Chapter 2

Patterns and Antipatterns

2.1 Amorphous vs. Tidy Endpoint

An endpoint is considered Tidy Endpoint if it has (1) lower-case resource naming, (2) no underscores, (3) no trailing slashes, and (4) no file extensions. In contrast, Amorphous Endpoint antipattern occurs when endpoints have either capital letters (except for camel cases), underscores, trailing slashes, or file extensions that make them difficult to use and read [27].

2.1.1 Example

- The endpoint `/Available-Data-Feeds/` is an Amorphous Endpoint as it contains trailing slashes and uppercase letters.
- In contrast, `/available-data-feeds/{dataSourceId}` is a Tidy Endpoint as it does not contain any uppercase letters, underscores, trailing slashes, or file extensions.

2.1.2 Detection Heuristics

```
1:  AMORPHOUS-ENDPOINT(Request-Endpoint):
2:    if uppercase or underscore or trailing-slash or file-extensions in Request-Endpoint
3:      return 'Amorphous Endpoint' antipattern
4:    end if
5:    return 'Tidy Endpoint' pattern
```

Figure 2.1: Heuristic for detecting *Amorphous Endpoint* antipatterns and *Tidy Endpoint* patterns.

2.2 Contextless vs. Contextualized Resource Names

Nodes in the endpoint should belong to the same semantic context, i.e., the endpoints should be contextual. Contextless Resource Names antipattern occurs when nodes in the endpoint do not belong to the same semantic context [27]. In the case, nodes in the endpoint belong to the same semantic context, it is known as Contextualized Resource Names pattern.

2.2.1 Example

- The endpoint `/newspapers/earth/players/{id}` is a Contextless Resource Names antipattern because nodes are not semantically related.
- In contrast, the endpoint `/football/club/players/{id}` is a Contextualized Resource Names pattern because all nodes are semantically related.

```
1:  CONTEXTLESS-RESOURCE(Request-Endpoint, API-Documentation)
2:    TopicsModel ← Extract-Topics(API-Documentation)
3:    EndpointNodes ← Extract-Endpoint-Nodes(Request-Endpoint)
4:    Similarity-Value ← Calculate-Similarity(EndpointNodes, TopicsModel)
5:    if Similarity-Value < Threshold:
6:      return 'Contextless Resource Names' antipattern
7:    end if
8:    return 'Contextual Resource Names' pattern
```

Figure 2.2: Heuristic for detecting *Contextless Resource Names* antipatterns and *Contextualized Resource Names* patterns.

2.2.2 Detection Heuristic

2.3 CRUDy vs. Verbless Endpoint

The Verbless Endpoint does not use CRUDy terms such as create, read, update, delete, or their synonyms. In contrast, the use of such terms as resource names is CRUDy Endpoint [27]. Moreover, resources should be identified using nouns, instead of verbs.

2.3.1 Example

- The endpoint `update/players/{id}` is a CRUDy Endpoint antipattern as it has CRUDy term "update".
- In contrast, endpoint `/players/{id}` is a Verbless Endpoint pattern as the endpoint does not contain any CRUDy terms or their synonyms.

```
1:  CRUDy-ENDPOINT(Request-Endpoint):
2:    CRUDyWords ← {"create", "read", "update", "delete", "get", ...}
3:    if CRUDyWords in Request-Endpoint
4:      return 'CRUDy Endpoint' antipattern
5:    end if
6:    return 'Verbless Endpoint' pattern
```

Figure 2.3: Heuristic for detecting *CRUDy Endpoint* antipattern and *Verbless Endpoint* pattern.

2.3.2 Detection Heuristic

2.4 Inconsistent vs Consistent Documentation

Inconsistent Documentation antipattern occurs when the HTTP method (i.e., the action) of an endpoint is in contradiction with its documentation. In contrast, for Consistent Documentation pattern, the HTTP method (the action) agrees with the documentation [29].

2.4.1 Example

- In Adobe Audience Manager API, the HTTP method (POST) of the endpoint `/datasources/bulk-delete` is in contradiction with the documentation 'Bulk delete multiple data sources', and thus is an Inconsistent Documentation antipattern. According to the API design guidelines, the POST method should be used to create some resources [22].

- In contrast, in Pipefy API the HTTP method (POST) of the endpoint /createCardRelation is consistent with its documentation 'Creates a card relation', and thus, is a Consistent Documentation pattern.

```
1:  INCONSISTENT-DOCUMENTATION(HTTP-Method, Request-Endpoint, Documentation)
2:    Documentation ← Remove-Stop-Words(Documentation)
3:    Action ← Extract-Intended-Action(Documentation)
4:    if ((HTTP-Method = 'POST' and Action in SYNONYMS (Delete or Update or Get)) or
5:      (HTTP-Method = 'DELETE' and Action in SYNONYMS (Create or Update or Get)) or
6:      (HTTP-Method = 'PUT' and Action in SYNONYMS (Create or Delete or Get)) or
7:      (HTTP-Method = 'GET' and Action in SYNONYMS (Delete or Update or Create)))
8:      return "Inconsistent Documentation" antipattern
9:    end if
10:   return "Consistent Documentation" pattern
```

Figure 2.4: Heuristic for detecting *Inconsistent Documentation* antipattern and *Consistent Documentation* pattern.

2.4.2 Detection Heuristic

2.5 Non-descriptive vs. Descriptive Endpoint

In API design, endpoints must be as user-friendly as possible. An endpoint needs to be easy to understand and as precise as possible. When an endpoint design has encoded nodes (e.g., basic resource names not used), it becomes a Non-descriptive Endpoint antipattern and gets harder to comprehend. A Self-descriptive Endpoint, on the other hand, has resource identifiers that are short and to the point [28].

2.5.1 Example

- The endpoint `/auth/token/oauth1` is a Non-descriptive Endpoint as it is not descriptive enough and hard to understand the purpose of the endpoint.
- In contrast, the endpoint `/account/set-profile-photo` is a Self-descriptive Endpoint as the endpoint is descriptive and easy to understand.

```
1:  NON-DESCRIPTIVE-ENDPOINT(Request-Endpoint)
2:    Nodes ← Split-Compound-Words(Request-Endpoint)
3:    Nodes ← Expand-Acronyms(Nodes)
4:    isValidWord = False
5:    for each word in Nodes:
6:      if PerformWordLookup(word):
7:        isValidWord = isValidWord or True
8:      else
9:        isValidWord = isValidWord or False
10:     if isValidWord:
11:       return 'Descriptive Endpoint' antipattern
12:   end if
13:   return 'Non-Descriptive Endpoint' pattern
```

Figure 2.5: Heuristic for detecting *Non-Descriptive Endpoint* antipattern and *Descriptive Endpoint* pattern.

2.5.2 Detection Heuristic

2.6 Non-hierarchical vs Hierarchical Nodes

The nodes in endpoints in the Hierarchical Nodes pattern are in a hierarchical relationship. In contrast, a Non-hierarchical Nodes antipattern occurs when at least one node in an endpoint is not hierarchically related to its neighbor nodes [27].

2.6.1 Example

- The endpoint `/professors/university/faculty/` is a Non-hierarchical Nodes antipattern since 'professors', 'faculty', and 'university' are not in a hierarchical relationship.
- In contrast, `/university/faculty/professors/` is a Hierarchical Nodes pattern since 'university', 'faculty', and 'professors' are in a hierarchical relationship.

```
1:  NON-HIERARCHICAL-NODES(Request-Endpoint)
2:    Nodes ← Extract-Endpoint-Nodes(Request-Endpoint)
3:    for each index in Length(Nodes):
4:      if Is-Hierarchical-Relation(Nodes_index, Nodes_index+1) = false or
5:      Is-Specialisation-Relation(Nodes_index, Nodes_index+1):
6:        return 'Non-Hierarchical Nodes' antipattern
7:      end if
8:    end for
9:    return 'Hierarchical Nodes' pattern
```

Figure 2.6: Heuristic for detecting *Non-Hierarchical Nodes* antipattern and *Hierarchical Nodes* pattern.

2.6.2 Detection Heuristic

2.7 Non-pertinent vs Pertinent Documentation

Non-pertinent Documentation antipattern occurs when an endpoint and its corresponding documentation are not semantically related. In contrast, a properly documented endpoint uses semantically related terms to clearly describe its purpose, denoted as Pertinent Documentation pattern [26].

2.7.1 Example

- In PokéAPI, the endpoint `/v2/berry-firmness/{names}/` is not semantically related with its documentation 'The name of this resource is listed in different languages', and thus is a Non-pertinent Documentation antipattern.
- In contrast, another endpoint-documentation pairs from PokéAPI: `/v2/berry-firmness/{berries}/` – 'A list of the berries with this firmness.' shows a higher semantic relationship, and thus is considered a Pertinent Documentation pattern.

```
1:  NON-PERTINENT-DOCUMENTATION(Request-Endpoint, Documentation)
2:    Documentation ← Remove-Stop-Words(Documentation)
3:    Tokens ← Lemmatise-Tokenise(Documentation)
4:    TopicsModel ← Extract-Topics(API-Documentation)
5:    EndpointNodes ← Extract-Endpoint-Nodes(Request-Endpoint)
6:    Similarity-Value ← Calculate-Similarity-Score(EndpointNodes, TopicsModel)
7:    if Similarity-Value < threshold:
8:      return 'Non-Pertinent Documentation' antipattern
9:    end if
10:    return 'Pertinent Documentation' pattern
```

Figure 2.7: Heuristic for detecting *Non-Pertinent Documentation* antipattern and *Pertinent Documentation* pattern.

2.7.2 Detection Heuristic

2.8 Non-standard vs Standard Endpoint

A Standard Endpoint design does not contain (1) non-standard characters such as é, å, ø, etc, (2) blank spaces, (3) unknown characters, and (4) double hyphens. In contrast, The use of characters such as é, å, ø, etc., the presence of blank spaces, the usage of double hyphens, and the presence of unknown characters in endpoint are the four main indicators of Non-standard Endpoint design [29].

2.8.1 Example

- The endpoint `/data--feeds/billingreport` from IBM Cloud Pak System API is an example of Non-standard Endpoint Design as endpoint contains a double hyphen.
- In contrast, the endpoint `/data-feeds/billing-report` represents Standard Endpoint design.

```
1:  NON-STANDARD-ENDPOINT(Request-Endpoint)
2:    if (Non-English-Characters or
      Space or Double-Hyphens or Unknown-Characters) in Request-Endpoint:
3:      return 'Non-standard Endpoint' antipattern
4:    end if
5:    return 'Standard Endpoint' pattern
```

Figure 2.8: Heuristic for detecting *Non-Standard Endpoint* antipattern and *Standard Endpoint* pattern.

2.8.2 Detection Heuristic

2.9 Pluralized vs Singularized Nodes

Endpoints should use singular/plural nouns consistently when naming resources in the API. The last node of the request endpoint should be singular when clients send PUT/DELETE requests, thus, a Singularized Nodes pattern occurs. In contrast, the last node should be plural for POST requests. Consequently, when singular names are used for POST requests or plural names are used for PUT/DELETE requests, the Pluralized Nodes antipattern occurs [27].

2.9.1 Example

- In Adobe Audience Manager API, the POST method is used with the `/data-feeds/usageendpoint` whose last node is a singular noun, and thus it is Pluralized Nodes antipattern.
- In contrast, if PUT or DELETE were used with the same endpoint, it would have been Singularized Nodes pattern, as singular last nodes are supposed to be used with PUT or DELETE methods.

```
1:  PLURALIZED-NODES(Request-Endpoint, HTTP-Method)
2:    Last-Node ← Get-Last-Node(Request-Endpoint)
3:    Second-Last-Node ← Get-Second-Last-Node(Request-Endpoint)
4:    if (HTTP-Method = 'PUT' or 'DELETE' and Is-Plural(Last-Node) = true) or
5:      (HTTP-Method = 'POST' or 'DELETE' and Is-Plural(Second-Last-Node) = false):
6:      return 'Pluralized Nodes' antipattern
7:    end if
8:    return 'Singularized Nodes' antipattern
```

Figure 2.9: Heuristic for detecting *Pluralized Nodes* and *Singularized Nodes* antipatterns.

2.9.2 Detection Heuristic

2.10 Unversioned vs Versioned Endpoint

Versioned Endpoint makes maintenance simpler for client developers as well as API providers. The format or type of response data may change, a resource may be removed, a new endpoint may be added, response parameters may change, and major or minor API versions are needed to track all the changes. An endpoint exhibits the Unversioned Endpoint antipattern if not versioned, while endpoints using version information is known as Versioned Endpoint pattern [29].

2.10.1 Example

- For example, the endpoint `/file\_requests/count` from Dropbox is an Unversioned Endpoint antipattern because the endpoint does not contain any version information.
- In contrast, another endpoint `/v1/me/library/playlists/{id}` from Apple Music is a Versioned Endpoint pattern because the endpoint contains the version number.

2.10.2 Detection Heuristic

```
1:  UNVERSIONED-ENDPOINT(Request-Endpoint):
2:    if Is-Version-Info-Available(Request-Endpoint) = true:
3:      return 'Versioned Endpoint' pattern
4:    end if
5:    return 'Unversioned Endpoint' antipattern
```

Figure 2.10: Heuristic for detecting *Versioned Endpoint* and *Unversioned Endpoint* antipatterns.

2.11 Parameters Tunneling vs Adherence

Web API clients often required to provide additional information (parameters) via path parameters and query parameters with the endpoint. Query parameters are normally used to sort, filter, and paginate request data, while path parameters are used to identify or retrieve a specific resource. Query parameters can be used with the GET method for sorting, filtering, and paginating resources. Path parameters, on the other hand, can be used with any HTTP method to identify or retrieve a specific resource. Thus, the use of query parameters with methods other than GET results in Parameter Tunneling antipattern. Conversely, the consistent use of the correct HTTP method with query and path parameters is considered as Parameter Adherence pattern.

2.11.1 Example

- An endpoint `/api/books?category=fiction` with the PUT method illustrates Parameter Tunneling antipattern.
- Another endpoint `/api/books/{id}` with PUT exemplifies Parameter Adherence pattern.

2.11.2 Detection Heuristic

```
1:  PARAMETERS-TUNNELING(Request-Endpoint, HTTP-Method)
2:    HasQueryParam ← Extract-Query-Parameter(Request-Endpoint)
3:    if HasQueryParam = true and HTTP-Method ≠ GET:
4:      return 'Parameter Tunneling' antipattern
5:    else:
6:      return 'Parameter Adherence' pattern
```

Figure 2.11: Heuristic for detecting *Parameter Tunneling* antipattern and *Parameter Adherence* pattern.

2.12 Inconsistent vs Consistent Resource Archetype Names

Resource archetypes serve as the fundamental building blocks of API endpoints. Four commonly used resource archetypes are Document, Collection, Store, and Controller. Singular resource names should be used for Documents, while plural names should be used for Collections and Stores. Verbs, combined with the POST method, should be used for Controllers [22]. Inconsistencies in resource archetype naming, i.e., using singular nouns for Collections or Stores, plural nouns for Documents, or Controllers without the POST method, result in the Inconsistent Resource Archetype Names antipattern. In contrast, proper naming of archetype results in Consistent Resource Archetype Names pattern [7].

2.12.1 Example

- The endpoint `/recipe/desserts/pie` (following a singular/plural/singular structure) demonstrates the Consistent Resource Archetype Names pattern.
- The endpoint `/recipes/desserts/pie` (with a plural/plural structure) exemplifies the Inconsistent Resource Archetype Names antipattern.

2.12.2 Detection Heuristic

```
1:  INCONSISTENT RESOURCE ARCHETYPE NAMES(Request-Endpoint)
2:    Nodes ← Split(Request-Endpoint)
3:    Category ← Categorize-Nodes(Last-Node(Nodes))
4:    if Category = "Document"and Is-Singular(Category) = true:
5:      return 'Consistent Resource Archetype' pattern
6:    else if Category = "Collection"or "Store"and Is-Plural(Category) = true:
7:      return 'Consistent Resource Archetype' pattern
8:    else if Category = "Controller"and Is-Verb-Phrase(Category) = true:
9:      return 'Consistent Resource Archetype' pattern
10:   end if
11:   return 'Inconsistent Resource Archetype' antipattern
```

Figure 2.12: Heuristic for detecting *Inconsistent Resource Archetype* antipattern and *Consistent Resource Archetype* pattern.

2.13 Identifier Ambiguity vs Identifier Annotation

Path parameters or resource identifiers should be enclosed in curly braces, angle brackets, or followed by a colon sign. Using such symbols in endpoint design is referred as Identifier Annotation pattern, which improves endpoint readability and understandability. In contrast, the absence of curly braces, angle brackets, or colon to represent resource identifiers would result in an Identifier Ambiguity antipattern.

2.13.1 Example

- The identifier in endpoint `/v2/lists/:id/members/appInstallation/{id}/rules/` is enclosed in curly braces and follows the Identifier Annotation pattern.
- In contrast, the identifier in the endpoint `/rules/ruleKey/idFromLegacyId` is enclosed in any braces making it hard to identify and resulting in an Identifier Ambiguity antipattern.

2.13.2 Detection Heuristic

```
1:  IDENTIFIER_AMBIGUITY(Request-Endpoint)
2:    Identifiers ← Extract-Identifiers(Request-Endpoint)
3:    if Identifiers enclosed in "{}" or "<>" or starts with ":":
4:      return 'Identifier Annotation' pattern
5:    end if
6:    return 'Identifier Ambiguity' antipattern
```

Figure 2.13: Heuristic for detecting *Identifier Ambiguity* antipattern and *Identifier Annotation* pattern.

2.14 Flat vs Structured Endpoint

Forward slash (/) must be used to separate nodes of an endpoint and indicate a hierarchical relationship. A Structured Endpoint pattern occurs when forward slashes are used to break down complex resource names to improve the readability and understandability of the endpoint. In contrast, a Flat Endpoint antipattern occurs when complex or large resource names are not broken down with forward slashes [22].

2.14.1 Example

- The endpoint `/requests/{request_id}/receipt/requests/{request_id}/map` is an example of a Structured Endpoint pattern because it does not have any complex and large resource names.
- In contrast, the endpoint `/requestFirmwareUpdateFromInStoreReader` and `/requestItemDisplayFromInStoreReader` exhibits Flat Endpoint antipatterns as it contains complex and large resource names.

2.14.2 Detection Heuristic

```
1:  FLAT-ENDPOINT(Request-Endpoint)
2:    Nodes ← Split(Request-Endpoint)  on  "/"
3:    for each Node  in  Nodes:
4:      SplittedNodes ← Split(Node)
5:      if SplittedNodes > 2:
6:        return 'Flat Endpoint' antipattern
7:      end if
8:    end for
9:    return 'Structured Endpoint' pattern
```

Figure 2.14: Heuristic for detecting *Flat Endpoint* antipattern and *Structured Endpoint* pattern.