Measuring The Architecture Fit Of System Integration Designs

Konrad Nadobny[1,2,3], Andreas Schmietendorf[2,1]

[1]*Otto-von-Guericke-University Magdeburg, Faculty of Computer Science, Universitätsplatz 2, 39106 Magdeburg, Germany*

[2]*Berlin School of Economics and Law, Department 2 - Cooperative Studies Business and Technology, Alt-Friedrichsfelde 60, 10315 Berlin, Germany*

[3]*Bayer AG, Clinical Digital Innovation, Müllerstr. 178, 13353 Berlin, Germany*

Abstract

This paper introduces a measurement model for integration architectures and describes the initial validation based on an industrial case study.

The work is motivated by the rising challenge of transforming historically grown legacy system landscapes to reduce technical dept in large enterprises while mitigating limitations of competence sovereignty and sparse availability of integration architecture professionals.

To address this challenge, the authors describe an easy-to-use approach to classify integration designs and compare their strengths and weaknesses to give guidance on which integration concepts fit best to the needs of the respective use case. These insights can be used on a the one hand to identify potentially problematic integrations that need to be refactored and on the other hand to provide requirements-based guidance for designing new integrations. With this the authors aim to provide enterprise architects with a tool to quantify the architecture fit of the integrations in their information systems landscape while supporting them in prioritizing their efforts. To define the measurement model, the authors analyse established possibilities to classify system architectures and define a taxonomy for integration architectures based on topologies, paradigms and technologies. After defining the subject of measurements, the authors make use of established standards for software quality measuring and define *Confidentiality*, *Availability*, *Integrity*, *Flexibility*, *Throughput* and *Latency* as the metrics to measure integration designs against non-functional requirements.

To generate first empirical insights, the authors apply the measurement model to a real-world integration use case that is being refactored in context of a large industrial digital transformation project. By quantifying the gaps of the current integration design and comparing the results of the measurement model with the real-world observations, the authors gain first insights about the validity of their conceptualized approach. The results of the measurements imply, that the current integration using *File-Imports* has gaps in *Confidentiality*, *Integrity* and *Latency*. Switching to an integration using *Webhooks* or *Change-Data-Capture* would close those gaps and would conceptually result in a higher *Throughput* and a stronger *Integrity* and *Flexibility*. These outcomes are in line with the real-world observations, which implies that the conceptualized measurement model works and is fit-for-purpose. Even though the first results are very positive, the authors also identify weaknesses of the approach, so that further research is required for validate and refine the approach.

Keywords

integration architecture, software measurement, enterprise architecture, system integrations

IWSM-MENSURA'24, September 30 - October 4, 2024, Montreal, Canada

konrad.nadobny@ovgu.de (K. Nadobny); andreas.schmietendorf@hwr-berlin.de (A. Schmietendorf)
https://blog.hwr-berlin.de/schmietendorf/ (A. Schmietendorf)
0009–0005–1105–6768 (K. Nadobny); 0000-0003-2674-756X (A. Schmietendorf)

1. Introduction

1.1. Motivation and Background

”In an interconnected world, information and related processes, systems, and networks constitute critical business assets” [1, p. 13] To grow and maintain this asset is the role of Enterprise Architecture by maintaining a holistic view on an enterprise, including its the structures, processes and information systems in all its dimensions and complexity [2, p. 13-14]. This includes creating a software landscape that fosters end-to-end processes, including overcoming fragmented legacy architectures [3].

In context of system integration design, this means that IT architects are tasked to ensure the design, implementation, and operation of optimal system integrations. This is realised by defining fundamental concepts, principles and patterns [3, chapter 1.1] and applying them to create a well-functioning integrated application landscape. This is a challenge especially for large organizations with a historically grown integration architecture that consists of hundreds of systems. Technical dept‘ has been piling up and is hindering IT departments to invest into new solutions [4]. This challenge is also being illustrated by a study conducted by the German Institute for Economy (DIW). The DIW publishes a ’Digitization Index’ on a yearly basis. It shows a stagnation of successful digitization efforts since 2022 and identifies limited access to competent experts, or missing ‘sovereignty of competence‘ as one of the major reasons for this [5]. This observation is also backed by empirical findings published in the Enterprise Architecture Professional Journal (EAPJ) [6, page 12]. The EAPJ points out, that the skill base of practitioners varies a lot and needs to be improved. In this context, the need for an improved decision making on a basis of science and data, rather than opinion and conjecture is highlighted.

With this situation in mind, the authors are looking for possibilities to improve the ‘sovereignty of competence‘ in enterprise architecture by helping the sparsely available experts to prioritize their efforts and providing development teams with clear design guidelines for integrations. Software measurement provides an answer here: Instead of conducting a time-consuming in-depth analysis for each and every integration, measuring the architecture fit of different design patterns could be used to efficiently identify those individual legacy integrations that have a requirements gap and need to be looked at in detail. In addition, such a measurement model would provide guardrails to support less proficient personnel towards a fit-for-purpose integration design. The goal hereby is not to generate detailed insights about how exactly certain integration designs can be improved, but to provide a more high-level measurement of how well certain design patterns fit to the respective use case.

1.2. Scope and Scientific Contribution

This paper is looking at how qualitative software measurement can be applied to overcome the challenges in modern IT architecture in context of system integration. With this, the authors are addressing the timely need to cope with the growing complexity of enterprise systems.

In scope of this paper are the definition of an easy to use measurement model for integration design, including a classification of system integration designs, and a definition of metrics to measure those against given requirements. The initial calibration of the model is being conducted based on the authors empirical experience. A broader quantitative analysis of strengths and weaknesses of the respective integration designs is out of scope for this paper and subject to further research in this area.

The major scientific contribution lies in the construction and initial empirical validation of the measurement model. The challenge lies hereby in proving sufficient granularity of measurement to generate meaningful insights about strengths and weaknesses of individual integration designs while defining the taxonomy and metrics as generic as possible, so that the approach can be applied on a large scale in heterogeneous integration landscapes.

This paper introduces this novel approach to the enterprise architecture and software measurement community by providing a comprehensive definition of the measurement model and showing it's applicability using a real-world case study from one authors professional industrial background. Further empirical research is currently being conducted through applying the approach in a large-scale corporate setting, this however is beyond the scope of this paper.

1.3. Related work

This paper builds on methods and ideas discussed in the area of IT architecture and qualitative software measurement.

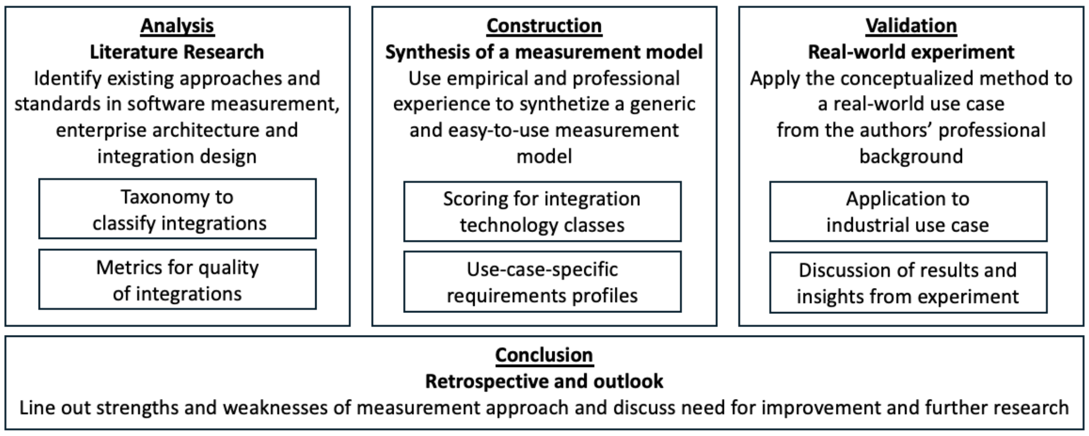
Foundational to the authors work is the COSMIC method for software sizing [7]. The idea of measuring system integration designs was briefly discussed in a short paper presented at the IWSM conference in 2019 by Hartenstein et al [8]. The authors contributed to this paper that elaborated on an approach for a Fast Cost Validation of Web-Based APIs supported by Functional Size Measurement with COSMIC and showed that integrations can be sized at the implementation level based on a technical design specification. This principle of qualitative software measurement is being applied here to a more abstract concept of architecture patterns. While the COSMIC Function Points method constitutes an elaborated approach towards a very detailed software sizing based on functional requirements, the authors acknowledge that the COSMIC method does not attempt to size non-functional requirements [7, page 12]. Even though the respective non-functional requirements are implicitly represented by the functional requirements after being addressed in the software design process, the authors aim to close this gap with their approach to measure integration architectures using non-functional requirements metrics.

With focusing on integration design, the authors work is embedded in the field of enterprise architecture. The authors refer to publications like Schekkermans work on comparing enterprise architecture frameworks [2] and are referring to empirical research, like the publications of Carr and Else on maturity of professional enterprise architecture and industry-relevant frameworks [6]. Based on this, the authors choose to follow the methods, approaches and techniques for architecture development described by 'The Open Group' [9]. The measurement approach itself can be embedded into the TOGAF Architecture Development Method [10].

The general measurement approach and definition of metrics is heavily based on established standards like ISO 25000 [11] and ISO 27000 [1]. In earlier work the authors also described an approach to weight qualitative requirements to rate solutions for API-Management [12]. This approach is now being transferred to weighting of non-functional requirements for integration designs.

Further related work of the authors elaborates on the challenges and integration-specific approaches towards transforming legacy system landscapes into modern connectivity-ecosystems. The central role of integration design in this transformation has been subject to papers published by the authors between 2017 and 2019 at the Enterprise Computing Conference (ECC) [13] and academic workshops on Evaluation of Service APIs (ESAPI) [14, 15]. While this previous work focused on the technical challenges, this paper now introduces a possible solution to solve the conceptual challenges when coping with large integration architectures and describes first empirical insights gained in context of industrial application.

1.4. Method and Structure



Analysis
Literature Research
Identify existing approaches and standards in software measurement, enterprise architecture and integration design
Taxonomy to classify integrations
Metrics for quality of integrations
Construction
Synthesis of a measurement model
Use empirical and professional experience to synthetize a generic and easy-to-use measurement model
Scoring for integration technology classes
Use-case-specific requirements profiles
Validation
Real-world experiment
Apply the conceptualized method to a real-world use case from the authors' professional background
Application to industrial use case
Discussion of results and insights from experiment
Conclusion
Retrospective and outlook
Line out strengths and weaknesses of measurement approach and discuss need for improvement and further research

Figure 1: Research approach and structure of this paper [own graphic]

The authors apply a qualitative empirical approach to analyse the situation, conceptualise a solution, and validate it against real-world observations. Figure 1 shows the structure of this paper that follows the authors' empirical research approach.

As a perquisite for this work, the authors are discussing the background and motivation to develop a measurement approach for integration architectures. This is driven by the authors professional experience and supported by scientific and commercial empirical research outcomes.

In Section 2.1 the authors are lining out established approaches from software engineering and IT architecture and use the outcomes of this literature-based analysis to compile a taxonomy to classify system integration architectures. In the following Section 2.2, the authors analyse existing standards for software engineering and software product quality evaluation to define an easy-to-use set of metrics to measure the quality of integration architectures with a focus on non-functional requirements.

In section 3 the authors describe the synthesis of the measurement model for system integration designs that could potentially fulfil the defined goal and prove the authors thesis explained above. Hereby the authors focus on the classification by technology and provide each with a scoring for all metrics based on strengths and weaknesses of the respective integration technology. This initial analysis is being conducted using the authors own empirical knowledge and professional experience. The authors then describe their concept on how to use the classification and quality metrics to measure actual integration designs. For this, users of the measurement model need to first define their use-case-specific requirements profile and then compare this with the measurements of different integration designs made according to the above described process.

Section 4 describes the validation of the thesis. The validation is conducted as a qualitative experiment using a real-world subject provided by one authors professional background. To ensure reproducibility, transparency and validity of the experiment, the authors describe the application of the measurements model step-by-step. The outcomes of this experiment are being discussed subsequently by comparing the results of the measurement with the observed strengths and weaknesses of the real integration design.

Even though it is required to conduct additional case studies to test the approach in a broad and diverse setting, these first results would indicates if the authors' approach is potentially valid and fit-for-purpose. At the end of this paper, the authors discuss the outcomes of their experiment, highlight the strengths and limitations of their approach and line out where further elaboration and scientific improvements is required.

2. Analyse

2.1. Classification of system integrations

The authors see the clear classification of integration design as the perquisite for analysing their architecture fit. According to the TOGAF Standard, architectures can be sliced in vertical and horizontal segments. While horizontal segmentation is determined by breath of the subject matter in scope and the architecture domain, vertical segmentation is determined by the level of detail that applies to the architecture analysis [10, chapter 4.2].

In context of integration design, the authors identify three relevant levels of detail:

- **Integration landscape:** holistic view over all relevant applications and their integrations
- **Integration pattern:** isolated view on integrations of one particular system with it's connected systems
- **Integration technique:** isolated view on the technical implementation of one particular integration between two systems

Following this logic, integrations can be classified by analysing the topology of the landscapes they form [16], by the paradigms that are being used to design the integration between individual systems [17], and by the technology that is used to implement exchange between two systems [18, chapter 8].

2.1.1. Integration landscape: classification by topology

The integration landscape contains the highest level and describes all integrations of the enterprise in scope. Depending on the horizontal segmentation, this can be a whole organization and it's partners, or an isolated part of that ecosystem [3, chapter1.2].

The authors transfer the classification of network topologies to integration landscapes. The applications are represented by nodes and the integrations are the links between them. The determining characteristics of communication networks are the control strategy and the flexibility of communication [16]. The control strategy describes whether there is a central component that can be used to control the exchange between the systems or whether the integration landscape is designed in a distributed manner. The flexibility of communication describes whether there integration pathways are designed excursively, or if there are distributed elements, that would allow systems to use different integrations for the same use case

By applying this logic, the authors classify integrations according to the following topologies:

- **Hub:** all applications connect to one central system (central control, exclusive communication) [19, page 26]
- **Bus:** all applications use the same middleware and exchange data and services directly (central control, distributed communication) [19, page 24]
- **Tree:** one applications is directly connected to several systems without cross-linking (distributed control, exclusive communication) [19, page 27]
- **Mesh:** all applications can be connected to each other, allowing for redundant integration pathways (distributed control, distributed communication)[19, pages 26-27]

2.1.2. Integration pattern: classification by paradigm

By looking at integrations in isolation, different paradigms can be identified. For this, the authors applies a classification of data flows that has been initially described for federated database systems, depending on how information is being integrated and how the exchange is initiated. Information flows are classified as either materialized, meaning that all data is being physically copied over, or virtualized, meaning that the consuming application references data from the providing application [20, page 86-87]. Initialisation can be proactive or reactive. Proactive communication means, that or the consumer actively pulls the required information e.g. on demand or in certain time periods. Reactive communication means, that provider pushes the information to the consumer e.g. on change [20, page 87].

By applying this logic, the authors classifies integrations according to the following paradigms:

- **Storage-driven:** applications that are actively copying data from other systems (materialized information exchange, proactive initialization)
- **Service-driven:** applications that are actively referencing information from other systems on demand (virtualized information exchange, proactive initialization)
- **Event-driven:** applications that are receiving an information from a provider (reactive information exchange - exchange can be virtualized or materialized)

2.1.3. Integration technique: classification by technology

On the technical level integrations can be categorised in numerous ways. In order to stay technology-agnostic, yet be able to apply the model to common integration patterns, the authors choose to use established standards for categorising the technical implementation of the information exchange. In this context, DIN 44302 provides a generic definition of relevant components in information processing, data transmission, and data communication [21]. An integration is in this context interpreted as a special type of communication that can be defined as the exchange between two transition devices. DIN 44302 provides a logic to categorise this exchange according to the format that is being used and the way the transmission is being initiated [18, chapter 8]. Transmission formats can be for example file-based, web-based, or also table-based, while the transmission can be initiated by the sender (push) or the receiver (pull).

By applying this logic, the authors classify integrations into the following technology classes:

- **File Exports:** applications create data-files and send it to the receiving applications (file-based push communication)
- **File Imports:** applications provide data-files on demand (file-based pull communication)
- **Webhooks:** applications broadcast changes to applications in the integration landscape using web-based communication (web-based push communication)
- **Webservices:** applications provide information on request using web-based communication (web-based pull communication)
- **Change-Data-Capture:** applications inform connected systems about changes on database level (table-based push communication)
- **Database Connection:** applications allow other applications to connect to their database (table-based pull communication)

This classification allows an initial comparison of different approaches without having to analyse the technical implementations in depth. To illustrate how technical implementations of system integrations can be classified into the classes defined above depending on the initiation and format of transmission, the authors have added exemplary integration approaches to Figure 2 and grouped them accordingly by class. Following the logic above described, all file-based integration approaches like XML-exports or FTP dumps are classified either as *File Imports* or *File Exports*, depending if the source system is actively sending the information (push), or if the consuming systems are collecting the information themselves (pull). Another example are REST-APIs, Graph API or SOAP-based integrations. These are all classified as *Webservices* as the format is web-based and the consuming systems actively initiate the request / transmission (pull).

2.2. Metrics for system integrations

The design of any software system is fundamentally driven by functional and non-functional requirements [7, p. 12]. Functional requirements describe the essence of the software's purpose while non-functional requirements are usually mentioned more implicit [22, p. 9]. A design needs to address both and non-functional requirements to be useful [23, p. 13].

In context of measuring the architecture fit of integration designs, the authors concentrate on non-functional requirements. While the literature is rich in different measurement approaches, the authors aim to identify metrics that are relevant to measure strengths and weaknesses of

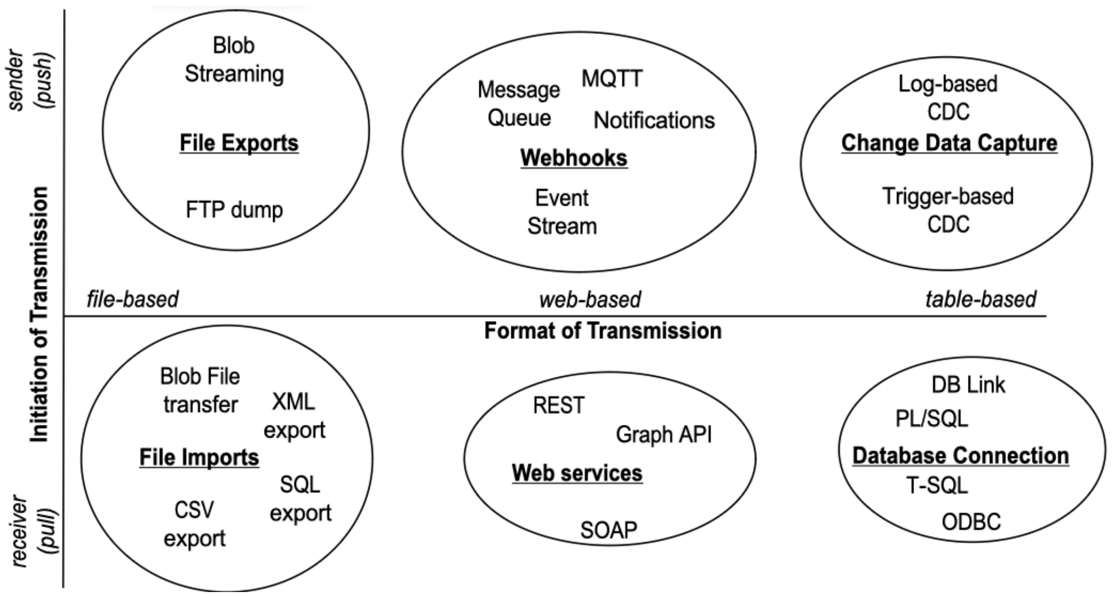


Figure 2: Exemplary classification of different integration technologies [own graphic]

integration designs. Such generic metrics can be found in standards like the ISO 25000 [11], ISO 27000 [1] and the BSI 200 [24] series.

ISO 27001 defines information security as "[A measure to] ensure the *Confidentiality*, *Availability*, and *Integrity* of information. Information security involves the application and management of appropriate controls that involves consideration of a wide range of threats, with the aim of ensuring sustained business success and continuity, and minimising consequences of information security incidents." [1, page 13] These three dimensions of information security are combined with measuring the performance of data exchange: *Throughput* and *Latency* [25, chapter 2]. In addition flexibility of the integration design is required to ensure operational integrity despite adjustment in isolated components [26, pages 33-34]. This corresponds to the system-dependent quality metrics for data products defined in ISO 25012 [11] as Accessibility (*Availability*), Compliance and Traceability (*Integrity*), *Confidentiality*, Efficiency (*Throughput* and *Latency*) and Precision (*Flexibility*).

The respective metrics are described in the following subsections.

2.2.1. Confidentiality

Confidentiality means that the owners have the control over the data they provide.

An integration with a high level of control is designed in a way that allows to limit the access to information and data flows can be monitored, managed, and adjusted [1, page 3-4].

2.2.2. Availability

Availability is a key measure for the robustness of an integration and defines whether business processes can continue or are at risk in case of a system failure [24, pages 24-25].

Integrations with a high level of robustness are designed in such a way that even if the source system cannot be reached, the required data to continue operations in connected systems is still available.

2.2.3. Integrity

Integrity is defined as the ”property of accuracy and completeness” [1, page 5]. In context of integration design, this means that ”accuracy and consistency [of data] is preserved regardless of changes made” across all integrated systems [25, chapter 2].

Integrations with a high level of integrity are designed in a way that data cannot be out of sync, outdated, incorrect, or incomplete.

2.2.4. Flexibility

Flexibility is the ability to adapt to a changing environment and address those in a resilient way while ensuring business continuity [27, page 14].

A high flexibility in context of integration architectures means, that structures can be easily adjusted or changed while the integration landscape itself remains intact throughout this process.

2.2.5. Throughput

Throughput is defined as ”the amount of work performed by a computer system over a given period of time” [1, page 5]. In context of integration design, this describes the performance of the integration and measures how much data can be transferred in a given time window.

High throughput integrations allow for the transfer of large datasets in a relatively small amount of time.

2.2.6. Latency

Latency is defined as the ”time interval between the instant at which an instruction control unit initiates a call for data and the instant at which the actual transfer of the data starts” [25, chapter 2] In context of integrations, this concept can be transferred to measure how ‘outdated‘ integrated data can be. Low-latency integration designs implement an immediate update of information in all connected systems, while high-latency integrations allow data to be updated in larger time frames.

3. Construction

3.1. Measuring integration designs

Integration designs come with different conceptual strengths and weaknesses. The authors use the metrics introduces in Section 2.2 to indicate how well respective integration designs meet those non-functional requirements on a conceptual basis. The authors apply a qualitative measurement on the following ordinal scale:

- **1:** very weak
- **2:** weak
- **3:** strong
- **4:** very strong

Very strong (4) describes the most desirable state, whereas **very weak (1)** describes the least desirable state. **Strong (3)** and **weak (2)** describe a tendency to the other direction. The desired states for each metric are described above in Sections 2.2.1 - 2.2.6.

The metrics can be applied to analyse architecture patterns on any granularity level using the taxonomy described in the above Section 2.1 to classify the respective integration designs. This analysis of integration designs can be performed in isolation at each granularity level or across different architecture levels by combining topology, paradigm and technology. The authors choose to limit the application of the measurement model in this paper to the level defined by the classification by technology as described in Section 2.1.3.

In order to conduct a first validation approach, the authors obtain these initial scoring through applying their practical experience in software engineering in combination with input from subject matter experts and IT architects from the corporate environment. While this light approach is acceptable for an initial application, the authors clearly acknowledge the potential need to conduct additional empirical research on conceptional strengths and weaknesses of each integration design pattern to further improve the validity and scientific robustness of the approach. The results of the analysis are described in the following Subsections 3.1.1 - 3.1.6 and summarised in Table 1. The determined values for these metrics are later applied to conduct the measurement model case study.

3.1.1. File Exports

File Exports implement a file-based push communication. Applications following this method create data files and send them to the receiving applications. Examples are blob streaming, or FTP dumps.

The authors qualify file exports as **very strong (4)** in *Availability* and *Throughput*, as large files can be easily exchanged to communicate large amounts of information in relatively small time frames.

Flexibility and *Integrity* are set to **strong (3)** The files can be easily adjusted and interchanged. Files can also be structured in a way to contain a full dataset that is compliant in itself.

Confidentiality and *Latency* are measuring **weak (2)** for file exports. Due to the file-based format there is the risk of losing control over the data without additional measures taken. Once

Table 1
Outcomes of integration design measurement per technology class

	Confidentiality	*Availability*	*Integrity*	*Flexibility*	*Throughput*	*Latency*
File-Exports	2	4	3	3	4	2
File-Imports	2	4	1	3	4	1
Webhooks	4	3	4	4	2	4
Webservices	4	2	4	4	2	3
Change-Data-Capture	3	3	3	2	3	4
Database Connection	2	2	4	2	3	4

data has been exported and transferred to a remote storage destination, the providing system does not have any possibility to control the further usage of this dataset. In addition, depending on the actual implementation, files provided to downstream systems might be outdated soon. Also, parsing errors can occur, making this kind of integration less reliable.

3.1.2. File Imports

File Imports implement a file-based pull communication. Applications following this method provide data files on demand, whenever an integration is initiated by the consuming systems. Examples are classical XML-, CSV-, and SQL- exports, as well as blob file transfers.

Similarly to File Exports, the authors see the strength of File Imports in a **very strong (4)** *Availability* and *Throughput*, as well as a **strong (3)** *Flexibility*.

For the same reasons as described above *Confidentiality* of this file-based communication format is rated only **weak (2)**.

File Imports are triggered by the consuming systems on demand, rather than having the ability to push updates proactively on change into the integration landscape. This pull communication style is why the authors choose the value **very weak (1)** for *Integrity* and *Latency*.

3.1.3. Webhooks

Webhooks implement a web-based push communication. Applications following this method broadcast changes to applications in the integration landscape and push this data to the consuming systems using a web-based communication format / protocol. Examples are event streaming, message queuing protocols like MQTT, or simple notifications sent by the source systems.

The authors sees clear strengths of this class of integrations in *Integrity*, *Latency*, *Confidentiality* and *Flexibility*, which receive the value **very strong (4)**. The reason for this is the push communication that ensures that updates are propagated immediately across connected systems. This ensures that data is always in sync and up to date, hence high integrity and low latency is given. The application can also change, adjust or end the broadcast of the respective updates and the consumers are well known and the information flows monitorible, a very high control, hence confidentiality is given. In addition, the abstraction and standardisation using web-based formats allows for a very flexible adjustment of applications while keeping the interfaces structurally unchanged.

Availability is ranked as **strong (3)**. The whole integration design depends on the ability of the application to send and the connected systems to receive the information. In case of failure this can be however mitigated with additional measures like message queues that persist the updates and manage temporary downtime of integration components.

The downsides of webhooks are a **weak (2)** *Throughput* in comparison to other integration designs due to the web-based communication protocols.

3.1.4. Webservices

Webservices implement a web-based pull communication. Applications following this method provide information on request using a web-based communication format / protocol. Examples are REST, SOAP, or Graph APIs.

Similarly to Webhooks, the authors sees the strengths of Webservices in a **very strong (4)** *Confidentiality*, *Integrity* and *Flexibility*.

Latency receives a slightly weaker value than proactive webhooks, as connected systems need to actively reach out for updates to the source system. This, however, is only the case if data is not fully virtualized. This is why the value for *Latency* is set to **strong (3)**.

Throughout and *Availability* receive the **value weak (2)**. Similarly to webhooks, webservices use web-protocols and require the providing system to respond when the service is required. However, since communication is only initiated on demand and does not require a ‘permanent‘ connection, shorter down times can be handled slightly better.

3.1.5. Change-Data-Capture

Change-Data-Capture implement a table-based push communication. Applications following this method use database technology to inform connected systems about changes on database level.

The authors identifies Change-Data-Capture as a very balance integration technology. It is valued **very strong (4)** in *Latency* and **strong (3)** in *Confidentiality*, *Availability*, *Integrity*, and *Throughput*.

Flexibility however is **weak (2)**. Table-based communication format requires a very strong coupling of systems so that changes to the underlying systems require the integrations to be adjusted as well.

3.1.6. Database Connection

Database Connection implement a table-based pull communication.

Applications following this method allow other applications to connect to their database and interact with it as required by the consuming systems, for example by using ODBC, or DB Link.

Similar to Change-Data-Capture, the integration method of database connection is **very strong (4)** in *Latency* and **strong(3)** in *Throughput*.

When using database connections, the entire dataset is always available by having direct access to the database. This, however, also as requires the database of the source system to be available at any given time for the integration to work. This is why *Integrity* is set to **very strong (4)** while *Availability* has the value **weak (2)**.

3.2. Calibrating the measurement model

Applying the metrics to an individual use case requires the measurement model to be calibrated.

3.2.1. Requirements weighting

To account for differences in non-functional requirements, the dimensions for the metrics described in 2.2 can be adjusted according to an individual rating. The authors apply the method of relative weighting in which the importance of every metric is compared to every other metric. Table 4 shows a real-world example weighting in context of the case study conducted

The authors are using a simple three-step scale from **1** (is less important), over **2** (has the same importance) to **3** (is more important). If the metric in the row is less important then the metric in the column, the rating is **1**. Same importance is indicated by a **2**, while a **3** indicates, that the metric in that row is more important then the metric in the column. The same metrics are not compared to each other, hence those fields are left empty. By comparing each metric and adding up these values, each metric gets assigned points that correspond to its relative importance. The additional columns labeled 'Points' and 'Percentage' indicate this absolute and relative importance of the metric in the respective row. 'Points' contains the total of all values in that row, while 'Percentage' divides this by the total of all values in the 'Points'column.

3.2.2. Requirement profile

The authors initially planned to simply multiply the relative importance with the values for the respective metrics to find out which integration design has the best fit. However, only multiplying the relative weighting as a percentage with the values per metric would not result in the best fitting integrations to come out on top, as high values for some metrics would over-compensate critical shortcomings in other dimensions of the integration design. Instead, we need to measure which integration archetype fits best to the given requirements. This means that all metrics should at least reach the minimal requirements, and not reaching them in one metric cannot be compensated with over-delivering in another metric. To address this, an additional transformation step is required: translating the requirements weighting results into a requirement profile.

The requirement profile contains minimal target values on the same four-step scale from **very weak (1)** to **very strong (4)** that has been introduced to measure integration designs in Section 3.1. The requirements weighting has 'Points' assigned to every metric that need to be translated onto this four-step scale. To do so, all possible values on the 'Points' scale need to be associated with the respective values on the 'requirement profile' scale.

Table 3 shows the variables that are being used for scale transformation.

The variable *rel(min)* represents the lowest possible rating for the importance of a metric. As we are using an *n* of six metrics, the lowest possible rating *r(min)* is 1 and we are not comparing

Table 2
Mapping ‘Points‘ from requirements weighting to the four-step measurement scale

Points	5	6	7	8	9	10	11	12	13	14	15
Value	0.0	0.4	0.8	1.2	1.6	2.0	2.4	2.8	3.2	3.6	4.0
Target	very weak (1)			weak (2)			strong (3)		very strong (4)		

Table 3
Values used for scale transformation to define the use-case-specific scale requirements profile

Transformation Variable	**Value**	**Comment**
rel (min)	5	*lowest possible rating for the importance of a metric*
rel (max)	15	*highest possible rating for the importance of a metric*
rel (step)	10	*possible intervals on the original scale [rel (max) - rel (min)]*
target (step)	4	*possible intervals on the target scale [1,2,3,4]*

the same metric to each other, the lowest possible ‘Points‘ value would be 5 (*(n-1)*r(min)*). It would mean that all columns in one row of Table 4 contain the value **1**.

The variable *rel(max)* represents the highest possible rating for the importance of a metric. As we are using an *n* of six metrics, the highest possible rating *r(max)* is 1 and we are not comparing the same metric to each other, the highest possible ‘Points‘ value would be 15 (*(n-1)*r(max)*). It would mean that all columns in one row of Table 4 contain the value **3**.

The variable *rel(step)* represents the possible intervals on the original scale and is simply the difference between *rel(max)* and *rel(min)* (*rel(max) - rel(min)*). In this case the value for *rel(step)* is 10.

The variable *target(step)* represents the possible intervals on the target scale. As we are using the four-step scale from **very weak (1)** to **very strong (4)** this value is 4.

The scale transformation is done by taking the theoretical minimum value in the ‘Points‘ column *rel(min)* as the null-point of the scale and taking the possible maximal value for ‘Points‘ *rel(max)* as the end of the scale and slicing it into the number of possible intervals on the target scale *target(step)*. The scale transformation is conducted by subtracting *rel (min)* from the respective value in the ‘Points‘ column, dividing the result with *rel (step)*, multiplying this with *target (step)* and rounding up to the next possible value on the target scale. With the lowest possible value being 5 and the highest one being 15, the difference between the minimal 5 points and the maximal 15 points equals 10. Mapping this to a 4-step ordinal scale means, that 2,5 points more then 5 equal 1 step on the scale.

As shown in Table 2, ‘Points‘ 5,6, and 7 maps to **very weak (1)**, while ‘Points‘ 8, 9 and 10 map to **weak(2)**, ‘Points‘ 11 and 12 map to **strong(3)** and ‘Points‘ 13, 14, and 15 map to **very strong (4)**.

Table 5 shows the respective values used on context of the conducted case study.

3.3. Identifying the right integration design

To identify the integration design with the best fit, the requirement profile is compared to the possible integration designs in the option space. The requirement profile defines which target values should be minimally reached for each metric. An integration design is recommended, if the value for each metric is higher or equal to the minimal requirement.

Other options can also be considered, while the gap between the measured value and the minimal requirement indicates where conceptual downsides of the design pattern need to be addressed and compensated for.

In the illustrated example shown in Table 5, the value for *Confidentiality*, *Availibility*, *Integrity* and *Latency* needs to be **strong (3)**, while the remaining metrics are less important. Table 6 in Section 4 shows all integration designs according to the classification performed in Section 2.1 and the measurement described in Section 2.2 with the coverage gaps being present in the brackets. In this example, the ideal integration design implements Change-Data-Capture, meaning that the source system actively keeps the connected systems in sync on database level.

This model can be extended to ‘integration pattern‘ and ‘integration landscape‘. Every architecture design pattern from every level can be combined with other patterns on other levels e.g. to address shortcomings or overcome integration challenges. This adds up to 72 possible architecture archetypes that are resulting of a combination of topology, paradigm and technology. For reasons of simplicity, the authors has limited the scope to the architecture granularity level previously described as ‘integration technique‘.

The option space can also be limited because the respective systems in scope would not support certain integration designs or there are other restrictions regarding topology, paradigm or technology in place.

4. Validation

4.1. Description of case study

In a first attempt to validate the approach described above, the authors conducted a qualitative experiment in an industry context. The method is applied to a real-world example and the results compared with observations made with the actually designed and implemented integration.

4.1.1. Integration needs

The subject of this experiment is the integration between two operational systems that share similar business data objects. System A is an application to plan, manage and oversee clinical trials. System B is an application for site monitoring and document management in context of clinical trials. System A provides the business data objects ‘Project‘, ‘Investigational Product‘, ‘Study‘, ‘Study Country‘, ‘Subject‘, and ‘Subject Visit‘ to System B. System B provides the business data objects ‘Monitoring Visit‘ and ‘Monitoring Report‘ to System A. Both systems use and update the business data objects ‘Investigator‘ and ‘Study Site‘. The integration challenge is, that workflows in both systems depend on the data from the other system and that there is a shared usage of data objects that need to be kept in sync in order not to risk the operational integrity. This means, that changes in System A need to be reflected in System B in near real time to allow for workflows to be conducted in both systems in parallel.

In the context of the measurement model described above, the authors expects a requirement profile that shows the priority of *Availability*, *Integrity* and *Latency*. *Flexibility* is less important, as both integrated systems are mature and do not change with a high velocity. Also, a high throughput is not required, if synchronisation would happen in near-real-time. A strong throughput would be required if the data integration happens in a virtualized way to ensure a good operational performance of the applications. *Confidentiality* is currently not in focus, as the exchange between the systems is implemented in an isolated secure environment and the data exchange is being closely monitored. However, it could play a role in selecting the right integration design which could minimize these operational efforts.

4.1.2. Current implementation

The current implementation is a scheduled *File Import*. Several times a day, the integration middleware extracts the data from System A, compiles a file and uploads this to System B. The same process is being conducted to extract data from System B and ingest it to System A.

This integration is very simple, but does not allow for a real-time synchronisation of data. Due to this shortcoming, there are only limited workflows that are implemented across the systems. For example, when data is being updated in another system the workflow has to pause until the synchronisation via file import has been completed. In addition, the same data object is being altered in both systems during between two synchronisation cycles, one of the updates is being overridden with the latest data causing potential problems with the completeness and integrity of the datasets. To mitigate this issue, the respective fields have been set to read-only in one of the systems, so that users have to log into the other system to make updates. This however impacts the operational efficiency of the business, which is the major reason why the respective integration has been identified for redesign. An additional downside of the current integration design is the risk of errors connected to the compilation and parsing of the files. This could lead to the integration to fail or to data being corrupted.

Based on these observations, the authors expects the option *File Import* to show a gap in *Latency* and *Integrity*, while *Availability* should be above the required threshold.

The ideal scenario would be an integration design with a proactive initiation of communication, so that the systems push the changes to the other system immediately on change. A virtualized integration with *Webservices* or using a *Database Connection* could also be good options. This however would need to be combined with the data being saved periodically in the consuming systems, so that with this fallback option business processes can be performed, even if the other system is currently not available.

4.2. Applying the measurement model

To measure which integration design is best suited, the authors needs to define a requirement profile for the use case described above. As described above in Section 3.2.1, the first step is to weigh the requirements by comparing their importance with each other. In the next step, the outcomes of this weighting is being used to define the requirement profile. Based on the importance of each metric for the specific use case, the ideal design would need to score **very strong (4)**, **strong (3)**, or can be **weak (2)**, or **very weak (1)**. In the last step, the requirement profile is compared with the respective integration design classes metric by metric. The classification is hereby done by following the taxonomy defined in Section 2.1. The

Table 4
Weighting of requirements by comparing them with each other. *[Scale]* ***1:*** *is less important,* ***2:*** *has the same importance,* ***3:*** *is more important*

	Confidentiality	*Availability*	*Integrity*	*Flexibility*	*Throughput*	*Latency*	**Points**	**Percentage**
Confidentiality	-	**2**	**2**	**2**	**3**	**2**	11	18%
Availability	*2*	-	**2**	**3**	**3**	**2**	12	20%
Integrity	*2*	*2*	-	**3**	**3**	**2**	12	20%
Flexibility	*2*	*1*	*1*	-	**2**	**1**	7	12%
Throughput	*1*	*1*	*1*	*2*	-	**1**	6	10%
Latency	*2*	*2*	*2*	*3*	*3*	-	12	20%

scoring itself is based on the initial ratings made by the authors in Section 2.2. As these ratings are rather subjective, they might need to be adjusted depending on the context in which the measurement model is being applied.

4.2.1. Perform requirements weighting

The requirements weighting is conducted by comparing each metric to each other, as described in Section 3.2.1. The results of the use-case-specific requirements weighting are shown in Table 4. Every row and column represents one of the metrics for measuring non-functional requirements described in Section 3.1. The first group of columns shows the same metrics. The values indicate, if the metric in the row is having a lower (**1**), similar (**2**), or higher (**3**) importance then the metric in the respective column. Those values are summed up per row and shown in the column labeled 'Points'. The column 'Percentage' shows the same value in relation to the total of all values in the 'Points' column.

As shown in Table 4, *Confidentiality* has the same importance as most of the other metrics and is more important than *Throughput. Availability* is equally important as *Integrity* and *Latency*, while it is more important than *Flexibility* and *Throughput. Integrity* is more important than *Flexibility* and *Throughput*, while having the same importance as *Latency. Flexibility* is equally important as *Throughput* and less important then *Latency.*

This results in *Availability*, *Integrity* and *Latency* to have the highest importance (12 Points / 20%), together with Confidentiality (11 points / 18%), while *Flexibility* (7 Points / 12%) and *Throughput* (6 Points / 10%) have a fairly low importance for this use case.

4.2.2. Define requirement profile

The requirement profile is defined by mapping the points resulting from the weighting matrix illustrated in Table 4 to the ordinal scale used to measure the integration designs.

To measure the architecture fit of different integration technology classes, the relative requirement weighting shown in Table 4 need to be translated into a measurable requirements

Table 5
Use-case-specific requirement profile *[Scale]* ***1:** very weak, **2:** weak, **3:** strong, **4:** very strong*

	Points	**Value for scale transformation**	**Minimal Target**
Confidentiality	11	2.4	strong (3)
Availability	12	2.8	strong (3)
Integrity	12	2.8	strong (3)
Flexibility	7	0.8	very weak (1)
Throughput	6	0.4	very weak (1)
Latency	12	2.8	strong (3)

profile. For this, the values in the 'Points' column of Table 4 are translated to the four-step scale used for the measurement by applying the scale transformation described in Section 3.2.2.

The results of this transformation is shown in Table 5. Each row lists the requirement metrics with the respective weighting 'Points' calculated above, the transformed 'Value' based on the scale transformation and the resulting minimal requirements target.

In this case, 'Points' 11 and 12 map to **strong(3)** and 6 and 7 map to **very weak (1)**.

The use-case-specific requirement profile shown in the last column of Table 5 shows a similar importance in *Availability*, *Integrity*, *Latency*, and *Confidentiality*, all rated **strong (3)**. *Flexibility* and *Throughput* are rated **very weak (1)** and are not relevant for the selection of the right integration design. This means that the desired integration design needs to synchronize data in an immediate and robust fashion while the possibility to easily adjust the integration or exchange of large dataset can be neglected. This maps well to the expectation described above and shows that the described method is able to provide a meaningful requirement profile.

4.2.3. Identify the right integration design

In the last step, the requirement profile is compared with the measured integration designs metric by metric. Table 6 shows all possible integration designs classified on the technical level, as described above in Section 2.1. The columns contain the values for each metric. The first row contains the requirements profile with minimal target values for each metric and is highlighted in bold. The following rows then contain the values for the respective integration design classes. For those metrics where the value is below the minimal target defined in the requirement profile, the requirements gap is indicated by the number in the brackets. Rows containing an integration design class without and architecture gap are highlighted in bold. Those are the integration designs that the measurement model identified as good architecture choices. As mentioned above, this method can be extended to additional granularity levels, allowing a more complex holistic measurement of the integrations. So, it can also be applied holistically to the whole integration landscape and topologies and paradigms in which the single integration is embedded.

4.3. Interpretation of results

The results of the measurement in Table 6 show, that for this particular integration either *Webhooks* or *Change-Data-Capture* would be a recommended integration design. While *Change-*

Table 6
Outcomes of integration design measurement per technology class
(calculated requirement gap in brackets) *[Scale]* ***1:** very weak, **2:** weak, **3:** strong, **4:** very strong*

	Confidentiality	Availability	Integrity	Flexibility	Throughput	Latency	*Requirement Gap*
min. Target	**3**	**3**	**3**	**1**	**1**	**3**	
File-Exports	*2 (-1)*	4	3	3	4	*2 (-1)*	*(-2)*
File-Imports	*2 (-1)*	4	*1 (-2)*	3	4	*1 (-2)*	*(-5)*
Webhooks	**4**	**3**	**4**	**4**	**2**	**4**	
Webservices	4	*2 (-1)*	4	4	2	3	*(-1)*
Change-Data-Capture	**3**	**3**	**3**	**2**	**3**	**4**	
Database Connection	*2 (-1)*	*2 (-1)*	4	2	3	4	*(-2)*

Data-Capture allows for a higher *Throughput*, *Webhooks* are stronger in *Integrity* and *Flexibility*. If one of the systems needs to be changed or adjusted, *Webhooks* would be the better option. It also indicates that the currently implemented integration design using *File-Imports* has gaps in *Confidentiality*, *Integrity* and *Latency*. This matches the observed weaknesses of the current implementation. Also, the recommended integration designs match with the actual expectations made form observing this real-world example.

5. Conclusion and outlook

Concluding on the research outcomes, the authors see clear signs for the illustrated approach to be valid and usable to tackle the challenge of modernizing integration landscapes in an efficient manner. The first practical application implies that model is able to provide meaningful guardrails for a fit-for-purpose integration architecture and can be used to measure the technical fit of integration designs. With this, the authors are making a valuable contribution to the enterprise architecture and software measurement community.

The authors have worked out a clear taxonomy for integration architecture segmentation and classification and have also have identified meaningful metrics to measure integration architectures by referring to established software measurement practices and standards. Based on this, the authors have described a measurement model that meets the desired ease of use and has provided valid outcomes in context of the first case study. The application was described step-by-step to ensure transparency and reproducibility of results. Beyond what is described in this paper, the method was applied to two additional scenarios to further increase the robustness of the research outcomes and investigate the validity of the approach. In both cases the measurement model provided plausible results and helped to improve integration designs.

Although the first outcomes are very positive, the authors also see the need for further maturation and refinement. Addressing the following gaps and weaknesses needs to be subject of future research.

It needs to be clarified, if applying the approach creates useful insight for practitioners, or if the method is too simplistic. While the approach is designed to significantly reduce the complexity that practitioners have to cope with, it's simplicity could also undermine it's usefulness. The model is intended to quickly measure the architecture fit for numerous integrations, provide provide architecture guardrails and help practitioners choose the right design patterns. It is not meant to be used for detailed design discussions. To clarify this question, the measurement model needs to be discussed with practitioners and further qualitative and quantitative empirical research needs to be conducted.

Furthermore, it needs to be demonstrated, that the approach is transferable, effective and applicable across diverse scenarios. To address this, additional case studies need to be conducted, ideally in different organisational areas or enterprises. In a more holistic analysis across multiple levels the authors could apply the measurement model on other granularity levels to measure the architecture fit of different integration topologies and paradigms. These outcomes could be used to investigate how a combination of topologies, paradigms and technologies mitigates weaknesses on individual levels. Also it could be investigated if a higher resolution of the scale or additional metrics might be beneficial to receive better measurements, or if this contradicts the goal of easy applicability.

Finally, the scoring for the classes of integration designs requires further elaboration. The rationale behind the initial scoring of the respective integration technology classes made in Section 3.1 of this paper is mainly based on the authors professional experience and requires a more solid empirical foundation. Also the scoring needs to be extended to the level of topology and paradigms. Additional empirical research on strengths and weaknesses of the different design patterns is required to improve the accuracy of the scoring, make the measurement model more robust and remove potential bias. Respective scientific insights could be generated by conducting interviews with experts, surveying IT professionals or enriching the results with outcomes of AI-based research agents.

References

[1] ISO/IEC 27000, Information technology — security techniques — information security management systems., 2018. URL: https://standards.iso.org/ittf/PubliclyAvailableStan-dards/c073906_ISO_IEC_27000_2018_E.zip.

[2] J. Schekkerman, How to Survive in the Jungle of Enterprise Architecture Frameworks: Creating or Choosing an Enterprise Architecture Framework, second edition ed., Trafford, Victoria, BC, 2004.

[3] TOGAF Standard, Introduction and core concepts, 2022. URL: https://pubs.opengroup.org/togaf-standard/introduction.

[4] F. Vishal Dalal and Krishnakanthan, K. Münstermann, B. and R. Patenge, "tech debt: Reclaiming tech equity", 2020. URL: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/tech-debt-reclaiming-tech-equity.

[5] Deutsches Institut für Wirtschaft (DIW), Digitalisierungsindex 2022, 2022. URL: https://www.de.digital/DIGITAL/Redaktion/DE/Digitalisierungsindex/Publikationen/publikation-digitalisierungsindex-2022-langfassung.pdf.

[6] D. Carr, S. Else, State of Enterprise Architecture Survey: Results and Findings, Enterprise Architecture Professional Journal, 2018. URL: https://eapj.org/wp-content/uploads/2018/05/EAPJ-Special-Edition-State-of-EA-Survey.pdf.

[7] Symons, Charles, Cosmic - a guide to software size measurement, 2020. URL: https://cosmic-sizing.org/wp-content/uploads/2020/08/Measuring-software-size-v1.0-August-2020-1.pdf.

[8] S. Hartenstein, K. Nadobny, S. Schmidt. and A. Schmietendorf, An Approach for a Fast Cost Validation of Web-Based APIs supported by Functional Size Measurement with COSMIC, 2019. URL: https://ceur-ws.org/Vol-2476/short2.pdf.

[9] TOGAF Standard, Content framework, 2022. URL: https://pubs.opengroup.org/togaf-standard/architecture-content/chap01.html.

[10] TOGAF Standard, Applying the adm, 2022. URL: https://pubs.opengroup.org/togaf-standard/applying-the-adm.

[11] ISO/IEC 25012, Information technology — vocabulary, 2015. URL: https://www.iso.org/obp/ui/en/#iso:std:iso-iec:2382:ed-1:v2:en.

[12] K. Nadobny, Vergleich von enterprise api-management lösungen, in: Berliner Schriften zu modernen Integrationsarchitekturen - Evaluation of Service-APIs“ (ESAPI 2020), 2020.

[13] K. Nadobny, Api-fizierung von altanwendungen, in: Berliner Schriften zu modernen Integrationsarchitekturen - Enterprise Computing Conference“ (ECC 2019), 2019.

[14] M. Binzen, K. Nadobny, H. Neumann and A. Schmietendorf, Service apis als enabler einer erfolgreichen digitalisierung, in: GI Softwaretechnik Trends, Band37, Heft 3, GI Fachgruppe Measurement Data Science, 2017.

[15] K. Nadobny, Api-fizierung von legacy-systemen im kontext agiler applikationsentwicklung., in: Berliner Schriften zu modernen Integrationsarchitekturen - Evaluation of Service-APIs“ (ESAPI 2019), 2019.

[16] A. Georgi, Theorie und einsatz von verbindungseinrichtungen in parallelen rechnersystemen: Klassifizierung von verbindungsnetzwerken, 2012. URL: https://tu-dresden.de/zih/ressourcen/dateien/lehre/ss2012/tevpr_content/V2_Klassifizierung_4_on_1.pdf.

[17] U. Leser, U. and F. Naumann, Informationsintegration: Architekturen und Methoden zur Integration verteilter und heterogener Datenquellen, dPunkt, Heidelberg, 2007.

[18] J. Reese, Wirtschaftsinformatik. Eine Einführung., Springer Gabler, Wiesbaden, 1990.

[19] C. Baun, Computer Networks. Bilingual Edition: English - German / Zweisprachige Ausgabe: Englisch - Deutsch., Springer Gabler, Wiesbaden, 2019.

[20] U. Leser, S. Busse, R.-D. Kutsche and H. Weber, Federated information systems. concepts, terminology and architectures, 1998. URL: https://www.researchgate.net/publication/237380766_Concepts_Terminology_and_Architectures.

[21] DIN 44302, Informationsverarbeitung; datenübertragung, datenübermittlung, 1987.

[22] Chen, L., Babar, M.A.and NUSEIBEH, B., ”characterizing architecturally significant requirements”, 2023. URL: https://hdl.handle.net/10344/3061.

[23] J. Grundy, P. Lago, P. Avgeriou, J. Hall, I. Mistrík, Theoretical Underpinnings and Reviews, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 13–15. URL: https://doi.org/10.1007/978-3-642-21001-3_2. doi:10.1007/978-3-642-21001-3_2.

[24] BSI 200-2, Managementsysteme für informationssicherheit, 2017. URL: https://www.bsi.bund.de/dok/10027846.

[25] ISO/IEC 2382, Data quality model, 2008. URL: https://iso25000.com/index.php/en/iso-25000-standards/iso-25012.

[26] BSI 200-3, Risikoanalyse auf der basis von it-grundschutz, 2017. URL: https://www.bsi.bund.de/dok/10027822.

[27] BSI 200-4, Business continuity management, 2023. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/BSI_Standards/standard_200_4.pdf?__blob=publicationFile&v=8.