Conformance Assessment of Architectural Design Decisions on the Mapping of Domain Model Elements to APIs and API Endpoints

1st Apitchaka Singjai
Research Group Software Architecture
University of Vienna
Vienna, Austria
apitchaka.singjai@univie.ac.at

2nd Uwe Zdun
Research Group Software Architecture
University of Vienna
Vienna, Austria
uwe.zdun@univie.ac.at

***Abstract*—Microservice APIs are often identified and designed based on Domain-Driven Design (DDD). To help in the continuous analysis of mappings of domain model elements to APIs and API endpoints, we aim to automate the assessment of conformance to Architectural Design Decision (ADD) options. The ADDs, their decision options, and relevant decision drivers studied in this paper originate from an empirical study on the mapping of domain model elements to APIs and API endpoints in practice. We propose automated detectors to detect the decision options of the ADDs taken in a given microservice API model, and an assessment scoring scheme based on the empirical knowledge codified in the ADDs. We evaluate our work, by first manually creating a ground truth for 14 cases in a multi-case study, and then comparing the results of our automated detectors to the ground truth for each of the 14 cases. In the cases we were able to identify 86% of the decision points correctly, and a statistical analysis of our data shows only a negligible effect size for differences to the ground truth.**

***Index Terms*—Microservice Architecture, API Design, DDD, Conformance Assessment, Architectural Design Decision**

I. Introduction

Microservices are independently deployable, scalable, and changeable services, each having a single responsibility [1]. They typically communicate via Application Programming Interfaces (APIs) in a loosely coupled fashion. Such remote APIs can be realized using various technologies, including RESTful HTTP, queue-based messaging, SOAP/HTTP, or remote procedure call technologies such as gRPC. A critical aspect in designing a microservice architecture is API design [2]. Many microservices and API abstractions are identified using Domain-Driven Design (DDD) [3]. It is an approach that places the (business) domain at the center of software designing and architecting.

This paper focuses on automating the assessment of Architectural Design Decisions (ADDs) [4] applied in the mapping of domain model elements to APIs and API endpoints. In particular, we focus on establishing a conformance relation between the DDD and API models. In general, the conformance relation is defined as the consistency between models [5]. To guarantee the correctness of this consistency relation, an assessment is needed. Conformance assessment is challenging because it concerns the relation between a software system's architecture and its intended architecture [6]. The target audience of this work are software/API developers and architects who are interested in the relations of DDD and APIs, as well as software engineering researchers studying these concepts.

ADDs, in general, focus on the architecturally significant design decisions of a system, in the sense that a single decision change could significantly affect its entire architecture [7]. In the field of microservice APIs and their relations to domain models, many central design problems revolve around mapping domain model elements to APIs and their endpoints. We have adopted the API endpoints definition from our earlier work on patterns for deriving APIs and their endpoints from domain models [8]. Changes in an API or its endpoints can significantly affect the architecture of the API clients and of the microservices in the backend serving the API [9]. We selected three ADDs on the interrelation of microservice API design and DDD from an empirical study on ADDs in the field of DDD/API mapping [10]. These ADDs explain (1) how to map the domain model and its elements to the API; (2) how to map domain model elements to API endpoints; and (3) how to formally describe or document the API. Each of these ADDs is described with multiple possible decision options and decision drivers along with positive and negative impacts practitioners identified for each of the drivers on the various options.

Such an empirically grounded ADD model helps in guiding the manual derivation of API designs from DDD models based on current practices employed by practitioners. However, assessing a given microservice API model on which ADD options have been chosen and how well they support the decision drivers is still a laborious and error-prone manual task. It is problematic when continuous assessment is required, e.g. in a continuous integration/delivery (CI/CD) pipeline. Just consider today's large-scale microservice deployments that are deployed with high frequency (e.g. at least daily), such as those of Uber [11], Google [12], or Netflix [13]. To manually assess the services, whether the mapping of the APIs and their endpoints to domain models are still in place and correct, would be an extremely laborious and error-prone manual task.

To address this problem, in this paper we aim to automate the assessment of conformance to ADD options. We set out to answer the following research questions:

- **RQ1** How can we automatically assess conformance to ADD options used in the mapping of domain model elements to APIs and API endpoints?
- **RQ2** How well do such automated measures for assessing ADD conformance in the mapping of domain model elements to APIs and API endpoints perform?

To address the research questions, we first collected and modeled 14 cases in a multi-case study. We then derived a scoring scheme based on the decision drivers in the empirically-grounded ADDs. This scheme is used to manually assess each case, to create a ground truth for evaluating our automated approach later on. To support the automated assessment, we developed detectors that aim to identify each of the decision points in our scoring scheme. With this, we were able to automatically assess the cases in our case study evaluation. We generated simple count metrics to compare our detector results to our ground truth. Finally, we performed a statistical analysis of the results.

In this paper, we first present the three ADDs covered as background in Section II. After explaining the research method in Section III, in Section IV we present the case study preparation, while Section V presents our detectors approach for automating ADD assessment. Next, Section VI presents the multi-case study which evaluates our research, as well as a statistical analysis of the results. Section VII discusses threats to validity. Finally, in Section VIII we draw conclusions.

II. Background: ADDs on Mapping of Domain Model Elements to APIs and API Endpoints

This section explains the three ADDs from an empirical study on the mapping of domain model elements to APIs and API endpoints [10], for which we aim to provide automated conformance assessment support in this work.

1) Model Mapping Decision (MMD): This ADD focuses on how to map the domain model and its elements to an API. There are five alternative design options practitioners commonly apply in this ADD. Firstly, one can *Expose the Whole Domain Model in 1:1 Relation as API*, but this is seen rather negatively as it increases coupling and adversely impacts many API maintainability aspects. An often better working solution is *Expose Domain Model Subset as API* as it can reduce some of those negative impacts. Many practitioners use DDD's *Bounded Context* pattern [3] for defining boundaries of the APIs with the options *Expose Selected Bounded Contexts as APIs* being seen more positive than the option *Expose Each Bounded Context as an API*. Still, even in selected Bounded Contexts, domain model elements that do not belong to the API might get exposed, thus the options *Introduce and Expose Interface Bounded Context as an API* and *Expose a Shared Kernel between Client and Server as an API* are seen as the most favourable options, as in them a special *Bounded Context* or a *Shared Kernel* [3] is designed with the goal to specify an API interface.

2) API Endpoints Decision (API-ED): This ADD addresses the problem which domain model elements should be offered as endpoints in an API. Please note that it includes endpoints in various kinds of technologies such as RESTful HTTP, messaging, SOAP, gRPC, and so on. The options for this ADD are related to the basic patterns in tactical DDD [3]. The most positive options of this decision are those that expose *Aggregate* roots or *Service* as API endpoints. The option to expose *Entities* as API endpoints is often discussed, but also has many negative impacts e.g. on exposing domain model details in the API, data consistency, and chatty APIs. Broader concepts such as *Bounded Contexts* and *Processes* are also options, but issues regarding API complexity, data consistency, and coupling are reported for these options.

3) API Documentation Decision (API-DD): This ADD is about how to document the API. There are two main options that can be combined in different ways: *API Contract* and *API Description* [9], [14]. The *API Contract* specifies structural information about the API in a formal language, e.g., based on Open API or RAML for RESTful APIs, WSDL for SOAP APIs, and so on. The *API Description* is a detailed specification of the API and contains more information than the contract, such as invocation sequences, pre-conditions, post-conditions, quality management policies, and so on. API Descriptions can be classified into *informal* or *structured* descriptions. Combinations of API contracts and structured descriptions available for all APIs are seen as the most positive option. More incomplete or less formal combinations are gradually seen as more negative.

III. Research Methods

Figure 1 illustrates the research methods used. Firstly, we performed a *Case Studies Inspection*, in which we have prepared 14 cases in total for the later evaluation of our approach. In this, we followed the guidelines for conducting and reporting case study research in software engineering by Runeson et al. [15]. We used major search engines (e.g., Google, Bing, DuckDuckGo) and topic portals (e.g., InfoQ, DZone) to find relevant cases. To avoid personal bias in the research, we used as search words those keywords that provided decision categories (i.e. the most general categorization) in the used gray literature [10], in particular: "Application Programming Interface" or "API", "Domain Driven Design" or "DDD", and "Microservices" (all in singular and plural form). We had to check that the found cases contained information on their domain modeling, which substantially limited the possible cases we could consider. Further, we checked for each found case whether it was realized by authors with a substantial background in industrial practice. Finally, we selected systems from many different domains and covering a broad range of our ADDs' decision options and their combinations (for details see Table I explained below). We assume that our evaluation systems are, or reflect practical examples of microservice architectures. As many of them are open source systems with the purpose of demonstrating practices or technologies, they are at most of medium size and modest complexity, though.

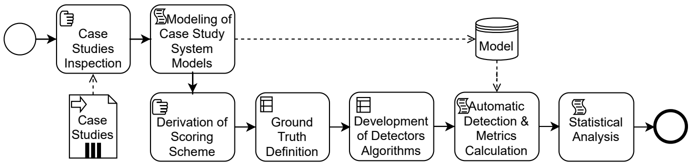
Case Studies Inspection
Modeling of Case Study System Models
Model
Case Studies
Derivation of Scoring Scheme
Ground Truth Definition
Development of Detectors Algorithms
Automatic Detection & Metrics Calculation
Statistical Analysis

Fig. 1: Research Method Overview

Next, we performed *Modeling of Case Study System Models*: We defined models and meta-models to precisely model the case study systems as UML models. We used the CodeableModels tool[1], a Python implementation for this purpose. Based on these models, we realized automated code generators to generate graphical visualizations of all meta-models and models in PlantUML[2].

After that, we conducted the *Derivation of a Scoring Scheme* in which we have selected the three ADDs on the interrelation of microservice API design and DDD (summarized in Section II). We derived the scoring for the practitioner views on the practices as objectively as possible, consisting of the following possible scores on an ordinal scale: [++: very positive, + : positive, o : neutral, -: negative, - -: very negative]. The ordinary scale helped us turn qualitative judgments in the practitioner texts into numerical assessments. We also defined the *Ground Truth Assessment* based on this scoring scheme.

As explained before, it is the goal of our study to provide support to automate this manual assessment as far as possible. In the next step, *Development of Detectors Algorithms*, we developed our DDD/API mapping assessment detectors approach as our main contribution. The major focus of our detectors approach is on automating the assessment provided by the scoring scheme. For this, we developed detector algorithms to detect each relevant criterion deciding on scores in the scheme.

In the next research step, *Automatic Detection and Metrics Calculation*, we mapped the detection algorithms to the assessments by calculating simple count metrics. Finally, we performed an *Empirical Assessment* of our approach using a multi-case study using the cases prepared and inspected earlier. We used the ground truth truth assessment and compared them to our automatically derived scores from the detectors.

Besides analyzing and discussing the results on a case-by-case basis for each decision option, we performed a *statistical analysis*. We use R's *shapiro.test()* function to perform Shapiro-Wilk normality tests [16]. As the data sets are non-normally distributed, we used the Wilcoxon signed rank test with Pratt method [17], provided by the *wilcoxsign_test()* function from R's *coin* package, to test for a significant difference between the ground truth and the automated detector results. Finally, Cliff's δ [18] is used for effect size estimation via *cliff.delta()* function from R's *effsize* package. Moreover, this way it was easier to provide an audit trail of the research, and thus enable the repeatability of the study through public access to original data [3].

IV. Case Preparation and Inspection

This section explains the case studies' preparation and inspection. Table I summarizes the 14 case studies we have manually modeled based on descriptions in the referenced sources. We have modeled DDD domain models, microservice API models, and the mapping of DDD domain models to API models for each case. The information in the table includes the number of domain elements, number of API elements, a brief description, the solutions for each of the ADDs, and a URL for the original sources of the cases.

A. Derivation of Scoring Scheme

In this section, we explain the scoring scheme to assess the ADDs in Section II. For each of the decisions introduced in Section II, we present precise decision points in conditional statements and boolean logic operators based on the possible decision options of the ADDs.

For example, consider the MMD decision, as discussed in [10]. According to the empirical evidence in the practitioner sources, *Expose the Whole Domain Model in 1:1 Relation as API* is seen as working only for small examples, as it leads to negative impacts on coupling and maintainability decision drivers such as *Brittle Interfaces*, *API Complexity*, and *Avoiding Exposing Domain Model Details in API*. A usually better working solution is *Expose Domain Model Subset as API* which partly improves on the negative decision driver impacts. Both solutions have benefits like little required *Design and Implementation Effort*. For complex domains, it can be advisable to consider the *Bounded Contexts* as well. Then there are the options to *Expose Each Bounded Context as an API* or *Expose Selected Bounded Contexts as APIs*. In most large domains, the latter solution is seen as being better suited to avoid *Brittle Interfaces*, *Exposing of Domain Model Details in the API*, and *API Complexity*, and improve *API Usability* and *API Modifiability*. Both solutions offer positive impact on *Design and Implementation Effort*, but require more effort than the first two solutions. The downside of those solutions is that *Clients Need to Manage Crossing Model Boundaries*, i.e., the boundaries between the *Bounded Contexts*. One suggested solution to this problem is to *Introduce and Expose Interface Bounded Context as an API* (or alternatively realized as a *Shared Kernel* [3]). That is, a new special *Bounded Context* or *Shared Kernel* that represents the API interface is exposed. These solutions are neutral or positive on all so far mentioned decision drivers, except the *Design and Implementation Effort* where they lead to additional effort compared to all other so far mentioned solutions. All those solutions are better than any *combinations of API and domain model where no fully traceable mapping between them can be discerned.*

These considerations have led to a literal derivation of the scoring scheme presented in the next section of the Decision MMD. For the other two decisions, their scoring scheme based on the empirical study results from [10].

[1] https://github.com/uzdun/CodeableModels

[2] https://plantuml.com/en/

[3] (https://doi.org/10.5281/zenodo.5176174).

TABLE I: Overview of the case study models

ID	Elements	Description and Summary of ADD Inspection
MD1	domain: 5 api: 14	Purchase Order System; applies DDD concepts and CQRS to decompose components. ADD Options used: **MMD** (Expose Each Bounded Context as an API), **API-ED** (Aggregate Roots as API Resources), **API-DD** (Each API has an API Contract). https://dzone.com/articles/bounded-contexts-with-axon
MD2	domain: 10 api: 11	Publication Management System; applying DDD concepts to API design, detailed API specifications and code generation. ADD Options used: **MMD** (Expose Selected Bounded Context as an API), **API-ED** (Aggregate Roots as API Resources), **API-DD** (Each API has an API Contract and a structured API Description). https://ozimmer.ch/practices/2020/06/10/ICWEKeynoteAndDemo.html
MD3	domain: 8 api: 3	Bank Account System; focuses on event-based architecture, CQRS and event sourcing patterns. ADD Options used: **MMD** (Expose Domain Model Subset as API), **API-ED** (some API resources it is unclear on which domain model elements they are based, and some are Domain Services as API Resources), **API-DD** (Each API has an API Contract). https://github.com/cer/event-sourcing-examples
MD4	domain: 19 api: 18	Online Shop System; uses DDD and pattern-based modelling of APIs. ADD Options used: **MMD** (Expose Selected Bounded Contexts as API), **API-ED** (Some Aggregate Roots and Entities as API Resources), **API-DD** (Each API has API Contract and structured API Desc.). https://github.com/socadk/design-practice-repository/blob/master/tutorials
MD5	domain: 12 api: 52	Cinema Microservice System; models multiple frontends and REST-based APIs. ADD Options used: **MMD** (Expose Each Bounded Context as an API), **API-ED** (Domain Services as API Resources), **API-DD** (Most APIs have an API Contract). https://github.com/Crizstian/cinema-microservice
MD6	domain: 4 api: 142	E-Shop on Containers System; follows both simple CRUD and DDD/CQRS patterns; uses pub/sub for event-based interaction. ADD Options used: **MMD** (Expose Each Bounded Context as an API), **API-ED** (For some API resources it is unclear on which domain model elements they are based), **API-DD** (Each API has an API Contract). https://github.com/dotnet-architecture/eShopOnContainers
MD7	domain: 8 api: 15	Customers and Orders System; implements transaction with the SAGA pattern and implements queries using CQRS. ADD Options used: **MMD** (Domain Model Subset Exposed to API), **API-ED** (Some Aggregate Roots as API Resources), **API-DD** (Each API has an API Contract). https://github.com/eventuate-tram/eventuate-tram-examples-customers-and-orders
MD8	domain: 3 api: 3	E-commerce Application; has a Web UI directly accessing microservices and an API gateway for service-based API. ADD Options used: **MMD** (Expose Each Bounded Context as API), **API-ED** (N/A), **API-DD** (No API description or contract). https://microservices.io/patterns/microservices.html
MD9	domain: 11 api: 34	Kanban Board System; a multi-user collaborative application using event-sourcing and pub/sub. ADD Options used: **MMD** (Expose Each Bounded Context as an API), **API-ED** (Bounded Contexts and Domain Services as API Resources), **API-DD** (Some APIs have an API Contract). https://github.com/eventuate-examples/es-kanban-board
MD10	domain: 8 api: 72	Disease Statistics App; a public API providing a wide range of virus information. ADD Options used: **MMD** (Map Domain Model Fully to the API), **API-ED** (Entities as API Resources), **API-DD** (Each API has an API Contract and a structured API Description). https://github.com/disease-sh/API
MD11	domain: 63 api: 215	Pokemon App; a RESTful API for infos on Pokemon game series. ADD Options used: **MMD** (Domain Model Exposed 1:1 (except Bounded Contexts)), **API-ED** (Entities as API Resources), **API-DD** (Each API has an API Contract and a structured API Description). https://github.com/PokeAPI/pokeapi
MD12	domain: 6 api: 56	Realworld Example App; provides API specs that support technology stack diversity. ADD Options used: **MMD** (Expose Selected Bounded Context as API), **API-ED** (Bounded Context as API Resources), **API-DD** (Each API has API Contract and structured API Description). https://github.com/gothinkster/realworld
MD13	domain: 12 api: 6	Taxi hailing Application; uses REST APIs, multiple frontends, and databases per services. ADD Options used: **MMD** (Expose Each Bounded Context as an API), **API-ED** (N/A), **API-DD** (no formal or informal description or API contract). https://www.nginx.com/blog/introduction-to-microservices
MD14	domain: 170 api: 92	Lakeside Mutual System; demonstrates DDD in microservices of an insurance product. ADD Options used: **MMD** (Expose Selected Bounded Context as an API), **API-ED** (Bounded Contexts as API Resources), **API-DD** (Each API has an API Contract). https://github.com/Microservice-API-Patterns/LakesideMutual

1) MMD: How to Map a Domain Model and its Elements to an API?:

- IF for some APIs no traceable mapping to domain model elements can be discerned, THEN assessment = (- -).
- ELSE IF the options *Introduce and Expose Interface Bounded Context as an API* and/or *Expose a Shared Kernel between Client and Server as an API* are used AND no other kinds of Bounded Contexts are exposed to the API, THEN assessment = (++).
- ELSE IF *Expose the Whole Domain Model in 1:1 Relation as API* is used, THEN assessment = (-).
- ELSE IF for all APIs *Expose Domain Model Subset as API* is used (where some of those might be realized using *Introduce and Expose Interface Bounded Context as an API* OR *Expose a Shared Kernel between Client and Server as an API*), THEN assessment = (+)
- ELSE IF for all APIs *Expose Selected Bounded Context as an API*, THEN assessment = (+)
- ELSE assessment = (o) (e.g. *Expose Each Bounded Context as an API*).

2) API-ED: Which Domain Model Elements Should be Offered as Endpoints in an API?:

- IF for some API endpoints it cannot be discerned on which domain model elements they are based, THEN assessment = (- -).
- ELSE IF for all API endpoints *Entities as API Endpoints* OR other domain model elements than [Services, Aggregates, Domain Processes, Bounded Contexts] as API Endpoints are used, THEN assessment = (-).
- ELSE IF for all API endpoints either *Domain Services as API Endpoints* OR *Aggregate Roots as API Endpoints* OR *Domain OR Business Processes as API Endpoints* are used, THEN assessment = (++).
- ELSE IF for some API endpoints, *Entities as API Endpoints* OR other domain model elements than (Services, Aggregates, Domain Processes, Bounded Contexts) as API Endpoints are used, THEN assessment = (o).
- ELSE: assessment = (+) (e.g., if Bounded Contexts are used as Endpoints).

3) API-DD: How to formally describe the API?:

- IF no API has a formal OR informal *API Description* OR *API contract*, THEN assessment = (- -).
- ELSE IF each API has an *API Contract* AND a structured *API Description*, THEN assessment = (++).
- ELSE IF each API has an *API Contract* AND an informal *API Description*, THEN assessment = (+).

- ELSE IF each API has an *API Contract* OR a structured *API Description*, THEN assessment = (o).
- ELSE: assessment = (-) (e.g. only an informal *API Description* or some APIs are not documented).

V. Detectors for ADD Conformance Assessment

In this section, we describe details about our detectors approach for automatically assessing conformance to decision options in the mapping of domain model elements to APIs and API endpoints. We propose a modular detector approach, in which one or more detectors are responsible for detecting each of the decision points in the scoring scheme from Section IV-A. Table II provides an overview of all detectors we have defined for the three decisions. The *assessment* column indicates how each detector helps in making a decision on the scoring scheme: *s* means that the detector must be *successful* for leading to the respective assessment, *f* means that the detector must fail to contribute to the assessment; *u* means that the detectors is *unused* in the respective assessment.

All detectors are implemented in Python and mainly operate by traversing models. All models are implemented using CodeableModels a Python tool for the precise specification of meta-models, models, and model instances in code. We create the association to identify which API elements are derived from domain elements. Based on the meta-models and decision models, we manually created model instances for every case study systems, and realized automated PlantUML code generators to generate graphical visualizations of all model instances.

VI. Multi-Case Study Results

Here, we discuss the results of our multi-case study. We analyze them on a case-by-case basis for each decision option. In doing so, we simply count the correctly identified results and discuss interesting findings. Next, we statistically analyze our dataset and show that there is no significant difference between the ground truth data and the detector results and the effect size between the two variables is negligible.

A. Discussion of the Results

Table III shows the results of the assessment. We show the expected results (E: the results of the ground truth assessment) and actual results (A: automated detector result). For the ground truth assessment, we have assessed each of our case studies manually based on our scoring scheme from Section IV-A. Some assessments are not applicable (“n/a”), if specific aspects are not explained in the case study source, meaning that we could not judge the decision based on this case’s model.

The column *Reason* shows which specific detectors from Table II have failed or been a success, and thus led to the automated assessment. Comparing the results to our ground truth in Table III, we can summarize the matching score ratios of MMD as 8/14 (57%), API-ED as 14/14 (100%), and API-DD as 14/14 (100%), respectively. In total, 86% of the decision points in our multi-case study have been correctly identified by the automated detectors. The places where ground truth and detector results diverge illustrate well those cases where human judgment is needed. This indicates, especially in the ADD MMD where we observe divergence, the detectors should not be applied “blindly” as automated tests but rather as indicators of possible conformance violations. When we apply the detectors, we also get the violation set as part of the detector result. These violations can help to spot the aspects humans need to inspect more closely.

Firstly, it can be observed that all decision points of API-ED and API-DD are matching. For API-ED to be 100% correctly assessed, we needed to extend our scoring scheme slightly, to also cover the case that no API endpoints are modeled (yielding “n/a” instead of the default case). This is because this decision is about the concrete mapping of DDD model elements to API endpoints, which is rather easy to specify precisely in automated detectors. In contrast to the other ADDs, the API-DD decision is pretty straightforward to model and realize with detectors, and thus no ambiguous cases have been observed.

Let us discuss a few notable cases where MMD diverges in detail. One case where the ground truth and the detector assessment can diverge is the option *Expose Domain Model Subset as API* of the MMD ADD. For instance, consider MD3. Here, a domain model subset is exposed as API elements, but this is only modeled at the level of API endpoints, not for the API. If the model would be corrected, to expose also the API, the detector would yield the correct result. So, here the detector has spotted an omission in the model, which could be fixed. A correctly identified case is the MMD ADD for MD7 where again a domain model subset is exposed as an API (here services). This could easily be wrongly modeled as in MD3, e.g. by exposing the services to API endpoints. But here this is not the case because aggregates are used as well, and they are exposed to API endpoint. In summary, for the option *Expose Domain Model Subset as API* of MMD there might be modeling issues in those cases where MMD and API-ED overlap. Here, modelers might feel the relations to APIs and API endpoints are redundant and thus omit them. The detectors deliver precise results on where the problem occurred and can thus help to fix the problem.

Another critical case is the detection of *no traceable mapping to domain model elements can be discerned* for MMD. This option can in a number of cases take precedence in detectors over other options. Interestingly, again this concerns cases where MMD and API-ED overlap: For example, in MD11, the assessment result should be “-” because the domain model is exposed 1:1 to the API. As some Bounded Contexts in the model are pure aggregation elements not exposed to the API (but all their members are exposed), formally the model is not fully exposed, leading to the difference in human and automatic judgment. A similar issue occurs in MD12 where selected Bounded Contexts are exposed. But as those are mapped to API endpoints for the API-ED decision, they are not mapped to the API again, leading to the difference in the assessment results. Finally, in MD14 the same issue as

TABLE II: Overview for Detectors for ADD Conformance Assessment

ADD	ID	Detectors	Description	Assessment				
				++	+	o	-	- -
MMD	d1	detect_is_each_domain_model_element_exposed_to_api (*input: Domain_Model*)	To detect if each domain model element is exposed to the API.	u	u	u	s	u
	d2	detect_is_each_bounded_context_exposed_to_api (*input: Domain_Model*)	To detect if each Bounded Context is exposed to the API.	f	f	s	u	u
	d3	detect_are_selected_bounded_ontext_exposed_to_api (*input: DomainModel*)	To detect selected Bounded Contexts are exposed to the API.	f	s	u	u	u
	d4	detect_all_apis_are_exposed_by_subset_of_domain_model_elements (*input: Domain_Model*)	To detect all APIs are exposed by a subset of domain model elements (Services or Processes).	u	s	u	u	u
	d5	detect_is_each_api_element_exposed_by_domain_model_element (*input: API_Elements*)	To detect all APIs have a traceable mapping to domain model elements.	u	u	u	u	f
	d6	detect_interface_bounded_context (*input: Domain_Model*)	To detect if a dedicated Interface Bounded Context is used.	s	s	u	u	u
	d7	detect_shared_kernel (*input: Domain_Model*)	To detect if a Shared Kernel is used as interface between Bounded Contexts.	s	s	u	u	u
API-ED	d8	detect_is_each_api_endpoint_exposed_to_domain_model_element (*input: API_Elements*)	To detect if all API endpoints have a traceable mapping to domain model elements.	u	u	u	u	f
	d9	detect_all_api_endpoints_exposed_by_an_entity (*input: API_Elements*)	To detect all API endpoints are exposed by Entities which are not Aggregate roots.	u	u	u	s	u
	d10	detect_api_endpoints_exposed_by_a_bounded_context (*input: API_Elements*)	To detect API endpoints are exposed by Bounded Contexts.	u	s	u	f	u
	d11	detect_all_endpoints_exposed_by_aggregates_services_process (*input: API_Elements*)	To detect all API endpoints are exposed by Aggregates roots, Services, or Processes.	s	u	u	f	u
	d12	detect_some_endpoints_exposed_by_entities (*input: API_Elements*)	To detect some API endpoints are exposed by Entities or domain elements other than (Services, Aggregates, Processes, Bounded Contexts).	u	u	s	u	u
API-DD	d13	detect_each_api_has_structured_description_and_contract (*input: API_Elements*))	To detect if each API has a structured API description and an API contract.	s	u	u	u	u
	d14	detect_each_api_has_informal_description_and_contract (*input: API_Elements*)	To detect if each API has an informal API description and an API contract.	u	s	u	u	u
	d15	detect_each_api_has_structured_description_or_contract (*input: API_Elements*)	To detect each API has either a structured API description or an API contract.	u	u	s	u	u
	d16	detect_api_has_description_or_contract (*input: API_Elements*)	To detect an API has either an (informal or structured) API description or an API contract.	u	u	u	u	f

TABLE III: Assessment Results (E: Expected Results, A: Actual Results)

ID	MMD		Reason	API-ED		Reason	API-DD		Reason
	E	A		E	A		E	A	
MD1	o	o	*Successful Detectors*: d2, d3, d5 *Failed Detectors*: d1, d4, d6, d7	++	++	*Successful Detectors*: d8, d11, d12 *Failed Detectors*: d9, d10, d13	o	o	*Successful Detectors*: d16, d17 *Failed Detectors*: d14, d15
MD2	+	+	*Successful Detectors*: d3, d5 *Failed Detectors*: d1, d2, d4, d6, d7	++	++	*Successful Detectors*: d8, d11, d12 *Failed Detectors*: d9, d10, d13	++	++	*Successful Detectors*: d14, d16, d17 *Failed Detectors*: d15
MD3	+	- -	*Successful Detectors*: d4 *Failed Detectors*: d1, d2, d3, d5, d6, d7	- -	- -	*Successful Detectors*: d12 *Failed Detectors*: d8, d9, d10, d11, d13	o	o	*Successful Detectors*: d16, d17 *Failed Detectors*: d14, d15
MD4	+	+	*Successful Detectors*: d3, d5 *Failed Detectors*: d1, d2, d4, d6, d7	o	o	*Successful Detectors*: d8, d12, d13 *Failed Detectors*: d9, d10, d11	++	++	*Successful Detectors*: d14, d16, d17 *Failed Detectors*: d15
MD5	o	o	*Successful Detectors*: d2, d3, d5 *Failed Detectors*: d1, d4, d6, d7	++	++	*Successful Detectors*: d8, d11, d12 *Failed Detectors*: d9, d10, d13	-	-	*Successful Detectors*: d17 *Failed Detectors*: d14, d15, d16
MD6	o	- -	*Successful Detectors*: d2, d3, d4 *Failed Detectors*: d1, d5, d6, d7	- -	- -	*Successful Detectors*: d10 *Failed Detectors*: d8, d9, d11, d12, d13	o	o	*Successful Detectors*: d16, d17 *Failed Detectors*: d14, d15
MD7	+	+	*Successful Detectors*: d2, d3, d4, d5 *Failed Detectors*: d1, d6, d7	++	++	*Successful Detectors*: d8, d11, d12 *Failed Detectors*: d9, d10, d13	o	o	*Successful Detectors*: d16, d17 *Failed Detectors*: d14, d15
MD8	o	o	*Successful Detectors*: d2, d3, d5 *Failed Detectors*: d1, d4, d6, d7	n/a	n/a	*Successful Detectors*: d11 *Failed Detectors*: d8, d9, d10, d12, d13	- -	- -	*Successful Detectors*: none *Failed Detectors*: d14, d15, d16, d17
MD9	o	o	*Successful Detectors*: d2, d3, d5 *Failed Detectors*: d1, d4, d6, d7	+	+	*Successful Detectors*: d8, d10, d12 *Failed Detectors*: d9, d11, d13	-	-	*Successful Detectors*: d17 *Failed Detectors*: d14, d15, d16
MD10	-	- -	*Successful Detectors*: d4 *Failed Detectors*: d1, d2, d3, d5, d6, d7	-	-	*Successful Detectors*: d8, d9, d13 *Failed Detectors*: d10, d11, d12	++	++	*Successful Detectors*: d14, d16, d17 *Failed Detectors*: d15
MD11	-	- -	*Successful Detectors*: d4 *Failed Detectors*: d1, d2, d3, d5, d6, d7	-	-	*Successful Detectors*: d8, d9, d13 *Failed Detectors*: d10, d11, d12	++	++	*Successful Detectors*: d14, d16, d17 *Failed Detectors*: d15
MD12	+	- -	*Successful Detectors*: d2, d3, d4 *Failed Detectors*: d1, d5, d6, d7	+	+	*Successful Detectors*: d8, d10 *Failed Detectors*: d9, d11, d12, d13	++	++	*Successful Detectors*: d14, d16, d17 *Failed Detectors*: d15
MD13	o	o	*Successful Detectors*: d2, d3, d5 *Failed Detectors*: d1, d4, d6, d7	n/a	n/a	*Successful Detectors*: d11 *Failed Detectors*: d8, d9, d10, d12, d13	- -	- -	*Successful Detectors*: none *Failed Detectors*: d14, d15, d16, d17
MD14	+	- -	*Successful Detectors*: d2, d3, d4 *Failed Detectors*: d1, d5, d6, d7	+	+	*Successful Detectors*: d8, d10 *Failed Detectors*: d9, d11, d12, d13	o	o	*Successful Detectors*: d16, d17 *Failed Detectors*: d14, d15

in MD12 has happened. Also selected bounded contexts are exposed, but this is modeled only at for the API-ED decision, not for the API. All of these cases can be fixed by exposing the respective elements to the API as well, and the detectors pinpoint the problem location.

Both problems have their root cause in the fact that there is a slight overlap in the MMD and API-ED decisions, and redundant modeling of API and API endpoint mapping is needed to avoid issues. Instead of requiring the redundant modeling, we could modify the named detectors to fall back to the exposed API endpoints, if the API is not exposed.

Another issue occurred in MD6 and MD10 for decision MMD: Here “--” is reported as in each case one of many APIs (a “helper” API) is not linked to any domain model elements.

For a human it is obvious how the overall mapping logic is constructed, the machine identifies no traceable mapping for those. Here, completing the model would be needed for fixing the model, and the detectors clearly indicate where to fix this.

In summary, our case studies have shown that all ADD options could automatically be detected based on our scoring scheme (**RQ1**). Regarding **RQ2**, 86% of the decision points have been correctly identified, and the remaining cases are those where a modeling omission has occurred due to the conceptual overlaps of the MMD and API-ED decisions. Would we use the detector modification explained in this section, we could even reach 12/14 (86%) for MMD, and thus 95% in total. Please note that this modification would be only based on the experiences of this study, not on the empirical data from [10]. In cases, where case study models were missing certain redundant expose-relations, our detectors

B. Statistical Analysis

To confirm the results for **RQ2** and get a more precise estimate for the effect size, we statistically analyzed the results in R. In our data set, we had to deal with ordinal variables, a rather small sample size, and data that is not normally distributed. We first confirmed the non-normal distribution with a Shapiro-Wilk test [16] using R's *shapiro.test()* function. The data in Table IV shows that, as the p-values for both ground truth (0.0002982) and detector results (0.0004204) data are significant, we must reject the null hypothesis that the data is normally distributed for both variables. Thus, the t-test, which assumes the normal distribution, is not applicable. For our problem the Wilcoxon signed-rank test would be applicable, but as many data points are identical, we get many zero values, which are in Wilcoxon's method removed from the test, making the results non-exact for our data set. The Wilcoxon signed rank test with Pratt method can handle those zero values [17], which means that it is more appropriate for our data set. For Wilcoxon-Pratt Signed-Rank Test calculation, we used the *wilcoxsign_test()* function from R's *coin* package. The test's result (0.3173) is not significant (see Table IV), meaning that the null hypothesis cannot be rejected. Thus, we must assume the true μ is close to 0.

To confirm this result and assess the relevance of the result further, we computed the effect size. Here, Kitchenham et al. [19] suggest Cliff's δ [18] as a robust method for empirical software engineering. For this calculation, we used the *cliff.delta()* function from R's *effsize* package. As shown in Table IV, the delta estimate is 0.009375, which is to be interpreted as: the effect size is negligible [20].

TABLE IV: Statistical Analysis Results

Shapiro-Wilk Test for Ground Truth Data	
p-value	0.0002982
Shapiro-Wilk Test for Detector Results Data	
p-value	0.0004204
Wilcoxon-Pratt Signed-Rank Test	
p-value	0.3173
Cliff's Delta	
delta estimate	0.009375 (negligible)

VII. Threats to Validity

To avoid system composition and structure bias, we investigated multiple cases from a number of third-party authors. The search procedure of these systems might have led to unconscious exclusion of certain sources. We mitigated this by collecting a relatively high number of cases (14), and checking for each the background (e.g. all case authors are practitioners or have a practitioner background). The wide range of third-party systems increases the generalizability (external validity). Nonetheless, the threat to validity remains that most of our systems and the case authors have a business/enterprise system background (where DDD is usually applied). Thus our results might not be easily transferable to other contexts such as embedded systems. This threat is mitigated by the fact that the selected systems are from many different domains in this context and use various ADDs and ADD option combinations (see Table I). Further, some systems are built for demonstration purpose. Thus, it is possible that some aspects important in full-scale commercial systems are missing.

Regarding internal validity, we avoided researcher bias with faithful modeling from the evidence-based information. However the modeling process can be another source of an internal validity threat. We mitigated this threat by independently cross-checking our results numerous times in the author team. Lastly, the ground truth assessment depends on the decision maker's interpretation, and different practitioners might provide different assessment results. We mitigated this subjective point by comparing the ground truth assessment carefully to multiple data points from [10] in which a relatively high number of practitioner sources are studied (32). But some misinterpretation or bias cannot be entirely excluded here.

The construction of our scoring scheme is based on an interpretation and aggregation of practitioner texts in a qualitative, empirical study [10]. While precise decision drivers and impacts have been identified in the empirical study and followed by us in our scheme, an exact mapping e.g. to crisp numerical assessments would have introduced a significant threat of misinterpretations. In contrast, the used ordinal scale enabled us to turn the qualitative practitioner judgments into numerical information, significantly reducing this threat.

While ordinal scales are commonly used to reflect qualitative judgments [21]–[23], the threat to validity remains that some interpretations might not reflect the practitioner judgments accurately. Please note that this threat is mitigated by the fact that our study in first place provides a method for automation, not an empirical assessment of 14 system. If others have a different interpretation of some practitioner judgments in [10], it is easily possible to adapt the respective failing detector(s) accordingly. As we provide all artifacts (code, data set) as open access artifacts to enable reproducibility, such a calibration of our approach can be performed with low effort. Our approach even provides traceability helping to locate the failing detectors. The concrete system assessment results in Table III would then change, but the automation approach would not require any alterations.

VIII. Conclusion

In this paper, to answer **RQ1**, we introduced an automated assessment approach for conformance to ADDs on the mapping of domain model elements to APIs and API endpoints. We started with case study inspections and modeling. Next, we derived an empirically grounded scoring scheme based on the data from an empirical study on this subject. Then, we developed automated detectors to identify each of the decision points in our scoring scheme. We evaluated this approach in a multi-case study in which we compared a manually created ground truth to the detector results. Finally, we confirmed our results with a statistical analysis. To answer **RQ2**, in the case studies we were able to identify 86% of the decision points from our scoring scheme correctly. To confirm our results, we performed a statistical evaluation which showed no statistically significant difference between the two variables (ground truth and automated detector results), as well as a negligible effect size between the two variables. The remaining 14% were mainly cases where redundant modeling at the overlap of two of the ADDs was necessary, but was not correctly performed in the case study models. In two cases, for one of many APIs (a helper API) the mapping to the domain model was omitted. As shown in Section VI with a simple extension of our detectors to cover the redundant cases, we could even reach 95% correct identification. Our detectors delivered for each of the diverging cases regarding precise results where the problem was located and how to fix it. Thus, in summary, our detectors always yielded clear results for human architects to either assess the models correctly, or inspect and potentially fix modeling issues. Due to the proposed modeling framework and the traceability supported by our detector results, it is relatively easy to adapt or calibrate the approach to a given system setting and set of best practices, and create or improve detectors in a straightforward manner with low effort. Due to the high level of automation and accuracy achieved, our approach is applicable in frequent release and/or large-scale system setting, in which a frequent manual inspection would be clearly infeasible.

In our future work, we plan to work on other ADDs for API design and related design patterns. It is also possible to apply weighted scoring when we would focus on each API instead of each system in future work.

Acknowledgments: This work was supported by FWF (Austrian Science Fund) project API-ACE: I 4268. Our work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 952647 (AssureMOSS project).

References

[1] O. Zimmermann, "Microservices tenets," *Computer Science-Research and Development*, vol. 32, no. 3-4, pp. 301–310, Jul. 2017.

[2] O. Zimmermann, M. Stocker, D. Lübke, C. Pautasso, and U. Zdun, "Introduction to microservice api patterns (map)," *Post-proceedings of Microservices 2017/2019*, vol. 78, no. 4, pp. 1–17, 2020.

[3] E. Evans, *Domain-Driven Design: Tacking Complexity In the Heart of Software*. Reading, MA.: Addison-Wesley, 2003.

[4] J. S. van der Ven, A. G. Jansen, J. A. Nijhuis, and J. Bosch, "Design decisions: The bridge between rationale and architecture," in *Rationale management in software engineering*. Springer, 2006, pp. 329–348.

[5] D. Quartel and M. van Sinderen, "On interoperability and conformance assessment in service composition," in *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*, 2007, pp. 229–229.

[6] F. Deissenboeck, L. Heinemann, B. Hummel, and E. Juergens, "Flexible architecture conformance assessment with conqat," in *2010 ACM/IEEE 32nd International Conference on Software Engineering*, vol. 2, 2010, pp. 247–250.

[7] L. Lee and P. Kruchten, "A tool to visualize architectural design decisions," in *International Conference on the Quality of Software Architectures*. Springer, 2008, pp. 43–54.

[8] A. Singjai, U. Zdun, O. Zimmermann, and C. Pautasso, "Patterns on deriving apis and their endpoints from domain models," in *The 25th European Conference on Pattern Languages of Programs*, 2021.

[9] O. Zimmermann, M. Stocker, D. Lübke, C. Pautasso, and U. Zdun, "Microservice api patterns," https://microservice-api-patterns.org/, 2021.

[10] A. Singjai, U. Zdun, and O. Zimmermann, "Practitioner views on the interrelation of microservice apis and domain-driven design: A grey literature study based on grounded theory," in *18th IEEE International Conference on Software Architecture (ICSA 2021)*. Washington, DC, USA: IEEE, March 2021.

[11] M. Schwarz, "Uber engineering's micro deploy: Deploying daily with confidence," https://eng.uber.com/micro-deploy-code/, 2016.

[12] Google, "Devops tech: Architecture," https://cloud.google.com/architecture/devops/devops-tech-architecture, 2021.

[13] C. D. Nguyen, "A design analysis of cloud-based microservices architecture at netflix," https://medium.com/swlh/a-design-analysis-of-cloud-based-microservices-architecture-at-netflix-98836b2da45fe, 2020.

[14] D. Lübke, O. Zimmermann, C. Pautasso, U. Zdun, and M. Stocker, "Interface evolution patterns: Balancing compatibility and extensibility across service life cycles," in *Proceedings of the 24th European Conference on Pattern Languages of Programs*, ser. EuroPLop '19. ACM, 2019.

[15] P. Runeson, M. Host, A. Rainer, and B. Regnell, *Case study research in software engineering: Guidelines and examples*. John Wiley & Sons, 2012.

[16] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika*, vol. 52, no. 3/4, pp. 591–611, 1965.

[17] J. W. Pratt, "Remarks on zeros and ties in the wilcoxon signed rank procedures," *Journal of the American Statistical Association*, vol. 54, no. 287, pp. 655–667, 1959.

[18] N. Cliff, *Ordinal methods for behavioral data analysis*. Psychology Press, 2014.

[19] B. Kitchenham, L. Madeyski, D. Budgen, J. Keung, P. Brereton, S. Charters, S. Gibbs, and A. Pohthong, "Robust statistical methods for empirical software engineering," *Empirical Software Engineering*, vol. 22, no. 2, pp. 579–630, 2017.

[20] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data: Should we really be using t-test and cohen'sd for evaluating group differences on the nsse and other surveys," in *annual meeting of the Florida Association of Institutional Research*, vol. 13, 2006.

[21] O. Larichev and H. Moskovich, "Unstructured problems and development of prescriptive decision making methods," in *Advances in multicriteria analysis*. Springer, 1995, pp. 47–80.

[22] W. D. Perreault Jr and L. E. Leigh, "Reliability of nominal data based on qualitative judgments," *Journal of marketing research*, vol. 26, no. 2, pp. 135–148, 1989.

[23] G. Canfora, L. Cerulo, and L. Troiano, "Transforming quantities into qualities in assessment of software systems," in *Proceedings 27th Annual International Computer Software and Applications Conference. COMPAC 2003*. IEEE, 2003, pp. 312–319.