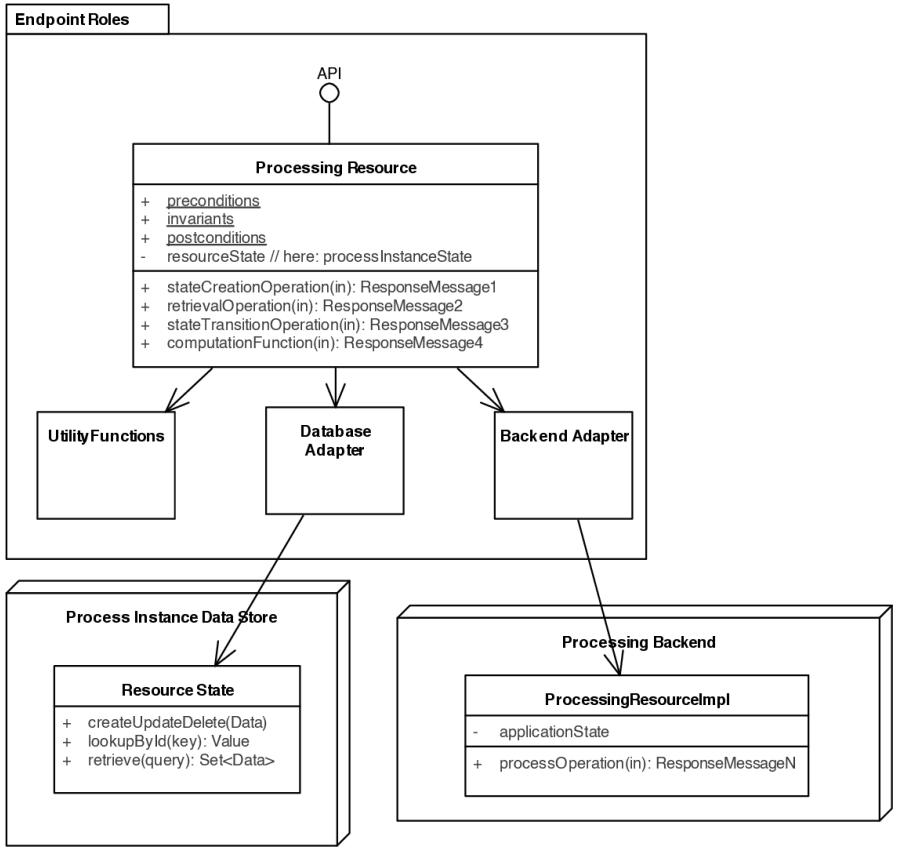


https://api-patterns.org/patterns/responsibility/endpointRoles/ProcessingResource.html

Reference: Patterns for API Design, Addison Wesley 2022.

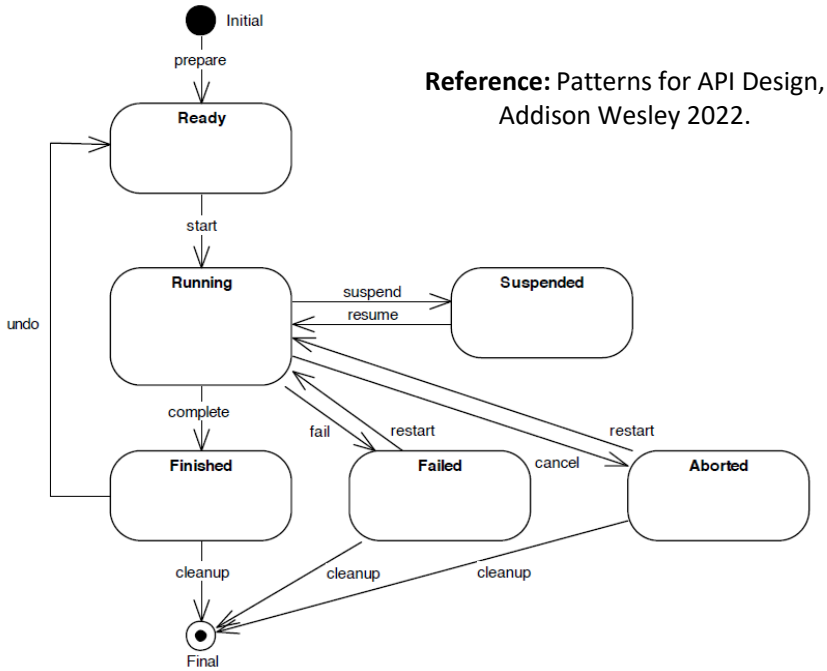
Processing Resource

- **Design issues:**
  - Backend interface
  - Transaction management
  - Idempotency
- **Implementation hints:**
  - Framework support available
  - Domain-driven design as input
  - Testing matters



Frontend BPM vs. BPM-as-a-Service

- Business Activity Processor variant of State Transition Operation pattern
  - Book, website archive, EuroPLoP 2020 paper
- Each API operation realizes one state transition, whose mileage may wary:
  - Entire business process
  - Single activity
- General state machine to be adapted according to domain requirements
  - E.g. DDD, Context Mapper DSL and tools
- Technology mapping:
  - HTTP POST, PUT, PATCH
  - Mutations in GraphQL



Reference: Patterns for API Design, Addison Wesley 2022.

Information Holder Resource

	**PATTERN: INFORMATION HOLDER RESOURCE**
Problem	How can domain data be exposed in an API, but its implementation still be hidden? How can an API expose data entities so that API clients can access and/or modify these entities concurrently without compromising data integrity and quality?
Solution	Add an Information Holder Resource endpoint to the API, representing a data-oriented entity. Expose create, read, update, delete, and search operations in this endpoint to access and manipulate this entity. In the API implementation, coordinate calls to these operations to protect the data entity.

Forces:

- *Modeling approach* and its impact on coupling
- *Quality attribute conflicts and trade-offs* such as concurrency, consistency; data quality and integrity; recoverability and availability; mutability and immutability
- *Security*
- *Data freshness versus consistency*
- *Compliance with architectural design principles*

https://api-patterns.org/patterns/responsibility/endpointRoles/InformationHolderResource

Operational Data Holder

Problem

How can an API support clients that want to create, read, update, and/or delete instances of domain entities that represent operational data: data that is rather short-lived, changes often during daily business operations, and has many outgoing relations?

Forces

- *Processing speed* for content read and update operations
- Business *agility* and schema update *flexibility*
- *Conceptual integrity and consistency* of relationships

Solution

Tag an INFORMATION HOLDER RESOURCE as OPERATIONAL DATA HOLDER and add API operations to it that allow API clients to create, read, update, and delete its data often and fast.

Optionally, expose additional operations to give the OPERATIONAL DATA HOLDER domain-specific responsibilities. For instance, a shopping basket might offer fee and tax computations, product price update notifications, discounting, and other state-transitioning operations.

https://api-patterns.org/patterns/responsibility/informationHolderEndpointTypes/OperationalDataHolder

Master Data Holder

Known uses:

- Customer Relationship Management
- Product Inventory

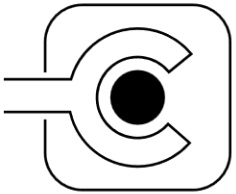
How can I design an API that provides access to master data that lives for a long time, does not change frequently, and will be referenced from many clients?

Forces:

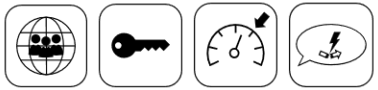
- Master data *quality*
- Master data *protection*
- *Data under external control*, for instance master data management systems

Mark an INFORMATION HOLDER RESOURCE to be a dedicated MASTER DATA HOLDER endpoint that bundles master data access and manipulation operations in such a way that the data consistency is preserved and references are managed adequately. Treat delete operations as special forms of updates.

https://api-patterns.org/patterns/responsibility/informationHolderEndpointTypes/MasterDataHolder



Google Calendar API v3



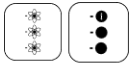
GET

Events: list

`https://www.googleapis.com/calendar/v3/calendars/`**`calendarId`**`/events`

Request

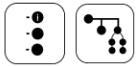
calendarId (path parameter)



Optional request parameters

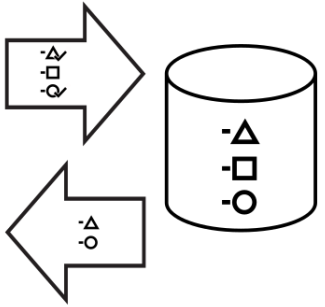
-  timeMin
-  maxResults
-  orderBy
-  pageToken

Response



-  etag
-  nextPageToken
-  items
  - 

Wish List Pattern (1/2)



- **Problem:**
  - *How can an API client inform the API provider at runtime about data that it is interested in?*
- **Forces:**
  - Client diversity, message size vs. number of messages
  - Endpoint complexity
  - And more

https://api-patterns.org/patterns/quality/dataTransferParsimony/WishList

- **Solution:**
  - *As an API client, provide a WISH LIST in the request that enumerates all desired data elements of the requested resource.*

Wish List Pattern (2/2)

- **Known uses:**
  - Found in many Web and product APIs, e.g. Atlassian Jira
- **Variations:**
  - Expansion, wild cards (*), query expression (GraphQL!)
- **Alternative:**
  - Wish Template (structured, mock object rather than flat name list)

```
curl -X GET
http://localhost:8080/customers/gktlipwhjr?fields=
customerId,birthday,postalCode

{
 "customerId": "gktlipwhjr",
 "birthday": "1989-12-31T23:00:00.000+0000",
 "postalCode": "8640"
}
```

Error Report

https://api-patterns.org/patterns/structure/specialPurposeRepresentations/ErrorReport

```
curl -i -X POST \
  --header "Content-Type: application/json" \
  --data '{"username":"xyz","password":"wrong"}' \
  http://localhost:8080/auth
```

```
HTTP/1.1 401
Content-Type: application/json;charset=UTF-8
Date: Wed, 20 Jun 2018 08:25:10 GMT

{
  "timestamp" : "2018-06-20T08:25:10.212+0000",
  "status" : 401,
  "error" : "Unauthorized",
  "message" : "Access Denied",
  "path" : "/auth"
}
```

Special Purpose Representations

Error Report

How can an API provider inform its clients about communication and processing faults? How can this information be made independent of the underlying communication technologies and platforms (for example, protocol-level headers representing status codes)?

Therefore:

> *Reply with an error code in response messages that indicate and classify the faults in a simple, machine-readable way. In addition, add a textual description of the error for the API client stakeholders, including developers and/or end users such as administrators.*

Known Use/Application Example: RFCs 9457 7807

Problem Details for HTTP APIs

Internet Engineering Task Force (IETF)
Request for Comments: 9457
Obsoletes: 7807
Category: Standards Track
Published: July 2023
ISSN: 2070-1721

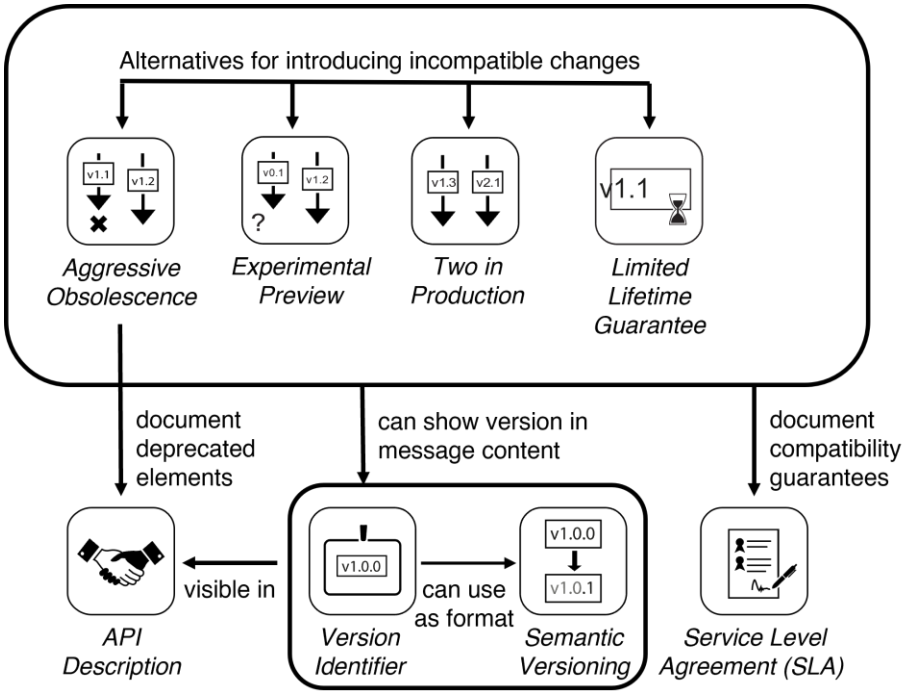

```
HTTP/1.1 403 Forbidden
Content-Type: application/problem+json
Content-Language: en

{
 "type": "https://example.com/probs/out-of-credit",
 "title": "You do not have enough credit.",
 "detail": "Your current balance is 30, but that costs 50.",
 "instance": "/account/12345/msgs/abc",
 "balance": 30,
 "accounts": ["/account/12345",
              "/account/67890"]
}
```

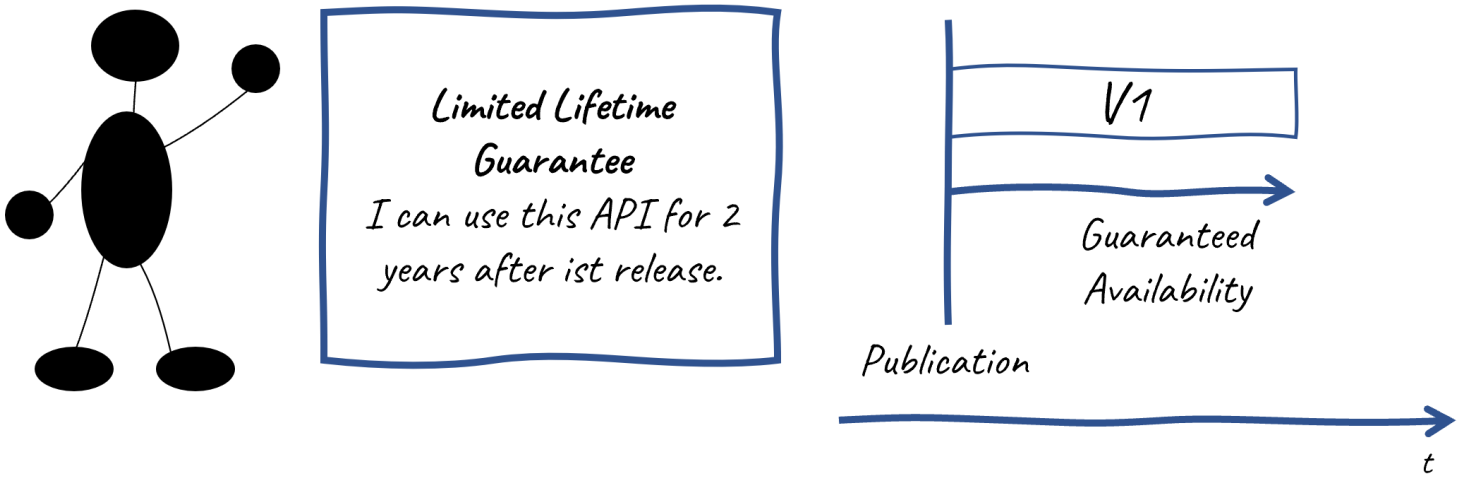
https://datatracker.ietf.org/doc/html/rfc9457

Evolution Patterns

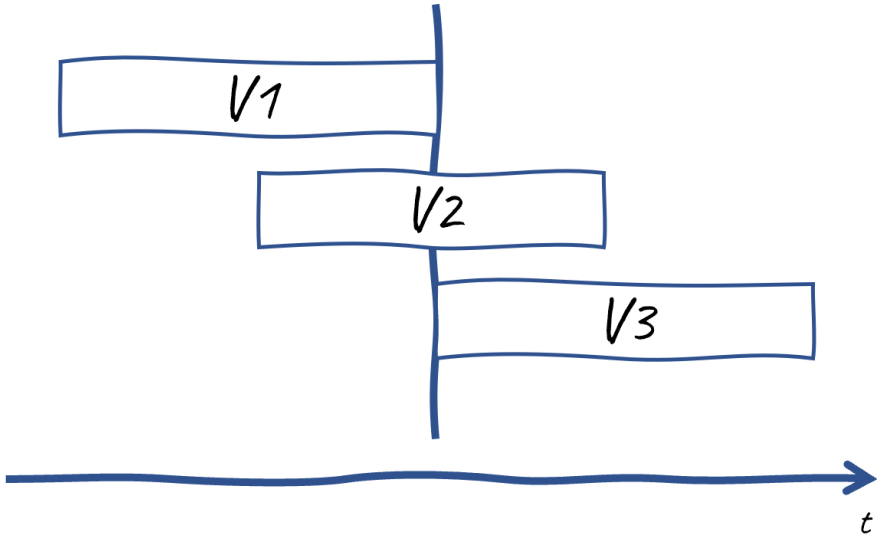
Reference: Patterns for API Design, Addison Wesley 2022.



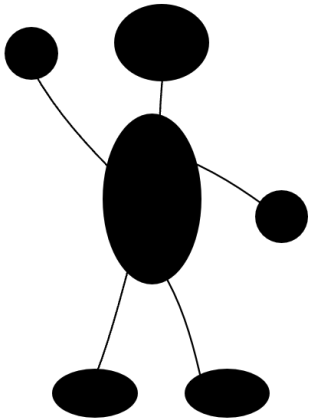
Limited Lifetime Guarantee



Two in Production



Two In Production
I can use this API until the version after next is released.



Pattern Adoption Story

https://api-patterns.org/news

Pattern Adoption Story: Dutch Government & Energy Sector

15 Mar 2023

We are delighted to present our first web-exclusive pattern adoption story, contributed by our reader Ton Donker.

In the pattern story Dutch Government & Energy Sector, Ton shares some other well-known uses of the Wish List, Pagination, and Error Report patterns, as well as additional discussion points and recommended reading.

The Wish List pattern is widely used in Dutch government APIs. One implementation hint is to distinguish between different variants of Metadata Elements by prefixing them in the query string:

To distinguish between 'real' query parameters and the more 'steering' or 'meta' parameters like expand, fields, limit and page, we – the Dutch Energy sector API Working Group – advocate the usage of an underscore prefix, so _fields, _expand, _limit and _page to prevent misinterpretation of the function of the query parameters.

This pattern adoption story was contributed by Ton Donker, Ruben Haasjes and their readers group. It covers the 21 patterns presented as API Design Pattern of the Week on LinkedIn and Medium as well as other patterns. This page features part 1 of 5 (part 2, part 3, part 4, part 5).

https://api-patterns.org/book/pattern-adoption-story-1

API First
(with ADDR and PfAD)

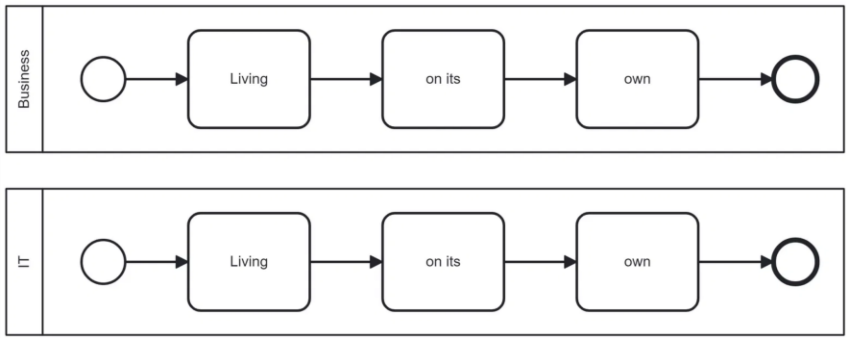
API First

- “API-first organizations develop APIs before writing other code, instead of treating them as afterthoughts.” https://www.postman.com/api-first/
- Develop API Contract in different state of mind
- Do not accidentally expose internals

APIs Model Business Capabilities

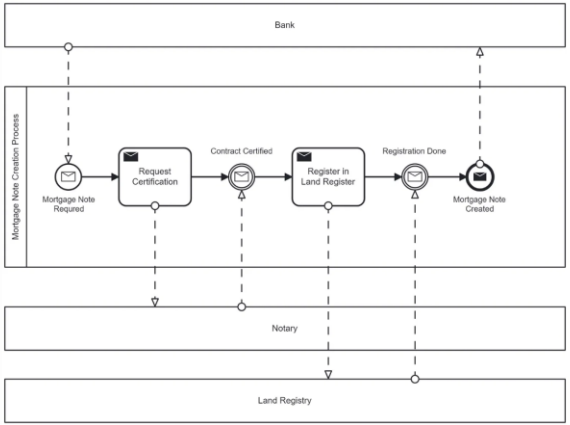
The Alignment Problem

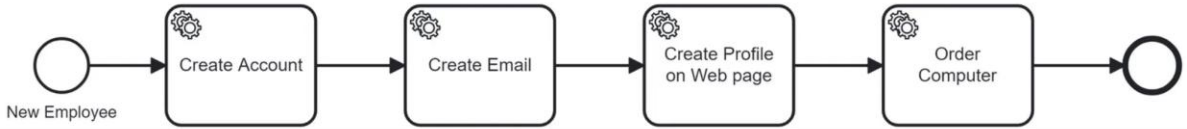
- Business talks in processes and outcomes
- IT talks in systems, services and APIs
- Missing shared context results in fragmented systems and duplicate APIs



Every API Lives Inside (at least) a Process

- Processes define **when** and **why** APIs are called
- Clarify data ownership, order, and responsibility
- Process context prevents building "accidental APIs"





Business processes are the top in top-down design

Of course, we learned to combine top-down and bottom-up
bottom-up design

… to APIs

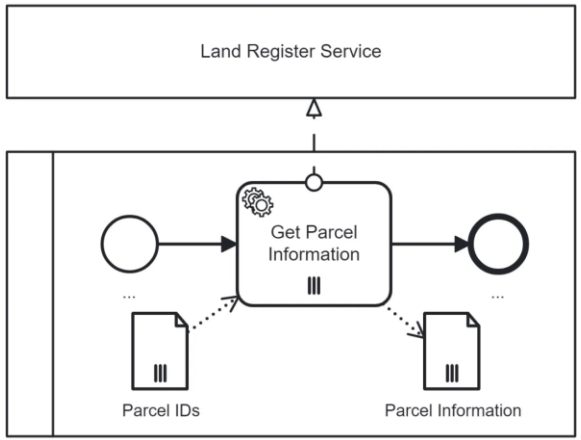
- **Likely candidates for APIs are:**
  - Service Tasks
  - Sub-Processes
  - Any transition between different participants
- **May be synchronous, event-driven, message-based, AI, …**

Single Objects vs Request Bundles

- Processes reveal whether you handle single entities or batches
- Example: One invoice vs Invoice Batch upload
- API design must reflect that granularity

Pattern: Request Bundle

How can the number of requests and responses be reduced to increase communication efficiency?




```
for parcelId in ParcelIds {
  parcelInfos. add (
    landRegisterService. getParcelInformation (parcelId)
  );
}
```

https://api-patterns.org/patterns/quality/dataTransferParsimony/RequestBundle

Domain-Driven API Design

Reference: Singjai, A.; Zdun, U.; Zimmermann, O.; Pautasso, C.: *Patterns on Deriving APIs and their Endpoints from Domain Models*, Proc. of EuroPloP 2021 (PDF)

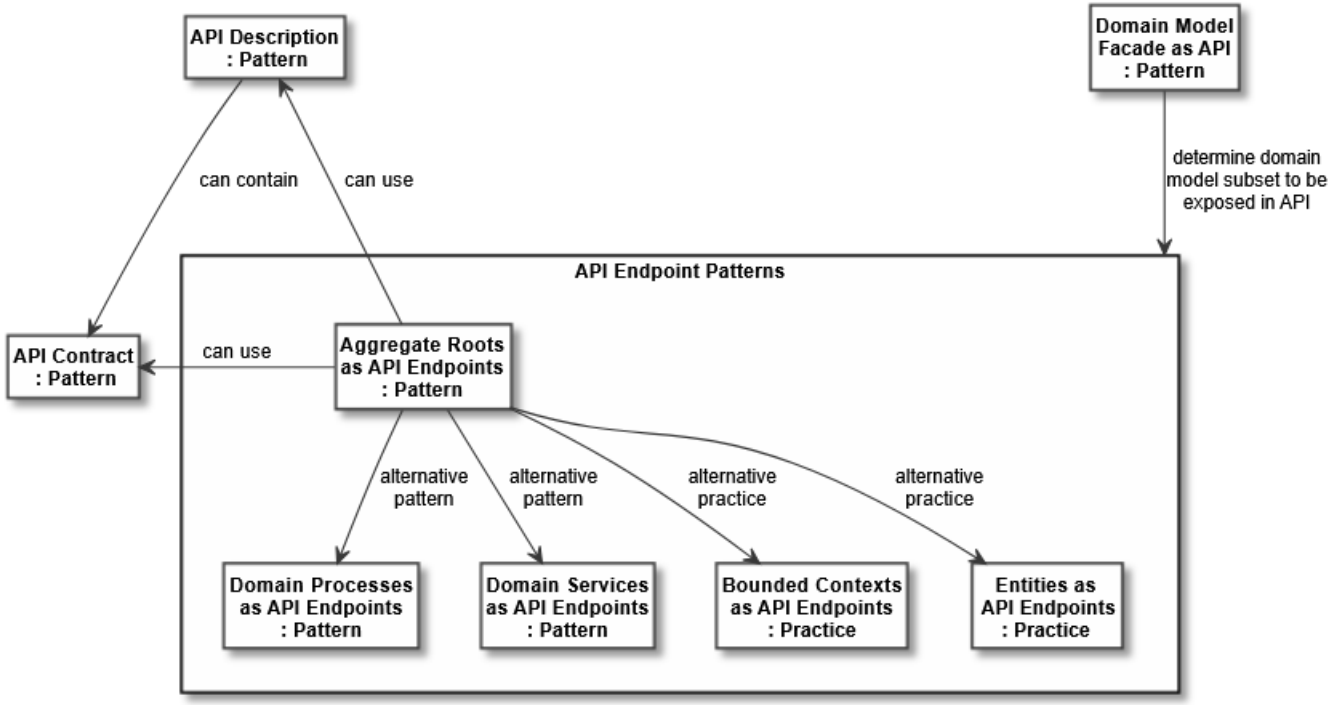
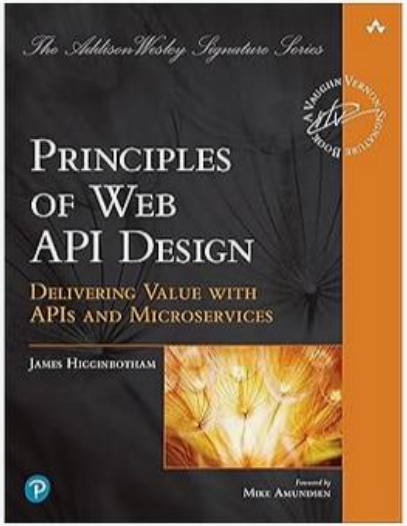


Figure 4: Overview of the patterns for how to derive API endpoints from domain model elements

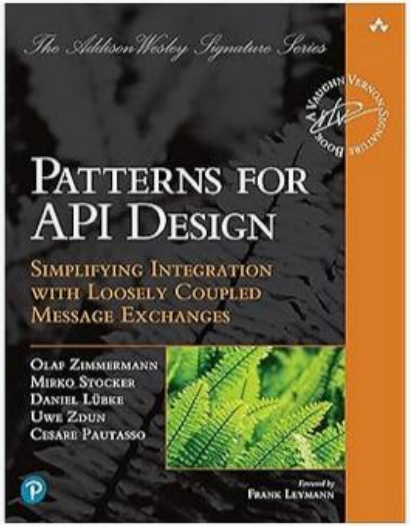
What is the development process?

(In which order should I do things?)

Frequently bought together



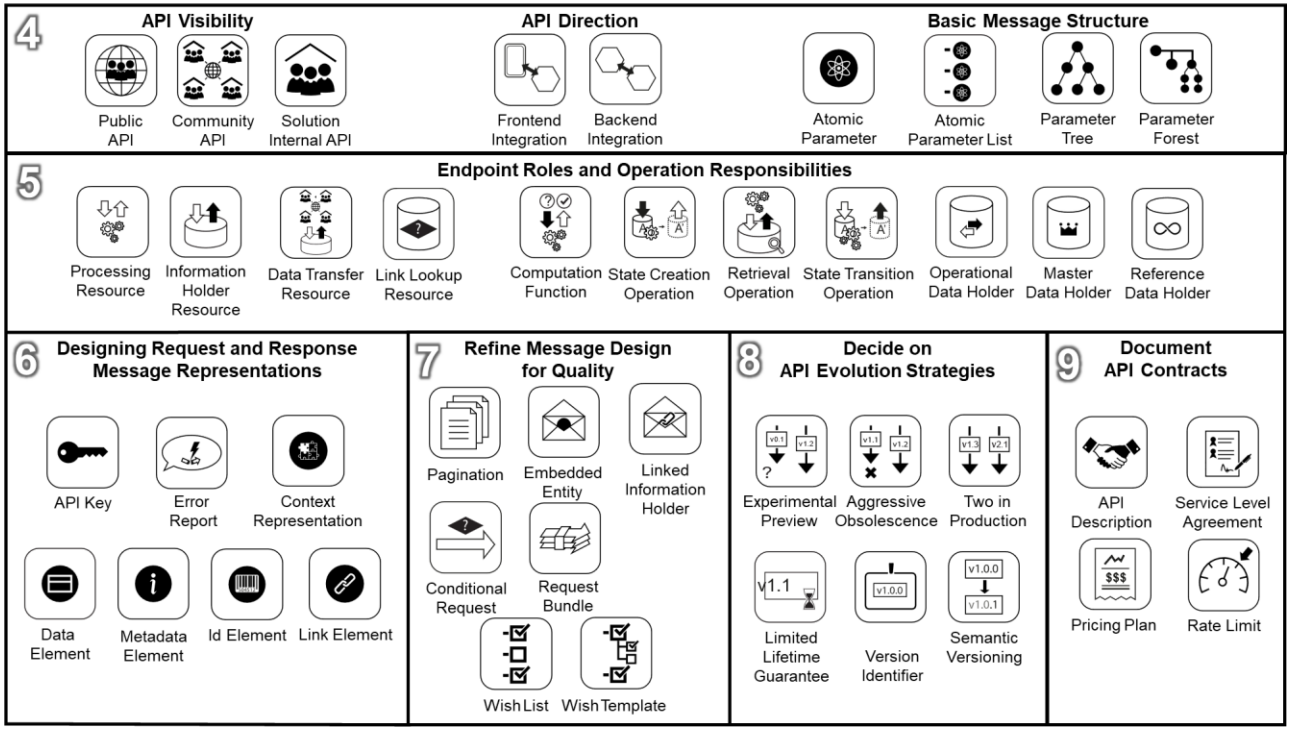
+

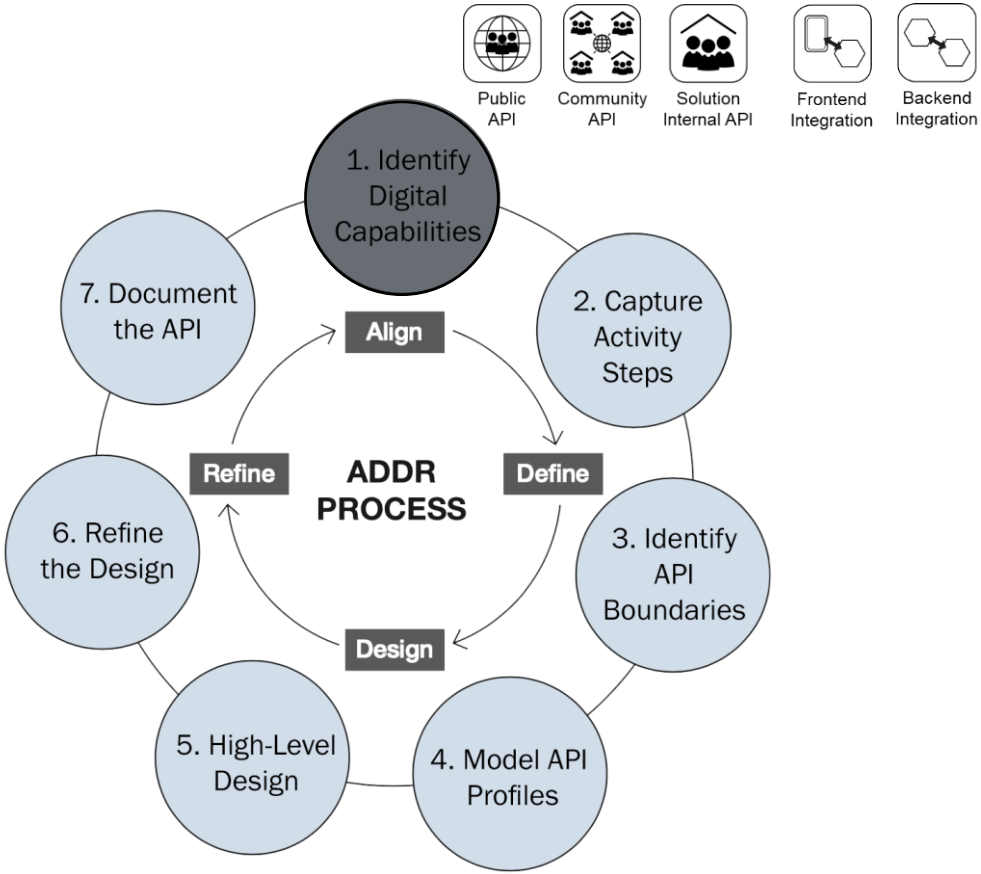


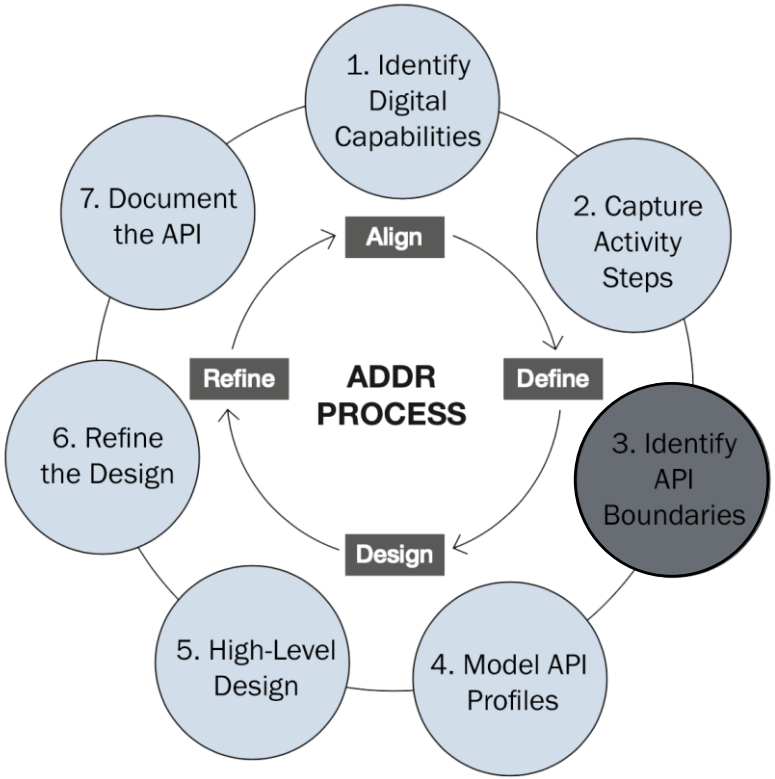
This item: Principles of Web API Design: Delivering Value with APIs and Microservices (Addiso…

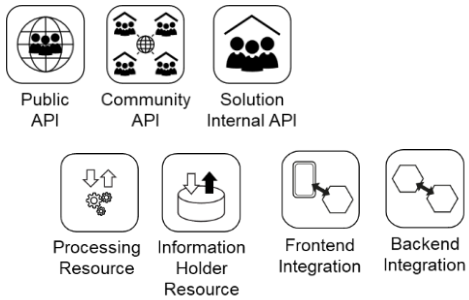
Patterns for API Design: Simplifying Integration with Loosely Coupled Message…

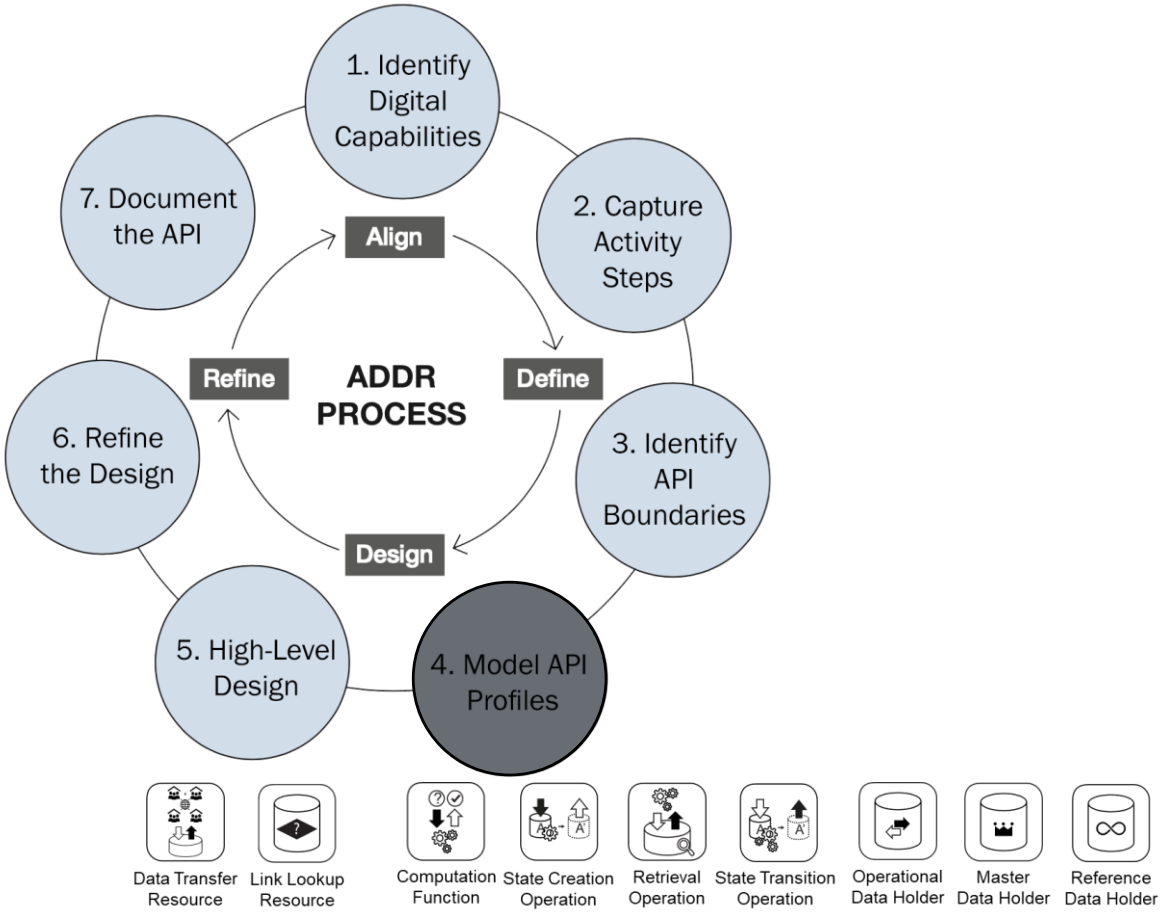
Design Patterns as a Checklist

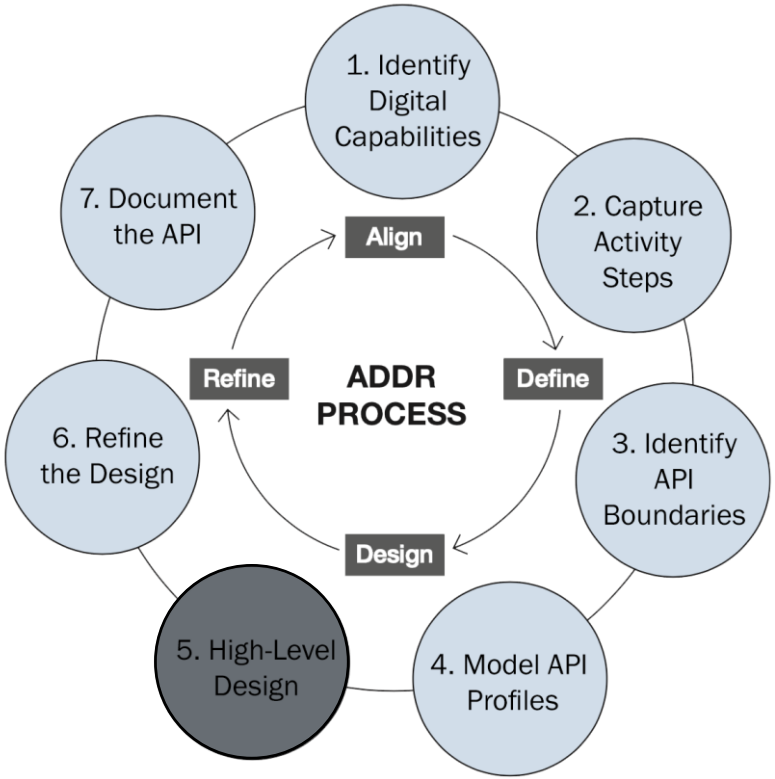


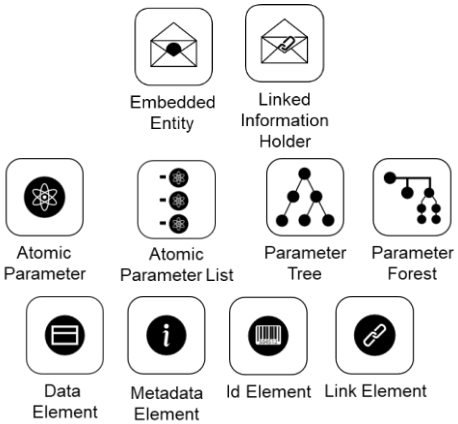


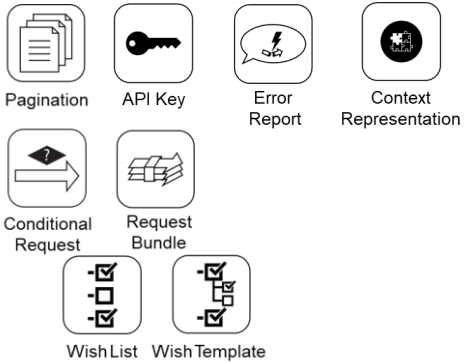


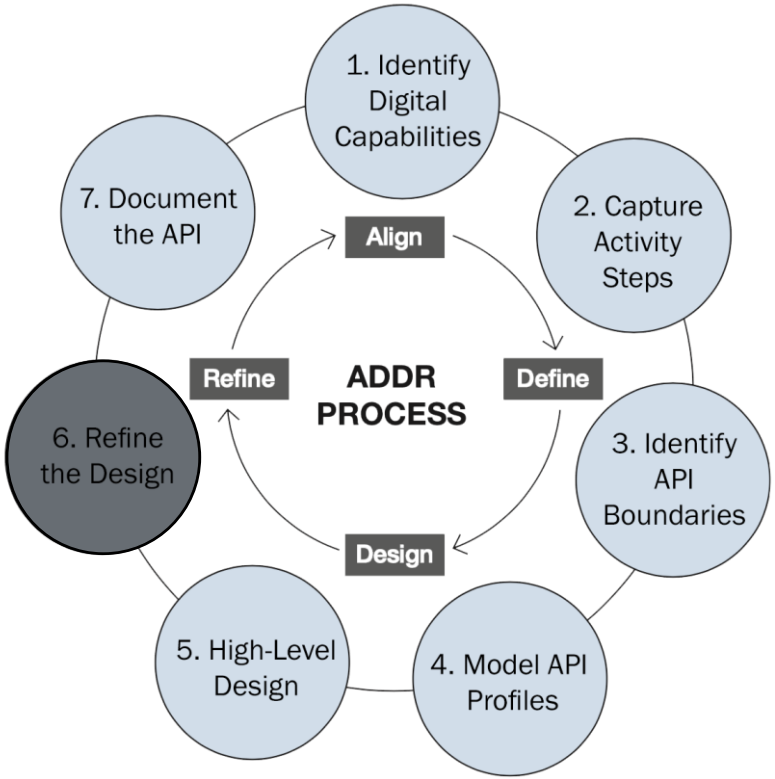


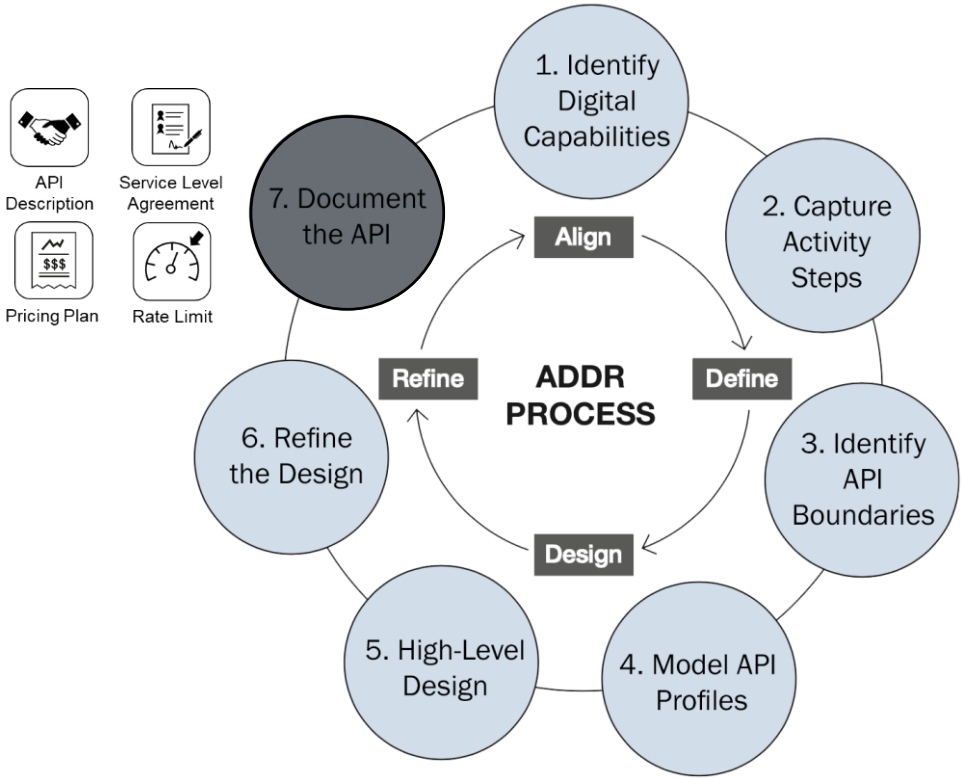

1. Identify Digital Capabilities
2. Capture Activity Steps
3. Identify API Boundaries
4. Model API Profiles
5. High-Level Design
6. Refine the Design
7. Document the API
Align
Define
Design
Refine
ADDR PROCESS
Data Transfer Resource
Link Lookup Resource
Computation Function
State Creation Operation
Retrieval Operation
State Transition Operation
Operational Data Holder
Master Data Holder
Reference Data Holder

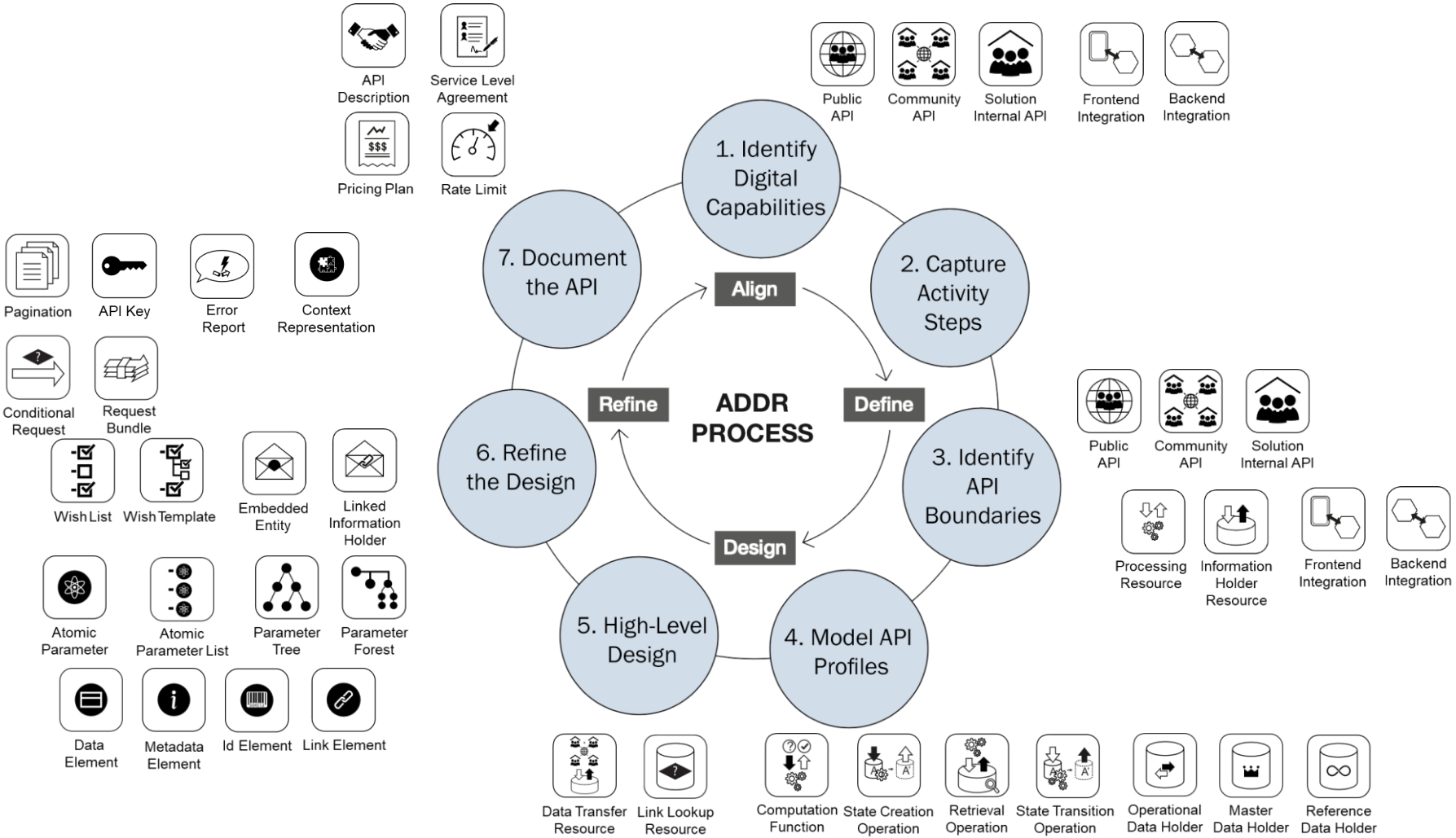




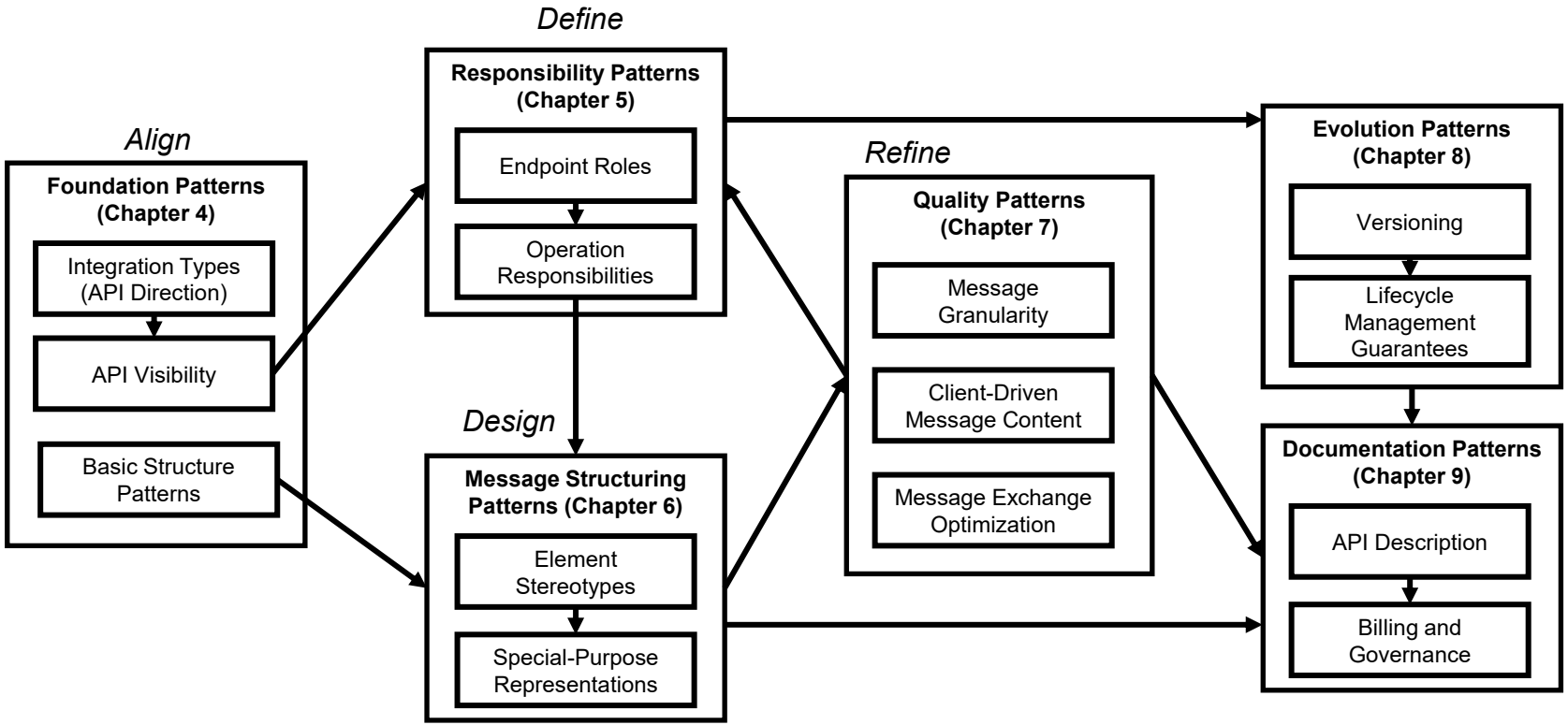







API Description
Service Level Agreement
Pricing Plan
Rate Limit
Public API
Community API
Solution Internal API
Frontend Integration
Backend Integration
Pagination
API Key
Error Report
Context Representation
Conditional Request
Request Bundle
Wish List
Wish Template
Embedded Entity
Linked Information Holder
Atomic Parameter
Atomic Parameter List
Parameter Tree
Parameter Forest
Data Element
Metadata Element
Id Element
Link Element
1. Identify Digital Capabilities
2. Capture Activity Steps
3. Identify API Boundaries
4. Model API Profiles
5. High-Level Design
6. Refine the Design
7. Document the API
Align
Define
Design
Refine
ADDR PROCESS
Public API
Community API
Solution Internal API
Processing Resource
Information Holder Resource
Frontend Integration
Backend Integration
Data Transfer Resource
Link Lookup Resource
Computation Function
State Creation Operation
Retrieval Operation
State Transition Operation
Operational Data Holder
Master Data Holder
Reference Data Holder

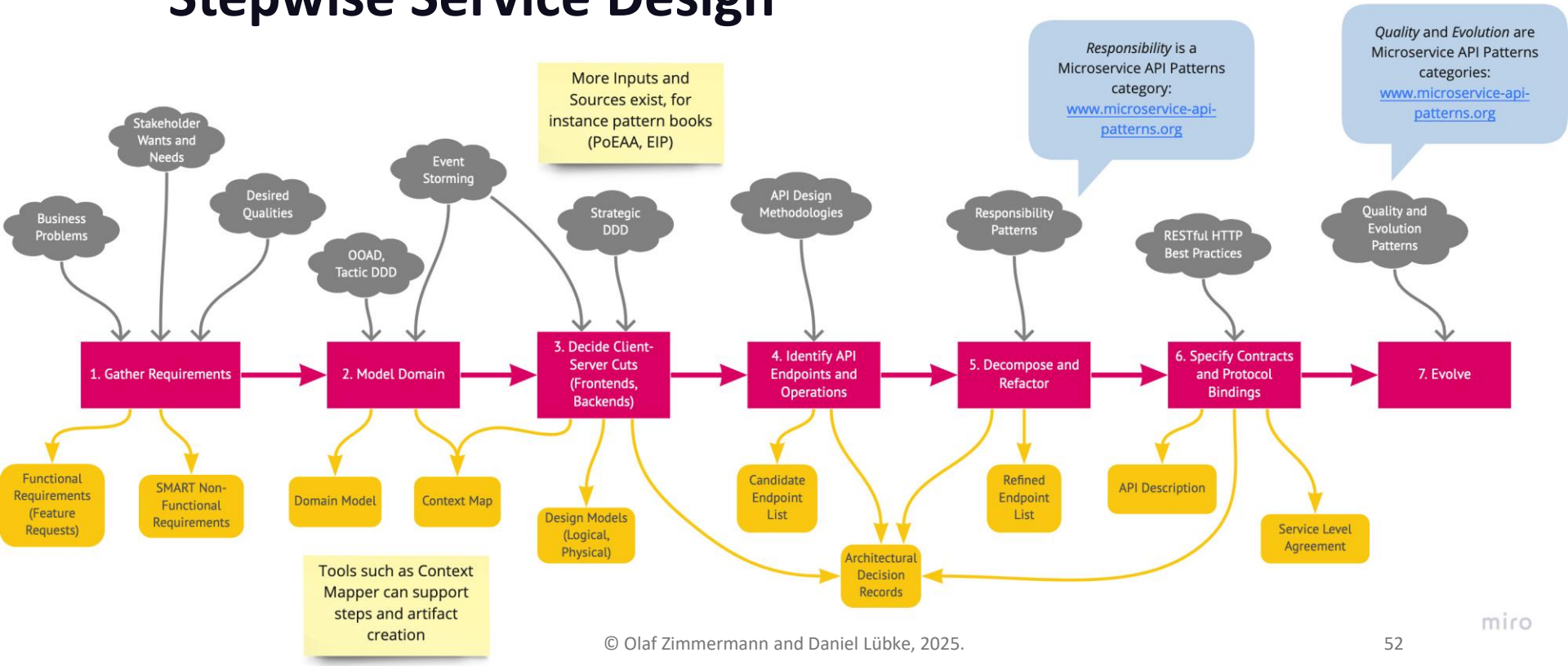
Patterns for API Design: Chapter/Category MAP



Outlook and Summary

https://socadk.github.io/design-practice-repository/activities/SDPR-StepwiseServiceDesign.html

Design Practice Repository (DPR) Practice: “Stepwise Service Design”



What to do next?

- Use API design patterns (ours, others) as checklist, as documentation tool
  - *See examples in this presentation*
- Make API design patterns part of your personal API design process
  - *Or community guidelines?*
- Donate known uses and discussion input:
  - *As Ton and team did: https://api-patterns.org/book/pattern-adoption-story-1*
- **Afternoon sessions:**
  - *Top 5 patterns deep dive (interest-driven, interactive)*
  - *Asynchronous APIs: questions, options, criteria (design space discussion)*

Professional Services & Research Project(s)

- Digital Solution Architecture: https://www.digital-solution-architecture.com or email Daniel
- Research project: "Towards Sustainable Software Architectures for Cyber-physical Systems of Systems" Systems"
  - *The IoT and CPSS have many APIs… requirements and decision modelling input very welcome! welcome!*
- JSS Dear Researchers column to narrow gap between practice (you!) and academia

Swiss National Science Foundation

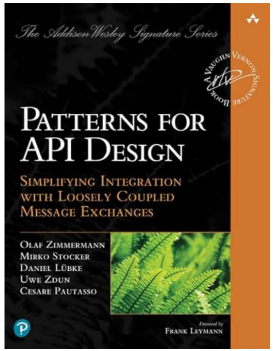
https://www.sciencedirect.com/special-issue/10DML17WPDQ

Summary

https://api-patterns.org/book/

- *API design is a wicked problem, many criteria and options, no single optimal solution*
- *API first: Business processes and domain models shape API purpose and structure*
- *API processes/guidelines and patterns complement each other nicely*
- *Patterns establish a common vocabulary, report proven solutions, serve as checklists*
- *“Patterns for API Design” available online, in articles, in book*

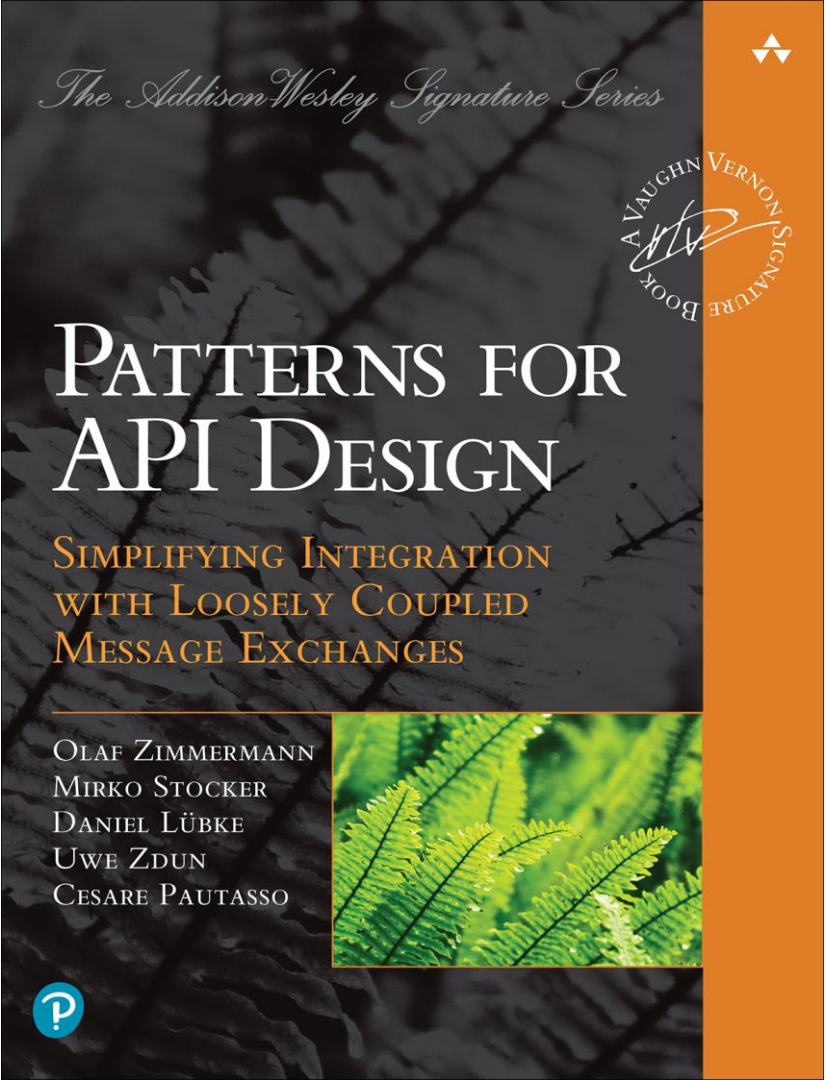
Slides



Thank you for your attention!

Are there any questions?

Slides



Order & Save 35%* on eBook at informit.com/api-patterns

- Use code **API-PATTERNS** during checkout
- Offer only good at informit.com
- eBook – DRM-Free PDF & EPUB

Print books available**
Please check your local or online store where you purchase technical related books.

*Discount code API-PATTERNS is only good at informit.com and cannot be used on the already discounted book + eBook bundle or combined with any other offer. Discount offer is subject to change.

**If your order print books from InformIT, your order is subject to import duties and taxes, which are levied once the package reaches the destination country.