DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Information Systems

Assessing the Cost and Benefit of a Microservice Landscape Discovery Method

Ludwig Achhammer

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Information Systems

Assessing the Cost and Benefit of a Microservice Landscape Discovery Method

Bewertung der Kosten und des Nutzens einer Methode für die automatisierte Rekonstruktion von Microservice Architekturen

Author:	Ludwig Achhammer
Supervisor:	Prof. Dr. rer. nat. Florian Matthes
Advisor:	M.Sc. Martin Kleehaus
Submission Date:	15.10.2019

I confirm that this master's thesis in information systems is my own work and I have documented all sources and material used.

Munich, 15.10.2019 Ludwig Achhammer

Acknowledgments

I would like to take the opportunity to express my special thanks to the many friends, colleagues and family out there who supported me over the last six months. First of all, I want to thank my advisers Martin Kleehaus, Benjamin M. and Matheus H. for advising me throughout the creation of this thesis. Their constructive feedback was of enormous value and they inspired me and this work in many ways. My gratitude also goes to the many colleagues at the industry partner for their time, interest and valuable feedback during the interview sessions. Further, I would like to thank Nina B., my family and friends who accompanied me during this time which was sometimes tough. Last but not least, I want to thank my proofreaders Benjamin B., Anna and Viktor H. and Karla S. as well as the SEBIS chair led by Prof. Dr. Florian Matthes for giving my the opportunity to write this master's thesis.

Abstract

Over the last decades, Enterprise Architecture Management has evolved to a well-established discipline in mid-size to large enterprises with the overall goal to enable strategic transformation of IT landscapes in close alignment with business goals. Usually, such transformation is accompanied by multiple Enterprise Architecture models reaching from a current state to a long term target state.
The documentation of the current Enterprise Architecture has been a challenge ever since to the discipline as the required information is used to be scattered across multiple stakeholders and information systems. Manual data collection and maintenance is laborious, error-prone and thus, cost-intensive work. The growing adoption of agile development methodologies and closely related technologies such as microservice architectures and continuous deployment make this situation even more severe. Artifacts change faster than ever whereas the technical diversity increases and applications are distributed across multi-platform environments. Manual Enterprise Architecture documentation is no more option to cope with such fast-paced environments.
This thesis puts the focus on a novel approach for automated Enterprise Architecture model maintenance that is based on the use of runtime data originating from cloud platforms and distributed tracing. Discovered applications are enriched with business-related information by instrumenting continuous deployment pipelines which, as a result, enables to generate and maintain multi-layer Enterprise Architecture models. To assess its capabilities and feasibility, the suggested solution approach is evaluated in a real-case environment at a large German enterprise. This includes the analysis and answers of (1) how to integrate the approach into agile software development, (2) what EA model elements can be discovered using which data sources and (3) what savings and what costs this implies. Finally, a series of interviews is conducted to obtain experts' judgment of whether the suggested approach is worth being implemented and what issues remain to be solved.

Contents

List of Figures

List of Tables

List of Abbreviations

ADM	Architecture Development Method
API	Application Programming Interface
APM	Application Performance Monitoring
BCM	Business Continuity Management
BPMN	Business Process Model and Notation
BRM	Binary Repository Manager
CD	Continuous Delivery
CI	Continuous Integration
CMDB	Configuration Management Database
EA	Enterprise Architecture
EAD	Enterprise Architecture Documentation
EAM	Enterprise Architecture Management
EAMM	Enterprise Architecture Model Maintenance
EPC	Enterprise Private Cloud
ESB	Enterprise Service Bus
FQDN	Full Qualified Domain Name
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
MSA	Microservice Architecture
PaaS	Platform as a Service

PCF	Pivotal Cloud Foundry
PERT	Program Evaluation and Review Technique
PPM	Project Portfolio Management
REST	Representational State Transfer
SLA	Service Level Agreement
SOA	Service-oriented architecture
SaaS	Software as a Service
TCO	Total Cost of Ownership
UML	Unified Modeling Language
VCS	Version Control Service

1. Introduction

1.1. Motivation & problem statement

As today's enterprises increasingly rely on their information systems and IT infrastructure, Enterprise Architecture Management (EAM) has become a strategic discipline for improving Business-IT alignment and support business goals [11]. An Enterprise Architecture (EA) models business and IT artifacts and puts them into relation with one another. Having a thorough understanding of the inter-dependencies between these elements, strategic decisions can be derived more easily to transform the enterprise architecture into the desired direction in line with business needs [8]. This process of transformation is usually accompanied by modeling a current state, one or more planned states and a long term target state of the enterprise architecture [38].

A critical success factor to reliably capture the current state of an enterprise architecture is to build upon data that is in sync and accurately reflects the reality. The documentation of the as-is enterprise architecture, therefore, constitutes a major challenge for the EAM discipline as being found by practitioners and researchers [15].

In the context of the growing adoption of agile software development methodologies and the increased usage of microservice-based architectures, today's IT landscapes are exposed to rapid architectural changes. As a consequence, EA documentation processes are challenged all the more to keep pace with continuously changing and growing complexity of the IT landscape. Current research and practitioners are in search of an appropriate solution to this challenge and to automate the documentation and maintenance of EA models. Several solution approaches have been proposed (e.g. based on Network Scanners [16], cloud platforms [7], Enterprise Service Buses [4], cloud platforms [8] and machine learning [23]) by literature. However, most of this solutions do not cover the EA business layer and have not been reflected within real-case enterprise environments.

As part of the chair's research in this area, a novel approach to automate the EA model documentation and maintenance process within microservice-based environments was developed. The two key enabling technologies of this approach are (1) the use of cloud platform and distributed tracing data which allows the discovery of microservices and their inter-dependencies at runtime and (2) the integration into CI/CD pipelines. This thesis puts the focus on automated, pipeline-driven EA documentation. The approach uses software deployment processes as a trigger for the automated EA documentation process. Application related data collected at runtime is completed to a multi-layer EA model by the help of static information coupled to the software artifact using a configuration file.

This work follows up on a proof of concept of this novel approach and further develops it into an integral part of the EA processes within a large German enterprise. This includes the roll-out to a real-case environment and the integration into existing processes. Based on this, the goal is to critically assess the cost and benefit of this solution approach from several perspectives. The assessment should deliver prove about the feasibility and the capabilities of the suggested solution in practical use.

1.2. Research questions

As part of this work the following research questions will be handled and answered:

- RQ1: How can the suggested solution be integrated into agile development and what challenges do occur?
- RQ2: What EA model elements should be documented and to what degree can this be automated using the solution approach?
- RQ3: What are the solution's integration costs and value propositions for Enterprise Architecture Management?

RQ1 targets at the integration and of the suggested solution approach into a real-case ecosystem (processes, tools and actors) and evaluate its behavior in productive use. The goal is to identify practical challenges that arise and provide recommendations on how they might be overcome.

RQ2 will be answered as part of the case study conducted at the industry partner. By assessing the solution's capabilities within a real-case enterprise context, it will be clarified to what degree the target EA meta model can be discovered automatically. This will also reveal the limitations of the suggested approach.

RQ3 will be answered by evaluating the suggested solution approach from multiple perspectives and evaluation criteria. Judgments are based on findings made throughout the case study, an analysis of cost and savings as well as feedback obtained during a series of expert interviews. The goal is to draw a cost/benefit ratio about the suggested solution approach.

1.3. Research methodology

In design science research, evaluation activities are differentiated in *formative* vs. *summative* (Purpose: Why to evaluate an artifact?) and *ex ante* vs. *ex post* (Point in time: When to evaluate an artifact?) [42]. This master's thesis follows the design science research methodology and comprises a formative, ex post evaluation of an Enterprise Architecture documentation solution concept. Thus, this work strives to assess the values of an implemented system (ex post) based on monetary and non-monetary measures. In parallel, it still exhibits a strong exploratory and improving (formative) character as the evaluand is still at a prototype stage. 1.1 puts this into the context of a design science cycle and points out the activities covered by this work.

So far, a prototype was developed and demonstrated in a fully controlled environment where realistic enterprise conditions were absent. Therefore, before the actual *USE* phase, the prototype is brought a step back to the *CONSTRUCT* phase to solve challenges identified in *EVAL 3* which are hindering the use of the prototype in a real-case setting. In the following, the advanced solution will be integrated by a selection of agile teams working at the industry partner. This will allow to evaluate the concept under realistic circumstances and identify problems that remain to be solved as well as insights about the solution's value proposition. The evaluation will comprise a set of financial and non-financial criteria defined by [28]. Details about this are covered in chapter 5.

According to [34], data collection methods can be categorized into three degrees. Typical *first degree* or *direct* data collection methods are interviews, focus groups or Delphi surveys. The precondition is direct contact with the studied object. Raw data collections without direct contact are considered *second degree* or *indirect* methods, e.g. video observations or software monitoring. Lastly, the use of independent, preexisting data or information is called *third degree* data. This work primarily makes use of first (a questionnaire for as-is analysis and a series of semi-structured interviews for evaluation purposes) and third degree data (database exports) collection methods. Expert's estimations part of this thesis can be considered second degree of data. Details about data collection methods used in this work are covered in chapter 5.

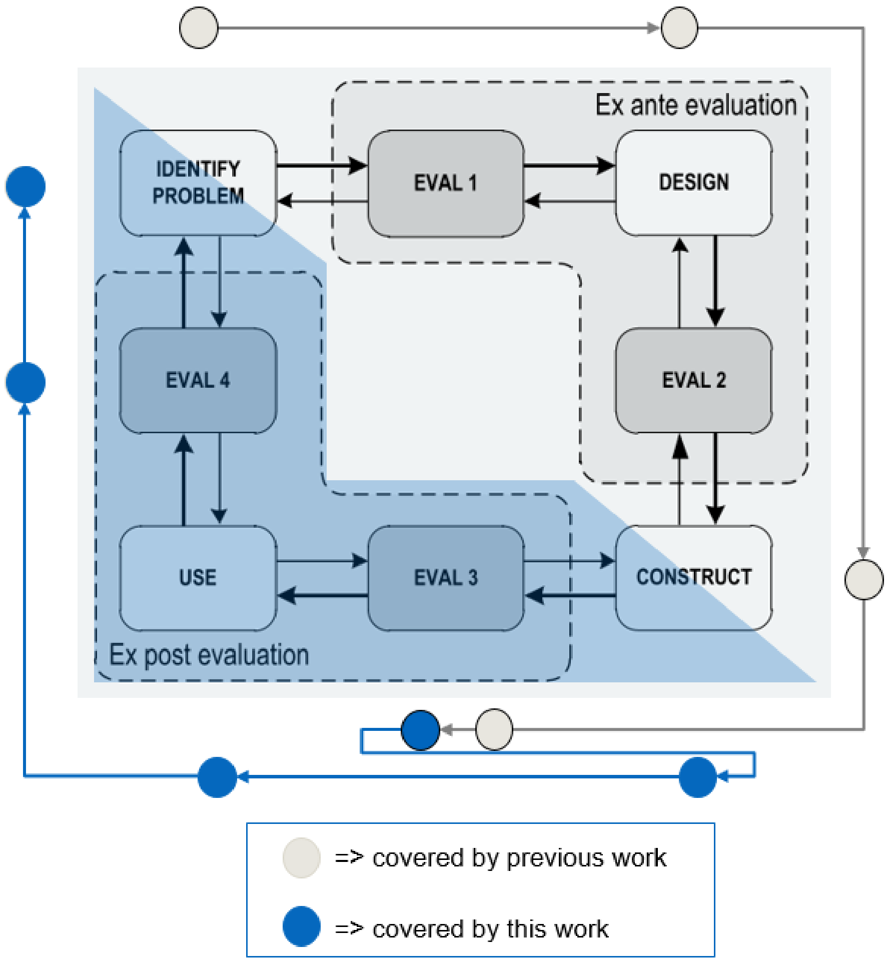


Figure 1.1.: Evaluation activities within this master's thesis, adapted from [37]

1.4. Outline

This master's thesis organizes as follows.

Chapter 2 will briefly introduce the reader into the most important terms and ideas of Enterprise Architecture Management. Additionally, it explains the most decisive trends that have an influence on this discipline which also includes certain technologies that bear challenges and opportunities as well.

Chapter 3 will introduce important related research that has an influence on this thesis and alternative solution concepts which have been suggested by literature.

At the beginning of chapter 4, the solution approach will be introduced in more detail. This includes the solution's core concepts, its architecture as well as details about the implemented prototype. Additionally, it describes the most important processes for a successful EA documentation and maintenance as well as how the solution can be seamlessly integrated into daily operation. Referring back to 1.1 this chapter covers an additional iteration between the phases *EVAL 3* and *CONSTRUCT*.

Chapter 5 explains the structure and content of the case study conducted at the industry partner. This includes the definition of goals, the scope of studied objects, the evaluation environment as well as data collection methods used.

Chapter 6 is the major part of this work. It includes an analysis of the *Status Quo*, a *Requirements Analysis* and the presentation of a *Target State*. The target state essentially describes the state that can be reached if the solution was rolled out to the entire organization. Section 6.5 shows exemplary results that prove that this is actually feasible under real-case settings. The chapter concludes with an assessment of cost and savings (section 6.6 and the evaluation interview results (section 6.7).

Chapter 7 finally generalizes the the findings and concludes to what degree posed requirements to an EAMM solution and evaluation criteria can be fulfilled by the suggested solution approach.

2. Foundation

2.1. Enterprise Architecture Management

Enterprise Architecture (EA):
The standard ISO/IEC/IEEE 42010:2011 defines the term *architecture* as "the fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution". The Open Group's enterprise architecture management standard *TOGAF* complements this by the "structure of components, their inter-relationships, and the principles and guidelines governing their design and evolution over time" [38]. On this foundation, the term **Enterprise Architecture** is defined as "the organizing logic for business processes and IT infrastructure, reflecting the integration and standardization requirements of the company's operating model". The operating model is denoted as "the necessary level of business process integration and standardization for delivering goods and services to customers" [30]. EA strives to provide a long term view of the enterprise with the purpose to "optimize across the enterprise [...] into an integrated environment that is responsive to change and supportive of the delivery of the business strategy [38].

Enterprise Architecture Management:
The term Enterprise Architecture Management (EAM) is considered a "well-defined practice for conducting enterprise analysis, design, planning, and implementation, using a holistic approach at all times, for the successful development and execution of strategy" [5]. Practically speaking, EAM typically documents the current state of EA and develops a desired target state of EA that better serves enterprise goals. EAM accompanies the transformation from the current to the target state, often passing one or more intermediate states of EA [10]. TOGAF lists improved business and IT operation, facilitated digital transformation, increased return on investment by reduced risk and improved procurement as benefits of EA [38]. Typical goals pursued by practitioners are for instance the increase in IT / Business alignment, identification of cost savings [8], [19], improved decision support [9], [41], strategic planning [7], [19], management of complexity [21], standardization and the assurance of legal compliance [8].

2.1.1. Federated Enterprise Architecture Management

According to [12] there are two strategies for the management of EA models. (1) an *holistic* EA meta model that defines all required types of artifacts. In this approach, specialized architecture models existing in the enterprise are interpreted and remodeled using the central notation. (2) the management of a *federated* EA model that feeds from specialized architectures using meta model integration. The data is either gathered on demand or by storing a copy which is periodically updated. Specialized architectures for instance are models contained in certain tools (e.g. CMDBs, network scanners, project portfolio system) are other modeling tools (e.g. BPMN, UML). The authors in [12] argue that the federated approach results in a more accepted EA model, more current data, less management effort and less risk for model misinterpretation. Examples for EA documentation approaches following the federated strategy are presented in [15] and [32]. The solution approach presented in this work also follows a federated EA management strategy.

2.1.2. Enterprise Architecture Model Maintenance

The TOGAF *Architecture Development Method* (ADM), describes a generic process of how EA can be developed and used to transform the business from an baseline architecture (*current EA*) into a desired direction (*target EA*) that is in line with business goals. To support this ambition the *Architecture Landscape* holds the "architectural representation of assets in use, or planned, by the enterprise at particular points in time". Results achieved in ADM are persisted in an *Architecture Repository* that stores "different classes of architectural output at different levels of abstraction" [38]. In the course of this work, Architecture Repositories will also be referenced as *EAM repositories*. A key precondition to successful EAM is the availability of an up-to-date and continuously maintained EA model which depicts the current state of an enterprise's architecture. The process of data gathering, modeling and model maintenance is denoted by Enterprise Architecture Model Maintenance (EAMM). This includes the initial documentation of EA elements as well as the process of continuous updating. In the course of this work, Enterprise Architecture Model Maintenance will also be referenced as *EA Documentation*.

EA documentation is a time consuming, error prone and therefore cost intensive task as it is mostly performed on a manual basis [8], [14], [15], [33]. A claim made is that existing EA frameworks and tools do not provide sufficient assistance to overcome this challenge [10]. For more than a decade practitioners and researcher seek for a reliable and efficient way to automate EA documentation and model maintenance.

Requirements and success factors to EAMM

An automated EAMM solution must satisfy various requirements to be successful. A list of such requirements was compiled first by [8] based on a literature analysis and a survey among practitioners. Results were categorized into *Architectural Requirements, Organizational Requirements, Integration/Data Source Requirements, Data Quality Requirements, Functional System Requirements* and *Non-functional Requirements*. Since then, researcher often refer back to this initial list, including [14], [15], [39], [41]. The most important requirements including supplements named by other literature are shortly summarized in the following.

An **Architectural Requirement** as per [8] is that EA data collection should be fed from federated systems. The authors [41] add that such a process should be optimized to allow an easy extension by further information sources and to reduce the need for manual inspection.

A central **Organizational Requirement** is the existence of an organizational process that regulates the EAMM process for involved individuals and tools. Such a process must be supported by technical means and incorporate quality assurance mechanisms. Furthermore the system must allow to adapt this features to preexisting processes in an organization [8].

Integration/Data Source Requirements, Data Quality Requirements subsume various functional requirements an automated EAMM system should satisfy with regards to EA discovery. Beside some others this includes the capability of the system to automatically detect (1) changes to the real-world EA, (2) changes to the infrastructure, (3) changes to and interfaces between information system as well as functionality required for (4) model transformation between source and target data structure [8].

With regards to **Data Quality Requirements** an automated EAMM system has to feature functionality to ensure data actuality, data consistency as well as handling data granularity and data validity (from a timely perspective). This either must be automated (e.g. automated conflict identification and resolution from different information sources) or provide active support to the EA team for quality assurance. Furthermore the automated propagation of changes must be supported [8]. The authors in [41] complement these requirements. They demand that data quality attributes must be measurable and controllable. Further, data sources' credibility should be known and provided along with the EA data.

Additional **Functional System Requirements** formulated by [8] are features that allow the definitions of KPIs and the ability to calculate them based on runtime information.

A **non-functional Requirement** is that the system must scale for large inputs [8].

Such a solution cannot be free of cost. It is therefore indispensable to reflect the cost/benefit ratio before implementation. Key success factors with an influence to the solution's value propositions are (1) the degree to which manual effort is reduced, (2) the surplus of EA information provided, (3) the increase of data quality and (4) the satisfaction of information demand [8]. The integration of federated data sources in this context must always be assessed with regards to their value add as the implementation and maintenance of such an integration itself is a costly task [10]. General research towards a cost-benefit assessment for automated EA data collection is set out as future work. In a survey conducted by Farwick et al., 60 percent of responses stated that the high cost incurred by EA data source integration is an issue. In parallel 28 percent claimed a low return on investment of such endeavors [10].

EAMM challenges

The authors in [33] set out the the key problems experienced by practitioners in regards to EAMM. Major issues named are enormous data collection efforts and low data quality when it comes to EA documentation. In [15] specific challenges faced with applying automation endeavors are analyzed. The authors summarize their findings in four categories. Typical **Data Challenges** faced are identifying the best fit out of a plethora of potential information sources, handling information overload and data quality issues as well as the detection and propagation of changes that are relevant to EA. [11] adds a lack of structured information sources for business layer elements and the detection of removed elements as additional challenges to this category. **Transformation Challenges** describe the difficulties with model transformation required to overcome different meta models and abstraction gaps between source systems and the EAM repository. The consolidation of ambiguous concepts imported and efforts to assure data consistency and actuality also belong to this category. **Business and Organizational Challenges** arise from security vulnerabilities of too invasive automation solutions, required participation of data source owners and low return on investment. [11] supplements the list by unclear responsibilities for connected data sources as well as content and quality assurance of automated updates. Finally, the lack of tool support for automation, synchronization of changes between source an target systems and the appropriate data granularity level are grouped by the category **Tooling** challenges [15].

Resulting to a systematic literature review, Farwick et al. did not only supplement the list by additional challenges but also found broad support in other literature for the items identified by Hauder et al. [11].

2.2. Trends and technologies impacting EAM

In this section important trends and technologies that recently impact the EAM discipline for the better or worse are defined and shortly introduced as these terms often reoccur throughout this thesis.

As a response to the increased pressure to adapt to rapidly changing business requirements, agile software development methodologies have emerged. Along with this, the growing adoption of microservice architectures (MSA), continuous integration and deployment (CI/CD) pose new challenges to EAM and EAMM. Combined with accelerated development cycles driven by agile development methodologies and continuous deployment this trend lead to a growing speed of architectural changes, increased volatility of deployed services and a spread in technological diversity [3]. While back in 2013 [33] already found that practitioners have difficulties to keep pace with architectural changes, the situation has become even more severe. However, these trends also bring forth new valuable information sources that could be exploited for the benefit of EA. Important examples considered in this section are

1. Continuous Deployment (CD)
2. Application Performance Management (APM)
3. API Gateways
4. Services Meshes

2.2.1. Agile Development Methodologies

The term *Agile* in context of software development is considered to be introduced by the *Agile Manifesto* written in 2001 following to a number of early frameworks that pioneered the way to agile concepts [24]. The Agile Manifesto defines *Agile* as a set of core values and principles. Agile is considered a mindset that wraps around these core values and principles but also as a methodology that includes certain engineering practices, tools and processes [24]. The key differentiating characteristic from classic software development approaches is an iterative and incremental development approach that allows for more flexible reaction to changing requirements. Agile methodologies are based on more lightweight processes and are less plan-driven while teams are self-organizing. Development cycles are considerable shorter and a functional software product should be delivered at the end of any iteration. According to the *Annual State of Agile* Report, *Scrum*, *SCRUM/Extreme programming (XP) hybrids* and *SCRUMBAN* count to the most popular and widely adopted agile frameworks [44]. Most employed agile engineering practices include *Continuous Integration*, *Continuous Delivery* and *Continuous Deployment* [44]. These practices are key enablers of the suggested solution approach and are explained in more detail later in this section. The following items (Top5) are the key reasons of applying agile development as found by the *Annual State of Agile* Report [44]:

- Accelerate software delivery
- Enhance ability to manage changing priorities
- Increase productivity
- Increase business/IT alignment
- Enhance software quality

Scrum

Scrum is the most widely-practiced agile methodology (72 percent of respondents as per 13th Annual State of Agile Report [44]). It denotes a framework "within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value" [35]. As part of this framework certain processes, techniques and artifacts are described that should allow for agile software development. The most central artifact is the so called *Sprint*, "a time-box of one month or less during which a “Done”, usable, and potentially releasable product Increment is created" [35]. All other Scrum events and artifacts are organized around sprints. The Scrum team consists of the *Product Owner*, the *Agile Master* and the *Development Team*. The Product Owner’s responsibility is to maximize the value of a product being developed by specifying and prioritizing the products requirements which are captured in *Backlog Items*. During *Sprint Planning* the Scrum team selects and defines what Backlog Items should be realized as part of a Sprint. This subset of items is called the *Sprint Backlog*. The Development Team is a self-organizing, cross-functional team of professionals, responsible to implement and deliver the product increment (the "Done") at the end of the Sprint. The specification in the *Definition of Done* ensures that the entire team is aligned on what "done" actually means. The Scrum Master has no active role in implementing the product but helps everyone in- and outside the team to understand the Scrum methodology and to remove *Impediments* which hinder the team's progress [35]. The aforementioned only covers the most important aspects the the Scrum methodology that are to be found again in the course of this work.

DevOps

DevOps is composed of the terms *Development* and *Operations*. DevOps follows the agile mindset and offers process frameworks and tools that strive to tear down organizational silos of development and operation. In classical environments development and operation are often separated by organizational units as well as by the tools and processes they use [40], [43]. The goal of DevOps is "to integrate all the phases of the application life cycle, and ensure they function as a cohesive unit" as well as to increase the degree of process automation in software development. Typical best practices for instance are Build Automation and Continuous Delivery [40]. To achieve this, DevOps follows a set of software design principles that lead to a increasing usage of Microservice Architectures [40]. According to 13th Annual State of Agile Report found that 72 percent of respondents already have a DevOps initiative running or are planning to do so [44].

2.2.2. Continuous Deployment

Continuous Deployment (CD) denotes the process of automating the software deployment process including *Continuous Integration* and *Continuous Delivery* [31]. The first part of the automation process, Continuous Integration, is "about ensuring your software is in a deployable state at all times. That is, the code compiles and the quality of the code can be assumed to be of reasonably good quality" [31]. Once this precondition is fulfilled, Continuous Delivery automates the rest of the process of deploying the produced software artifact to its operating platform [31]. In his master thesis [45] laid the foundation how this technology can be used for the benefit of EA documentation. It is the key enabler of the solution approach discussed in this thesis.

2.2.3. Microservice architectures and related technologies

Microservice Architecture (MSA) basically means the decomposition of an application into a set of independent, loosely coupled components which are built around business capabilities [3]. Microservices are small, autonomous and distributed services that each run their own processes and work together via the network by providing/consuming API resources. This architectural style is often seen as a more fine-grained service-oriented architecture [3]. The services' functionalities are leaned against business boundaries following the *Single Responsibility Principle* [26]. Microservices characterize themselves by being independent, horizontally scale-able, easy deploy-able, highly flexible and more resilient. However, they are also considered to be more complex than classic monolithic architectures due to a high distribution and increased technological heterogeneity [3], [13]. The adoption of this architectural style often relates to an increased need for agile development practices and a DevOps culture [3]. Impacts from an EAM perspective are the accelerated speed of architectural changes in a more-fine grained IT landscape with a higher level of interaction and increased technological diversity that quickly becomes unmanageable. EAM has therefore to address the "difficulty of keeping a healthy amount of governance and standardization while still allowing enough technological heterogeneity to not hinder innovation and agility" [3].

Application Performance Management

The term Application Performance Management (APM) bundles a set of technologies and skills that allow to monitor the runtime behavior of deployed components, visualize an applications topology, profile transactions as they pass through the system as well as reporting and analytic capabilities [17], [27]. These technologies have grown in maturity over recent years. Distributed tracing technologies enable an in-depth end-to-end analysis of communication flows within and across microservice applications. Aggregating such fine grained information to a higher level can be beneficial to EAM in order to understand the topology of IT landscapes in their entirety [20]. *MICROLYZE*, as presented in [21], already delivered initial prove about the value of the open distributed tracing technology [1] for EA documentation goals.

API Gateways

In the context of microservice and service-oriented architectures (SOA), API Gateways play a central role in the overall architecture. It constitutes the single entry point for incoming requests to a specific service and, thus, decouples clients from distributed and independent microservices. In this key position, it abstracts common functionality needed by microservices such as load balancing, authentication and authorization, caching and monitoring [46]. API gateways also help to overcome the challenges of deviation between the fine-grained APIs provided by microservices and the more sophisticated ones used by clients. To serve clients with different needs, API gateways allow to bundle sets of APIs and publish them to a dedicated set of clients [25]. Examples for API gateways are OpenAPI [2], ApiGee [3] or AWS' API Gateway [4]. In this thesis, ApiGee is analyzed and used as one of the EA information sources.

[1]https://opentracing.io
[2]https://www.openapis.org/
[3]https://apigee.com/
[4]https://aws.amazon.com/de/api-gateway/

Service Meshes

Similar to API gateways, service meshes govern the communication between services in a network. It "is a configurable, low-latency infrastructure layer designed to handle a high volume of network-based interprocess communication among application infrastructure services using application programming interfaces (APIs)" [36]. A differentiating feature of service meshes is "that it takes the logic governing service-to-service communication out of individual services and abstracts it to a layer of infrastructure" [29]. This is technically based on deploying lightweight proxies along with instances of a microservice whereas these proxies span up the service mesh. Based on that, service meshes provide common functionality such as discovery, observability, load balancing and security features [36]. With regards to EA documentation purposes, this makes them as interesting as distributed tracing. For instance, Istio [5] provides graphs of microservice application topology over REST APIs that could easily be integrated as an EAM information source.

2.3. ArchiMate

In the course of this work, numerous enterprise architecture models are included for visualization and better comprehensibility. Due to this, the most important aspects and basics of the ArchiMate notation language are summarized in this section. ArchiMate is an open and commonly used Enterprise Architecture modeling language that is published and maintained by the Open Group [6]. It contains a "set of entities and relationships with their corresponding iconography for the representation of Architecture Descriptions" that should serve to visually describe, analyze and communicate typical concerns of Enterprise Architecture [2].

The ArchiMate Core Framework slices the EA into three horizontal layers and three vertical aspects as depicted in 2.1:

- **Architectural Layers**
  - The **Business Layer** of an EA "depicts business services offered to customers, which are realized in the organization by business processes performed by business actors."
  - The **Application Layer** of an EA "depicts application services that support the business, and the applications that realize them."

[5]https://istio.io/
[6]https://www.opengroup.org/

  - The **Technology Layer** of an EA depicts technology services needed to run applications. These services are realized by networks, hard- and software as well as the underlying physical infrastructure.

- **Architectural Aspects**
  - **Active Structure Aspects** of an EA contains acting elements such as acting organizations or applications that expose a certain behavior
  - **Behavior Aspects** of an EA includes behaviors such as processes, functions, events and services exposed by certain acting EA elements
  - **Passive Structure Aspects** of an EA contains passive objects that are target of a behavior exposed by active EA elements. Examples are information and data objects but also physical elements.

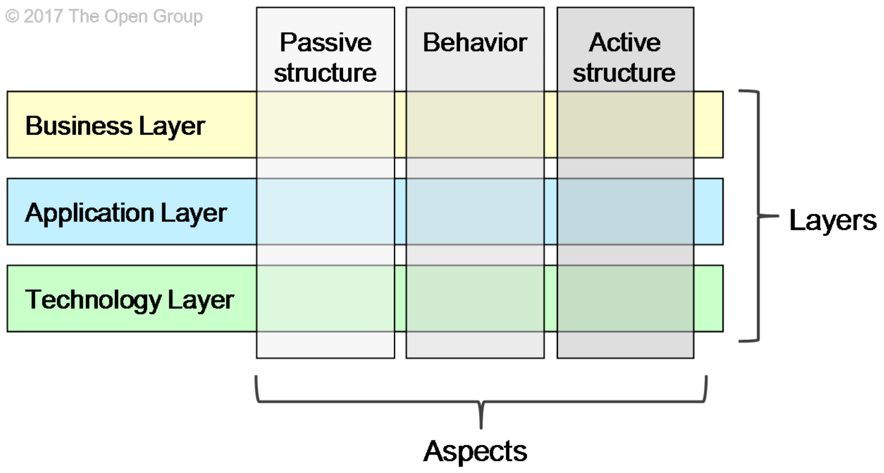


Table 2.1.: ArchiMate core model

This basic structure serves to group the specific EA model elements provided by ArchiMate and define their basic relationships to each other. The details about existing types of relationships are skipped to be described at this point as they are intuitively understandable in the context of an EA model and often are similar to the ones used in UML notation such as assignments, aggregations and specializations. Figure 2.2 contains the definition and visual notation of the essential EA elements used in this master's thesis. All provided definitions follow the official ArchiMate 3.0.1 standard.

Icon	Definition	Classification
Business actor	„A business actor is a business entity that is capable of performing behavior.“	Business Layer, Active Element
Business process	„A business process represents a sequence of business behaviors that achieves a specific outcome such as a defined set of products or business services.“	Business Layer, Behavioral Element
Business function	„A business function is a collection of business behavior […] (typically required business resources and/or competencies) […].“	Business Layer, Behavioral Element
Product	„A product represents a coherent collection of services and/or passive structure elements […]“	Business Layer, Passive Element
Capability	A capability represents an ability that an active structure element, such as an organization, person, or system, possesses.	Strategy Layer, Behavioral Element
Application component	„An application component represents an encapsulation of application functionality […], which is modular and replaceable. It encapsulates its behavior and data, exposes services, and makes them available through interfaces.“	Application Layer, Active Element
Application interface	„An application interface represents a point of access where application services are made available […]“	Application Layer, Active Element
Application service	„An application service represents an explicitly defined exposed application behavior.“	Application Layer, Behavioral Element
Node	„A computational or physical resource that hosts, manipulates, or interacts with other computational or physical resources.“	Technology Layer, Active Element
System software	„A physical IT resource upon which system software and artifacts may be stored or deployed for execution.“	Technology Layer, Active Element
Device	„Software that provides or contributes to an environment for storing, executing, and using software or data deployed within it.“	Technology Layer, Active Element
Artifact	„A piece of data that is used or produced in a software development process, or by deployment and operation of a system.“	Technology Layer, Passive Element

Table 2.2.: ArchiMate EA elements used in this work

3. Related work

3.1. Alternative EAMM solution approaches

This section briefly presents important related work with regards to alternative solution approaches to automate Enterprise Architecture documentation. Not included are rather procedural descriptions with the goal to structure and formalize the EA documentation process as suggested in papers like [8], [11] or [14].

Farwick et al. 2010 [7]: The paper presents a federated EA documentation approach. A central EA model controller retrieves application-related data from cloud platforms and a project portfolio management (PPM) tool and transforms it into an integrated model. Detected elements and occurring changes are pushed to a central EAM repository. The integration of the PPM tool serves the idea to register applications already at planned state (target EA). Once this application is deployed and detected by the system, the state is changed from *planned* to *current*. The reconciliation is based on a GUID that is already assigned to the planned application in the PPM system. This GUID is then added as a tag to the corresponding runtime artifact. This is a first attempt to automate EA documentation across lifecycle phases.

Buschle et al. 2012 [4]: The authors conduct a qualitative assessment of an Enterprise Service Bus which is a central interlinking component of the IT landscape. The goal pursued is to achieve automated EA documentation based on this information source. The results are measured against the entire EA meta model as suggested by ArchiMate. This work revealed that ESBs are a profound source to discover EA elements on the application layer (up to 75 percent) and on the technology layer (up to 50 percent) whereas there are weaknesses on the business layer (up to 20 percent).

Holm et al. 2014 [16]: The authors conduct an assessment of a network scanner. The goal pursued is to automatically generate ArchiMate based EA models from a network scan. The models obtained are very fine-grained and include beside infrastructure elements also logged in users as well as installed system software. The authors admit that the results might be too fine-grained for EA purposes. Also, the approach is restricted to a technology and application layer.

Valja et al. 2015 [41]: The authors present a generic, systematic process of how modeling automation can be achieved. The process is demonstrated using an example which incorporates an active network scanner and a network traffic analyzer as information sources. The data is used to automatically generate CySeMoL models. Reflected against typical enterprise architecture meta models, the approach is restricted to the technology layer.

Trojer et al. 2015 [39]: The paper introduces a novel EAM tool that features different editors for manual EA modeling and as well as a set of interfaces to common data repositories which should facilitate data imports. This includes a CMDB interface, Excel spreadsheet and relational databases. The solution assumes that preexisting architectural data is available for import. It does not discover such data itself.

Johnson et al. 2016 [18]: The authors introduce a machine-learning technique to automate EA modeling. The solution considered EA modeling a probabilistic state estimation problem where EA elements can be identified based on classification. The approach was tested using data produced by a network sniffer. However, the results were limited to two EA elements only, namely hosts and network messages. For all other EA elements, the paper remains theoretical.

Granchelli et al. 2017 [13]: With *MicroART* [13] presented a prototypical implementation of an architecture recovery tool dedicated to microservice-based systems. Required information sources are a source code repository (by providing the repository URL), dynamic queries to the Docker runtime environment and log files about the application components communication. Individual application components are discovered by calling the Docker runtime environment and matching the detected services against Docker-Compose and Dockerfiles found in the source code repository. Based on the assigned IP addresses and network interfaces, MicroART able to recover a communication graph by the help of the provided log files. Finally, the service discovery service is determined and resolved in order to make the direct communication paths between components visible.

Landthaler et al. 2018 [23]: The author suggests the use of machine learning techniques to classify installed executable files and map them to EA model elements. From an EAM perspective, the approach is restricted to the technology layer.

Hacks et al. 2019 [14]: Similar to the EA documentation solution approach at hand, [14] took notice about the potential that resides in continuous deployment technologies for EAM purposes. However, the authors do not use this technology for EA documentation. Instead, they describe how to implement and partly automate an

EA model evolution process from a given preexisting EA model towards a revised EA model. This process ensures the model quality by the help of several quality gates and the calculation of KPIs. Only if all quality gates are passed successful, the new EA model version is released to production.

3.2. Demarcation

The suggested solution approach part of this work differentiates itself in various aspects from preexisting solution approaches. The approach follows a hybrid strategy. In its core, it is based upon the analysis of runtime data which is retrieved from cloud platforms and distributed tracing systems. This data allows to discovery a full inventory of deployed applications and their communication relationships. To make the technical data more valuable from an EAM perspective, it is further enriched by static information that references relationships to the business architecture. For this purpose a configuration file is used which is stored in the source code repository can be extracted during the artifact's deployment process. Moreover, the usage of APIs exposed by other relevant information sources should enable a federated EA documentation strategy. The overall goal is to achieve the discovery of a holistic and multi-layer EA. What all solutions have in common, is the goal of reducing manual modeling efforts by the use of automation techniques. From a technical perspective, the solution in this work is not comparable to the approaches suggested by [23] and [18] as machine-learning techniques do not play a role. Approaches introduced by [16] and [41] both rely on network scanners and/or sniffers. These systems proved to be effective on an application and technology layer but lack to close the gap to the EA's business layer. Another limitations is that the obtained models might be too fine-grained for most EA initiatives. Also, network sniffers might be too intrusive as they require system credentials and could provoke failures. This might be an advantage of the suggested solution. Enterprise service buses were found to be valuable to discover EA elements on application and technology layer by [4]. A precondition is, that a significant part of the IT landscape has to be connected to this technology. It also lacks to provide EA information about the business layer. The EA tool introduced by [39] assumes suitable data repositories that contain preexisting, EA relevant data such as CMDBs, relational databases or spreadsheets. As such sources are often maintained on a manual basis, the approach deviates from the suggested solution approach as it does not discover the EA based on runtime information. The closest relationship exists to the approaches suggested by [7] and [13] as both are using data exposed by operating platforms. Whereas [7] does solely rely on cloud platform APIs, the approach is capable to provide an inventory but lacks to discoverer interrelationships. [13] is able to cover both, application inventory and respective communication flows. However, the solution concentrates on single microservice-based applications, not on an entire IT landscape that might contain numerous of such applications. Similar to the suggested solution approach, [7] tries to establish a more holistic EA model by incorporating the project portfolio management system and the introduction of unique identifiers which are propagated across the systems. This enables simple management of EA elements across the lifecycle stages (PLANNED vs. CURRENT). In this work no system overarching identifiers are introduced, still, it incorporates federated information sources to obtain a more holistic EA model. The only related solution approach that is also based on pipeline integration is presented by [14]. The solution, however, pursues a different goal. It is directed to revise a preexisting EA model to a new version in a controlled manner by the help of a pipeline that leads through several automated and semi-automated stages to ensure model quality. This means that the approach requires EA models as an input but does not itself discover or document EA models.

4. Solution approach

4.1. Introduction to *MICROLYZE*

As part of the "Software Engineering for Business Information Systems" (SEBIS) chair's research in the area of Enterprise Architecture Management, a novel approach to automate EAMM within microservice-based environments was recently presented. *MICROLYZE* denotes a framework for the real-time recovery of microservice architectures across multiple layers of enterprise architecture. This includes the business, application and technology layer as well as inter-dependencies among them. To this goal, several information sources that provide runtime and static information are leveraged herein [21].

MICROLYZE considers the recovery of enterprise architecture a never-ending process. This satisfies the fact that microservice-based architectures evolve over time and underlie constant change. For example, instances are dynamically scaled in or out whereas new services are being deployed and older ones disposed. To keep track of such changes, continuous monitoring and the collection of runtime data constitutes a central element of the discovery process.

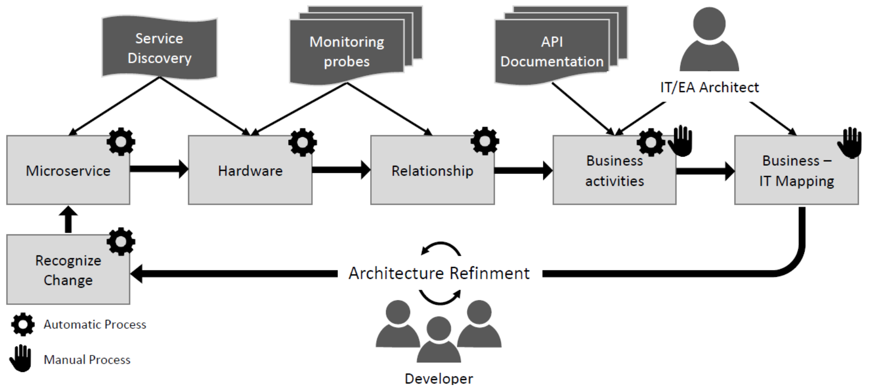


Figure 4.1.: *MICROLYZE* - Microservice architecture discovery process [21]

Figure 4.1 describes the architecture discovery process as a continuous cycle in a total of six phases (gray rectangles). Different information sources are used along this process which data is linked together to stepwise obtain more and more architectural information. The presence of a service discovery service and distributed tracing technologies are key preconditions to make this concept work. Human contribution is needed to some extent to correlate technical events processed by the application with certain activities from a business perspective. The individual phases of the architecture discovery process are shortly characterized as follows:

- **Phase 1 - Microservice:** In this phase, a service discovery service (e.g. Eureka[1] or Consul[2]) is polled to obtain the current inventory of registered microservices and their instances. Besides, data retrieved includes health information, IP addresses, ports, etc.
- **Phase 2 - Hardware:** The second phase serves to obtain the details about the underlying hardware. Based on mapping IP addresses, a relationship between microservice instances and hardware components can be established. Precondition to this phase is that all hardware components have a monitoring agent installed.
- **Phase 3 - Relationship:** In the third phase, distributed tracing data is analyzed to unveil the communications flows of incoming requests and to discover the inter-relationships between individual microservices. For this purpose, microservices need to be instrumented with a monitoring probe which supports distributed tracing[3].
- **Phase 4 - Business activities:** In this phase, business activities supported by the microservices are defined with a clear semantic description using a *business process modeller* featured by *MICROLYZE*. This is a manual process.
- **Phase 5 - Business-IT Mapping:** This phase makes use of the regular expression language in order to describe incoming technical requests which are streamed and stored to a database as they are issued to the application. These technical requests are mapped to the business activities defined in the previous phase.
- **Phase 6 - Recognize changes:** The last phase serves to detect and react to changes to the microservice landscape. This is achieved by regularly polling the service discovery service. Besides, unknown requests are stored for later a later mapping by the use of the *business process modeller*

[1]https://github.com/Netflix/eureka
[2]https://www.consul.io/
[3]http://opentracing.io/

A prototypical implementation applied to discover the microservice-based system TUM-LLCM[4] showed very promising results about the concept's capabilities. The evaluation environment consisted of nine microservices, operated with one instance each distributed over three virtual machines. *MICROLYZE* was able to identify all microservices, all running instances, the underlying operating machines as well as the interconnections and communication paths among them. Using the *business process modeller* and the business activity mapper (featured by *MICROLYZE*) it could be shown that the mapping between technical requests and business activities is feasible. This way, the gap between business and application layer could be closed [21]. A notable limitation of the solution approach is that *MICROLYZE* can only become fully operational in case distributed tracing complying with the open tracing standard is supported and a service discovery service or similar is in use that registers all microservices in place [21].

From that point in time, *MICROLYZE* has been further developed. The *Enterprise Architecture Discovery (EAD)* approach evolved the initial concept by adding a decentral component to the solution: A pipeline-driven, federated EA documentation approach which makes primarily use of continuous deployment (CD) technologies and cloud platform APIs in order to become operational [20]. As in typical cloud environments and agile development practices both, CD and cloud APIs, are present per default, this might be more lightweight to adopt. The initial concept behind *EAD* is presented in [20]. *MICROLYZE* and *EAD* are supposed to complement each other. Figure 4.2 depicts the interplay of the evolved solution concept.

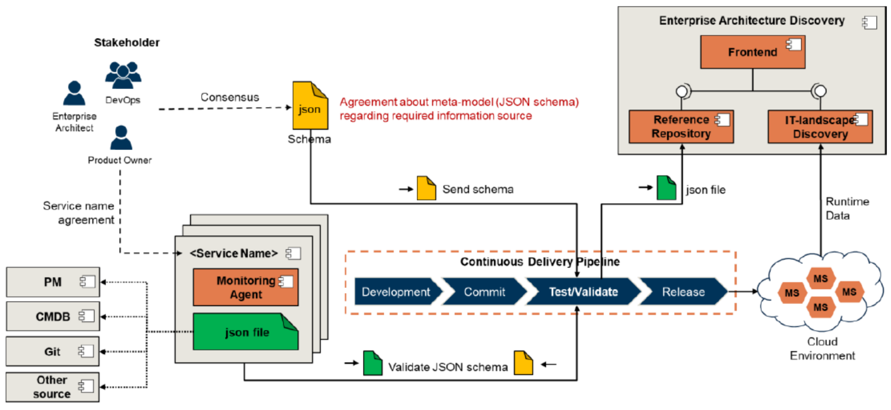


Figure 4.2.: Key building blocks of the EAD concept [20]

[4]http://tum-llcm.de/

A central idea to this concept is to enrich runtime data retrieved from cloud platforms with static information that is contained in a configuration file stored in artifacts' source code repositories. The file is meant to contain information about the business context of a microservice and contextual data such as a human-readable naming and its purpose. In addition, the file should deliver links to federated information systems. Using these references, it is possible to automatically obtain further architectural information from e.g. Configuration Management Databases (CMDB), Project Portfolio Management (PPM) systems, Version Control Systems (VCS) or any desired other information source. The definition of a JSON schema allows to validate the provided configuration file against requirements mutually agreed by key stakeholders. The required code logic is added to the continuous deployment pipelines which serves as a vehicle to extract and validate the configuration file's content and combine it with runtime data retrieved from cloud platforms. Based on this process, a multi-layer enterprise architecture documentation is enabled. An early-stage prototype was implemented from this concept as part of previous work which demonstrated that this is actually feasible [45].

This work is going to further enhance the concept and the prototypical implementation of *EAD* with the main purpose of evaluating it under real-case conditions in an enterprise context.

4.2. Solution architecture

This chapter gives an overview of the solution's architecture and introduces its functional components. Moreover, this section describes the most important procedures of how automated EAMM is achieved. It also proposes an approach on how the artifact can be integrated by agile development teams.

4.2.1. *EAD* functional components

The solution consists of five functional components that work together to achieve automated EA documentation.

EAD-tool: The *EAD-tool* is a central component whose foundations were lied by [45]. It consists of three components which are (1) a database, (2) a backend component providing a REST-based CRUD (Create, Read, Update, Delete) interface and (3) a frontend component. The *ead-frontend* allows to browse and visualize discovered EA elements that are stored in the solution's database. The *ead-backend* allows to store EA element information discovered and provided by the decentral components. It receives and processes data provided by the *ead-library* and the *ead-crawler* which contain the actual EA discovery logic. The backend component also features required functionality for EA model transformation before pushing data to an EAM repository. Together, the *EAD-tool* makes a middleware between the decentral components *ead-library* and *ead-crawler* and a central EAM repository. Both, frontend and backend, are prototyped in Java using the Spring Boot framework [5].

EAD-library: The *ead-library* provides the core functionality for EA documentation. To fulfill this goal, the configuration file (which contains the business context of the application) is extracted from the VCS and combined with runtime data retrieved from a given cloud platform. Further information sources can be integrated as to business needs. The library is meant to be imported and executed from within deployment pipelines. The details about the functionality contained are described in section 4.3.2. This component is implemented as a Jenkins Shared Library[6] using the language Groovy[7].

EAD-crawler: The *ead-crawler* provides functionality for the discovery of deployed microservices on a target cloud platform. It automatically polls the cloud platform for its application inventory and thus, allows to regularly update data about microservices already registered with the *EAD-tool* but also to identify newly deployed microservices. This component primarily serves to solve EA model maintenance tasks. The current prototype is also implemented using Groovy. More details are provided in section 4.3.2.

Additionally consumed APIs denoted with *further EA sources* represent the idea to enable a federated EA strategy by integrating additional information systems that contain EA relevant data. Depending on the type of source such integration could either be added to the *ead-library* or *ead-backend*. Figure 4.3 depicts the described components and their interplay as an UML component diagram.

[5]https://spring.io/projects/spring-boot
[6]https://jenkins.io/doc/book/pipeline/shared-libraries/
[7]https://groovy-lang.org/

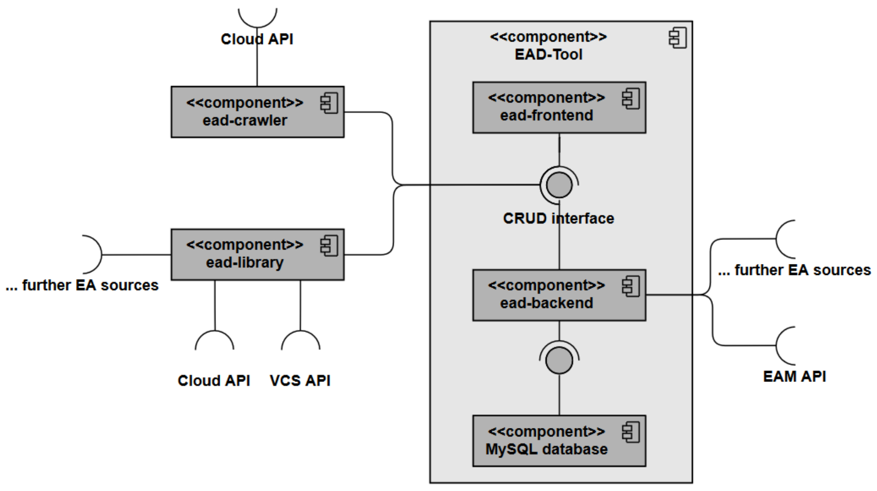

Cloud API
<<component>> ead-crawler
<<component>> EAD-Tool
<<component>> ead-frontend
CRUD interface
<<component>> ead-library
... further EA sources
<<component>> ead-backend
... further EA sources
Cloud API
VCS API
EAM API
<<component>> MySQL database

Figure 4.3.: *EAD* component diagram

4.2.2. Preconditions & required inputs

In order to become operational, agile teams need to meet two preconditions upfront:

1. Integration of a configuration file (called *ead.json*) into artifacts' source code repositories
2. Extension of artifacts' deployment pipelines with EA documentation logic provided by *ead-library*

1. Integration of a configuration file into artifacts' source code repositories

Agile teams need to fill a configuration file following a predefined structure and store it into the artifact's source code repository. This need to be done for each individual artifact. The integration of a configuration file serves two purposes:

- **Contextual information**: The configuration is meant to contain contextual information about a software artifact which can later be linked to the runtime artifact resulting from the deployment process. This serves the goal to make runtime data more valuable and easier to analyze from an EAM perspective. For instance, such contextual information is meant to include business layer assignments (e.g. business domain assignments, supported processes and business capabilities) and the artifact's belonging to a super-ordinate application.

- **References to federated EA information sources**: To enable a federated EA documentation strategy another key purpose of the configuration file is to provide references to the artifact's representation in federated systems. For instance, existing entries in Project Portfolio Management tools, monitoring tools or version control systems. Theses references allow enriching the EA meta model by further information residing in these sources.

The configuration file is specified in JSON[8] format. For the solution to become operational, the configuration file should at least specify the following fields. Fields indicated as *mandatory* are required for the proper functionality of the system. *<federated_system_name>* can be specified according to business needs.

```
{
  "application_component_name": "human-understandable name of the application component (mandatory)",
  "description": "short description of the application component's purpose (optional)",
  "superordinate_application": "name of the top level application the application component belongs to (mandatory)",
  "product_owner": "email of the product owner (optional)",
  "business_domain": "the business domain the application component belongs to (optional)",
  "<federated_system_name>":"URL to or ID of the corresponding element in the referenced system (optional)"
}
```

2. Extension of deployment pipelines by EA documentation logic

Agile teams need to adapt their deployment pipelines to include the EA documentation logic provided by the *ead-library*. This serves the main purpose of extracting the *ead.json* file from the source code repository, validate it for correctness/completeness and finally link the information contained to the runtime artifact that results from the deployment process. The pipeline-integration also enables a deployment-triggered EA documentation and automated updates along with each execution of the pipeline script. In an agile development environment, this means that the EA model is updated at least once per sprint and product increment release. Subsequent changes to the content of the *ead.json* files are therefore also captured and updated accordingly.

The approach on how these two precondition can be integrated into the daily business of agile development is described in detail in section 4.3.

[8]https://www.json.org/

4.2.3. Approach for automated pipeline-driven EA documentation

Figure 4.4 puts the aforementioned functional components and preconditions into the overall context of a DevOps cycle which is depicted as a process at the center of the illustration. There is a split of responsibilities between agile teams and the EAM team. The EAM team is responsible for the EA documentation logic contained in the *ead-library*. It needs to be centrally provided to the agile teams. This can be achieved using a Version Control System (VCS). Agile teams are responsible to create the two preconditions highlighted in red color (1) *integrate config file* and (2) *Pipeline integration*. (1) consists of filling the *ead.json* configuration file according to predefined requirements specified in a schema provided by the EAM team and its integration into the source code repository. (2) requires the team to adapt their deployment pipeline scripts to incorporate the EA documentation logic provided by the *ead-library*. The result of this step is an additional stage in the pipeline following the artifact's deployment to a cloud platform (called *document EA* in figure 4.4). The stage combines the static data contained in the provided configuration file with runtime data retrieved from the cloud platform. Based on this information an EA model of the artifact is constructed and posted to the *ead-backend*. Once the preconditions are established, the EA documentation becomes a natural part of the agile development process. It is executed along with each run of the deployment pipeline without any further manual intervention. In an iterative and incremental software development process the EA model is therefore updated at least with each product increment release. Nevertheless, it is necessary to regularly retrieve status information from the cloud platform as there can be change events (such as scaling, deletion, renaming, migrations, etc.) that also impact the EA model. This functionality is provided by the *ead-crawler* which main purpose is to detect changes to the current microservice landscape. Finally, the *ead-frontend* allows to browse and visualize the results of the EA documentation procedure. Detailed procedural descriptions are contained in 4.4.3 for the pipeline integration, 4.3.1 for integration into agile development processes and 4.3.2 for EA model maintenance.

4.3. Processes and procedures

This section introduces and explains the most important processes and procedures for the solution. The first subsection suggests an approach of how the solution can be integrated as part of agile development processes. The second subsection explains how automated EA model maintenance is enabled by the suggested solution. The last subsection introduces a validation process whose goal is to ensure the data quality and correctness of *ead.json* files provided which is a key success factor to the solution.

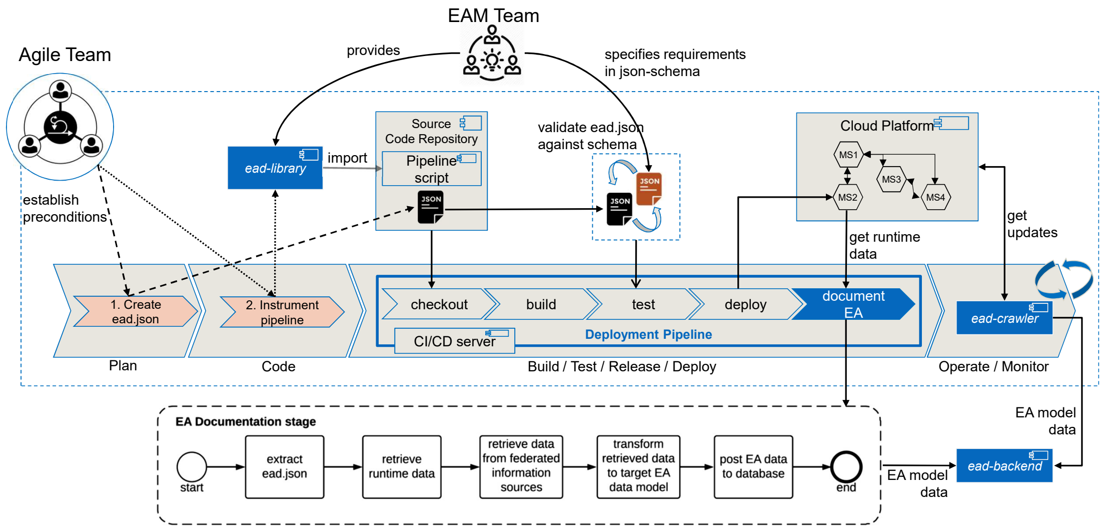


Figure 4.4.: Solution concept overview

4.3.1. Integration into agile development process

In order to ensure a high acceptance of the solution in agile teams, it is important to think about a smooth integration into their daily business and used processes. This section proposes an approach on how to achieve this in the context of the agile development methodology Scrum. To make the solution work, agile teams are responsible for (1) the initial adoption of the solution consisting of equipping artifacts with the *ead.json* file and integrating the *ead-library* into deployment pipelines (onetime activity) and (2) to keep the *ead.json* file up-to-date. This, like all manual documentation efforts, poses a certain risk for outdated information whereas the pipeline integration is a one-time effort.

Figure 4.5 (modeled in BPMN 2.0 notation[9]) describes the process of how the solution can be rolled out using regular Scrum processes. In essence, the solution's adoption should be handled just as any other requirement. First, Product Owners have to create the *ead.json* and pre-fill it with the required business layer assignment valid for the artifact (e.g. domain and sub-domain, business units, supported processes, etc.). Afterwards, a user story is created that contains the requirements for the solution's integration. For further facilitation and reduction of errors, the EAM team should provide product owners with a template. The user story is added to the Product Backlog and at some point selected for execution as part of a Sprint Backlog. Depending on

[9]https://www.omg.org/bpmn/

how fast the roll-out should proceed, the backlog items priority may vary according to the organization's needs. During the Sprint, the development team needs to complete the pre-filled *ead.json* file by references to defined federated information systems and finally to commit it to the source code repository. Besides, they need to integrate the *ead-library* into the deployment pipelines as described in section 4.4.3. This completes the user story. The EA documentation itself is automatically executed with each run of the deployment pipeline. There is no further need for manual intervention. In figure 4.5 this is exemplarily shown following to the Sprint Review. Scrum, however, does not set a fixed time for deployments. There can be multiple during a sprint or even be skipped.

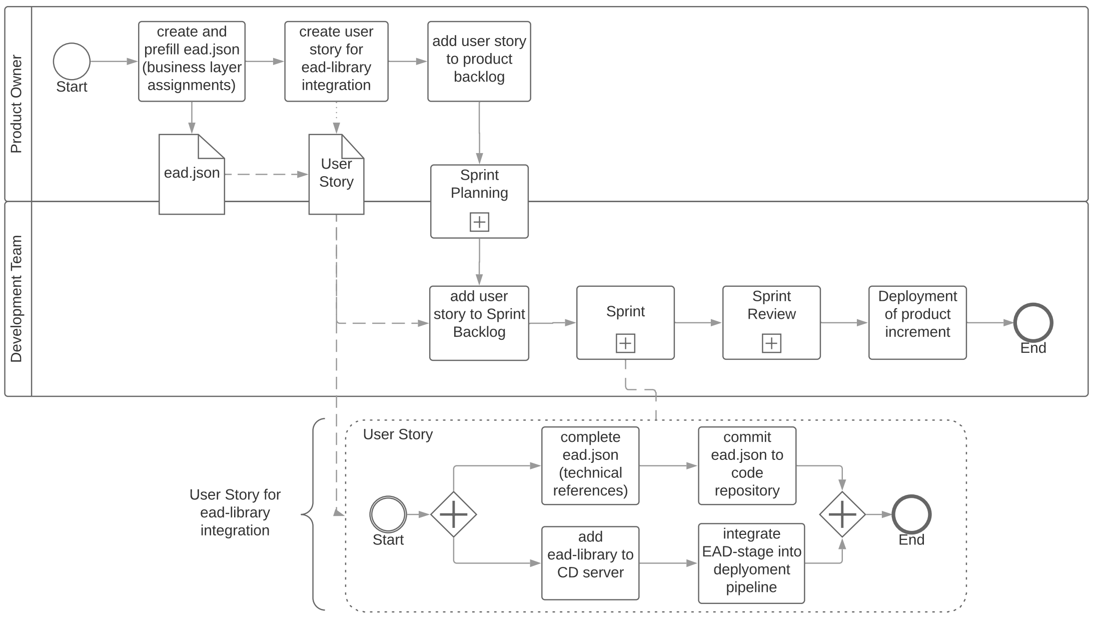

Product Owner
Development Team
Start
create and prefill ead.json (business layer assignments)
create user story for ead-library integration
add user story to product backlog
ead.json
User Story
Sprint Planning
add user story to Sprint Backlog
Sprint
Sprint Review
Deployment of product increment
End
User Story for ead-library integration
User Story
Start
complete ead.json (technical references)
commit ead.json to code repository
add ead-library to CD server
integrate EAD-stage into deplyoment pipeline
End

Figure 4.5.: Integration of the EAD-library using agile development processes

Once the solution is integrated, it has to be ensured that the information contained in the *ead.json* is kept up-to-date. A preventive measure to ensure this is to limit the content to static or almost static attributes. Nevertheless, it cannot be excluded that there may be changes in future. To cover this risk it is suggested to make an addition to the *Definition of "Done"* (DoD) for a product increment. The DoD is a Scrum artifact that ensures a common understanding of what "Done" means and what criteria have to be fulfilled [35]. By including the review of the *ead.json* as one of these criteria it is possible to ensure that the information is maintained before any product increment release. Section 4.3.3 gives further advice how data quality assurance with *ead.json* files can be achieved.

4.3.2. EA model maintenance process

The solution supports automated EA model maintenance by a close collaboration of the *ead-library* and *ead-crawler*. The integration of the *ead-library* ensures that at least with any execution of a deployment process, EA data is automatically updated (in case of changes to the *ead.json* file or any of the information sources incorporated in the EA documentation logic). This ensures a minimum update frequency at least once per sprint release in agile development.

However, there might be changes that are directly invoked from the cloud platform after an application's deployment that cannot be captured by the *ead-library*. Thinking of simple scenarios like application deletion, application renaming, changes of assigned URLs or changes in services used are only a few examples of such changes. Therefore, it is necessary to have a functionality that covers change events that occur after the deployment during runtime. Another scenario that needs to be covered are artifacts that are deployed to production without having integrated the *ead-library*. This is especially true applications which exist prior to the roll-out of the solution or if there is no possibility to technically enforce the integration of the *ead-library*.

The *ead-crawler* is executed fully independent from the *ead-library* with the goal to detect deployed apps and report about their current state. This allows updating the EA documentation about already registered apps and create new entries for newly discovered apps. In case agile teams did properly integrate the *ead-library* prior to the first deployment, apps are usually known and at least the *last-seen* timestamp is updated after each crawling process. In case an app is detected but unknown in existing documentation, the *ead-crawler* is capable of creating new entries with limited EA information as the *ead.json* is not provided. This scenario also implies that the responsible agile team's pipeline is not equipped with the *ead-library*. To sort this out, there can be notifications to the central EA team or directly to the respective contacts which are registered with the cloud platform. Especially during the roll-out phase this is a valuable vehicle to track the process and inform teams about open tasks. The *last-seen* timestamp serves to judge whether an app is still alive or not. For instance, deleted apps won't be registered any more during crawling processes. Also, no new deployments might come in. After a certain threshold since the time an app was last seen, the respective EA documentation can be handled accordingly and either set to an obsolete or inactive state or even just deleted.

Figure 4.6 depicts the EA model maintenance procedure in BPMN 2.0 notation.

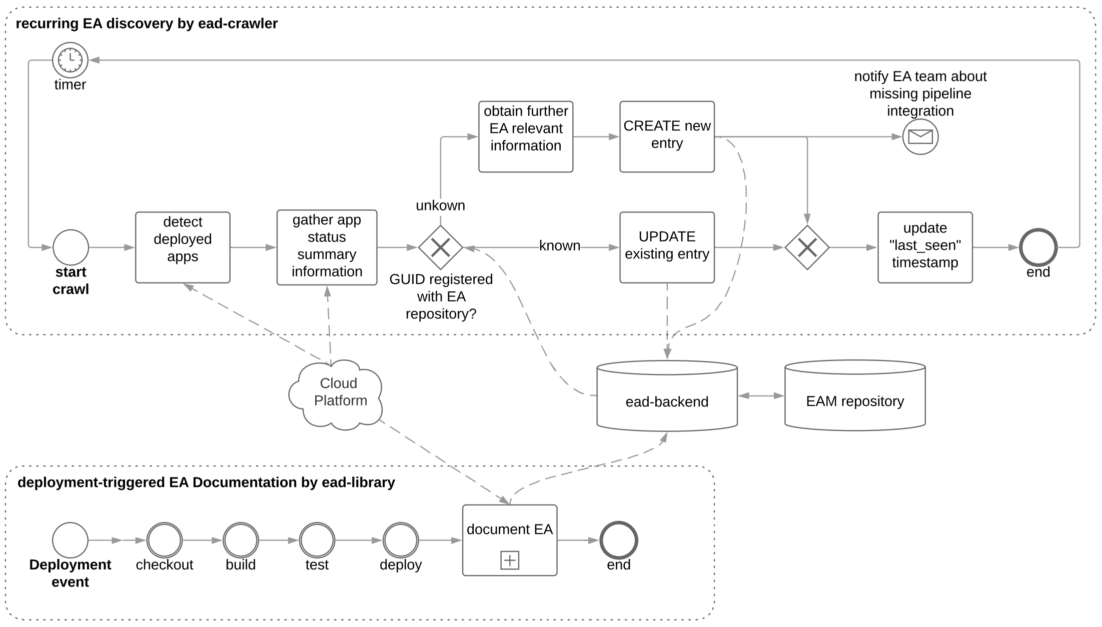

recurring EA discovery by ead-crawler
timer
notify EA team about missing pipeline integration
obtain further EA relevant information
CREATE new entry
unkown
start crawl
detect deployed apps
gather app status summary information
GUID registered with EA repository?
known
UPDATE existing entry
update "last_seen" timestamp
end
Cloud Platform
ead-backend
EAM repository
deployment-triggered EA Documentation by ead-library
Deployment event
checkout
build
test
deploy
document EA
end

Figure 4.6.: Continuous EA model maintenance process

4.3.3. Ead.json validation

As *ead.json* files have to be created manually, there is a risk of errors and incorrectly specified attributes. Therefore, quality assurance measures are required to ensure the proper functionality of the EA documentation process and the correctness of its results. There are two possibilities to achieve this. Either (1) have *ead.json* files generated by the help of interactive forms and/or (2) validate the file during the deployment process against a predefined json-schema[10]. These options can be combined to obtain the best results.

Form-support can be achieved in multiple ways. One option is to provide a simple web-based form that actively supports the user to choose the correct values and providing the generated *ead.json* as a download. Important is, that the option provided by the form (e.g. drop-down or multiple-choice fields) are always up-to-date. Ideally, the from is connected to a repository where the values are stored (e.g. enterprise domain models are usually stored in the EAM repository). Another way is to integrate the form into software project initialization tools such as the *Spring Initializr* [11]. This

[10]https://json-schema.org/
[11]https://start.spring.io/

tool supports developers setting up and initialize a new source code project. The first option suits best for preexisting projects which have to be equipped with the *ead.json* file subsequently. The second option is the better choice for new software projects. Both options ensure that the generated JSON file follows the schema and structure as expected by the *ead-library* and therefore reduces the risk for errors.

However, forms cannot guarantee the content is correct. Also, the links to federated information sources might be specified incorrectly. To tackle this issue, the concept includes a validation process as part of the overall EA documentation stage in deployment pipelines. The entire EAD pipeline stage, drilled down to the validation step is depicted in figure 4.7 outlined in BPMN 2.0 notation. The *ead.json* validation makes the first step in the EA documentation stage that starts with a method call to *ead()*. The *ead.json* located in the source code repository is read from the VCS and checked against a predefined json-schema. In case the file is not available at all or violates the schema, the validation fails and the EAD process is stopped with respective feedback to the user. *Ead.json* files that pass the schema validation run through further content quality checks. In the first place, business layer assignments (i.e. domains, sub-domains, supported business units, etc.) are checked for existence. At this place, it is assumed that this can be validated against a repository that stores the business layer elements (e.g. domains as part of an enterprise domain model or business processes as part of a business process management system). This is an important step as the *ead.json* is not meant to create business layer elements but to link the artifact to existing ones. The process moves on to verify the provided federated system references for correctness and reachability. Correctness in this context means that the paths (URL) are specified as expected by the *ead-library*. For instance, assuming that Jira Components are involved into the documentation process, it must be checked that the path actually leads to a component and not to a backlog item. Reachability means that a ping to the given URL is successful. Given that validation issues occur, they are recorded and provided to the user at the end of the process. This feedback could either be included into the deployment pipeline's logs or sent out as an email notification. A critical validation failure will terminate the EAD stage prematurely. Else, the process will continue. Depending on the organization needs the process can be designed to be blocking or non-blocking. Figure 4.7 depicts a non-blocking process as the validation follows the deployment. In a blocking variant, the validation should precede the deployment stage, so that a successful deployment depends on the success of EA documentation.

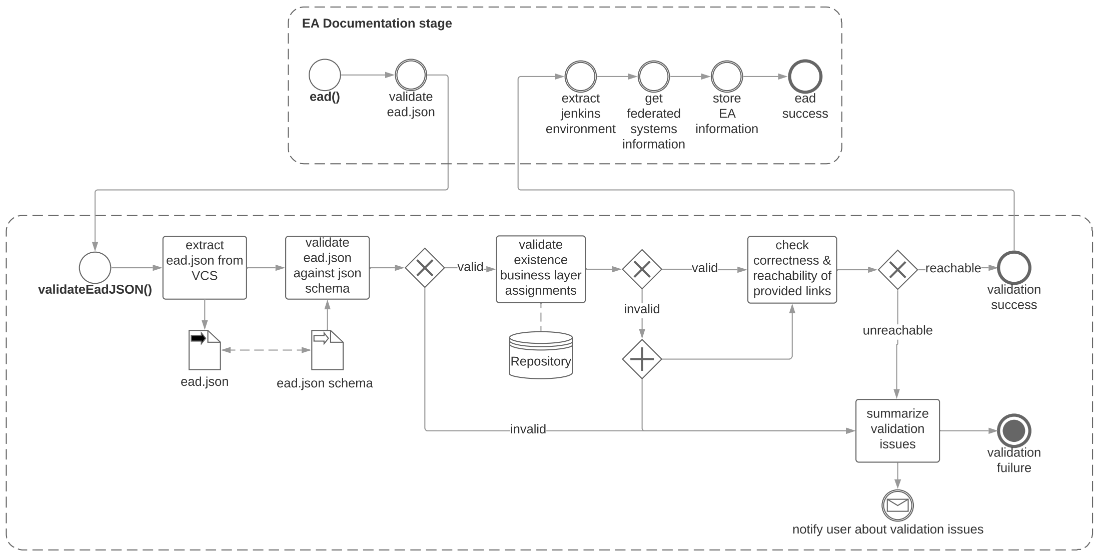

EA Documentation stage
ead()
validate ead.json
extract jenkins environment
get federated systems information
store EA information
ead success
validateEadJSON()
extract ead.json from VCS
validate ead.json against json schema
valid
validate existence business layer assignments
valid
check correctness & reachability of provided links
reachable
validation success
invalid
unreachable
Repository
ead.json
ead.json schema
invalid
summarize validation issues
validation fuilure
notify user about validation issues

Figure 4.7.: EAD pipeline stage and ead.json validation process

4.4. Prototypical implementation

This section describes the solution concept's essential aspects and features that were implemented. As previously mentioned an early stage prototype was developed by previous work [45]. Implementation activities concentrated on making the solution ready for productive use. The *ead-library* is a fully new developed component resulting from this process whereas preexisting components including the *ead-crawler* and *ead-backend* were improved with additional functionality.

4.4.1. Logical data model

Figure 4.8 depicts the logical data model depicted as a UML class diagram. Except for methods for setting and getting attributes, these classes do not expose specific behavior.

The class *Application Component* refers to any microservice discovered and registered by the *ead-library* or *ead-crawler*. It stores basic attributes (e.g. business layer assignments, technical and human-understandable name, descriptive information), technical attributes (e.g. Full Qualified Domain Name (FQDN), last deployment date, used programming languages and deployment location within Cloud Foundry (space and organization)) and functional attributes that help to manage data consistency (cloud platform GUID, EAM repository ID and various timestamps). In this model, business layer assignment including *business domains, business products, business units* and *business functions* are considered attributes, not individual objects as their name is unique. In a usual EAM repository these assignments would of course correspond to individual elements.

Other objects that usually are represented as dedicated elements in EAM repositories are depicted as individual classes. This includes the *Cloud Platform* the application is operated on, *Cloud Services* the application makes use of, the overall *Business Application* the component belongs to and *Interfaces* that are either consumed or exposed by the application component. All of these classes are equipped with unique IDs reflecting their co-existence in an EAM repository. This is a precondition to ensure data consistency when exporting data. *Interfaces* and *Cloud Platforms* are mainly characterized by the URL used to remotely address them. A *Cloud Service* in Cloud Foundry is characterized by its name, service plan and service type (e.g. MySQL database, redis database, autoscaler, monitoring agent). The attribute *name* refers to the name that is also assigned in the cloud platform. The class *Business Application* is meant to aggregate individual components to a logical application level. Usually, this level is also modeled in EAM repositories which is why it is expressed as a dedicated class instead of a simple attribute.

Lastly, the class *References* reflects federated information systems used for EA data collection. It stores the label, i.e. the name of the federated information source (e.g. JIRA, CMDB, etc.) and the URL that directly points to the application component's representation in this federated system.

4.4.2. Prototype improvements

From the point of takeover, the prototype implementation was not ready to be rolled out. It suffered the following deficiencies that blocked the adoption in a real case setting:

1. **ead-script deficiencies:**
    - **Provisioning model:** the pipeline script that contains the documentation logic counts more than 200 lines of code. This makes the integration difficult and prone to errors
    - **Lack of maintainability:** once rolled-out to the teams, it is impossible to centrally maintain or extend the functionality. Each change would require the agile teams to manually integrate the script again
    - **Generality:** the pipeline script is using CLI calls to retrieve information from the cloud platforms. The text-based console responses cannot be interpreted

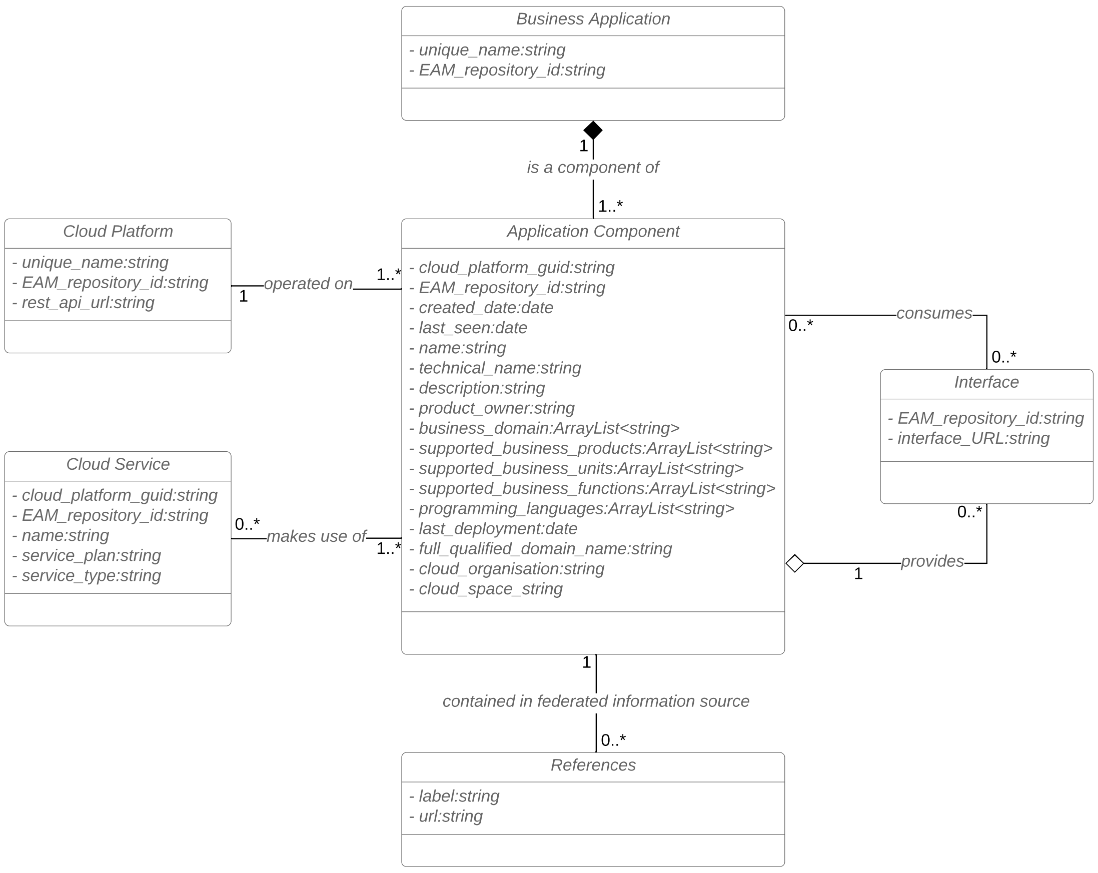

Business Application
- unique_name:string
- EAM_repository_id:string
1
is a component of
1..*
Application Component
- cloud_platform_guid:string
- EAM_repository_id:string
- created_date:date
- last_seen:date
- name:string
- technical_name:string
- description:string
- product_owner:string
- business_domain:ArrayList<string>
- supported_business_products:ArrayList<string>
- supported_business_units:ArrayList<string>
- supported_business_functions:ArrayList<string>
- programming_languages:ArrayList<string>
- last_deployment:date
- full_qualified_domain_name:string
- cloud_organisation:string
- cloud_space_string
Cloud Platform
- unique_name:string
- EAM_repository_id:string
- rest_api_url:string
1
operated on
1..*
Cloud Service
- cloud_platform_guid:string
- EAM_repository_id:string
- name:string
- service_plan:string
- service_type:string
0..*
makes use of
1..*
consumes
0..*
0..*
Interface
- EAM_repository_id:string
- interface_URL:string
0..*
provides
1
1
contained in federated information source
0..*
References
- label:string
- url:string

Figure 4.8.: Logical data model outlined as UML class diagram

right away but have to be segmented and cut into relevant pieces by the help text processing functions. This makes the functionality very prone to errors as well as hard to maintain and extend.

2. **ead-crawler script deficiencies:** The cloud crawler script suffers the same issue that are valid for the *ead-script*. So far, the crawling process only retrieves and updates information about applications that are already registered in the backend. It ignores apps where no documentation is available.

3. **Functional deficiencies:**
   - **Robustness against refactoring:** So far the prototype relies on name matching when it comes to checking discovered applications against already registered applications.
   - **Consistency:** Currently, the backend does not provide functionality to avoid the creation of redundant entries on post actions.
   - **Integration of federated systems:** Connectors to federated systems were not implemented to support basic or token-based authentication.
   - **Detection of communication relationships:** Upon investigation, it was found that the analysis of configured of network policies is not a reliable source to discover communication among applications as applied by the initial prototype. Such policies tend to be not used or too generic. Also, they do not give information about whether there actually is communication traffic for certain.

To overcome these issues one important decision was to use a Jenkins Shared Library[12] as provisioning model for the *ead-script* and the *ead-crawler-script*. The outcome of this implementation is the *ead-library* component that contains both, the logic for EA documentation and EA crawling. The most important benefits of using a library in the context of the suggested solutions are as follows:

- **Central management and maintenance:** a library can be centrally managed by the enterprise architecture department. They can maintain the source code, extend its functionality and provide different versions to cope with different cloud platform technologies in use at an enterprise.

- **Ease of integration:** libraries are easy to import and to integrate into pipelines. The centrally managed code logic is loaded into the CI/CD server instance and executed decentrally. From more than 200 lines of code to integrate, this can now

[12]https://jenkins.io/doc/book/pipeline/shared-libraries/

be achieved with only five lines of code. More detailed information about the integration procedure is contained in section 4.4.3

- **Reduced roll-out and run costs:** Thanks to the benefits mentioned above, the roll-out costs can be reduced. Also, in case of updates, agile teams are not required to integrate the logic anew. The only exception from this is the case of the method's signature (which is called from within deployment pipelines) to change, which would then also require the teams to act.

- **Reuse of existing credentials:** One issue claimed when using network sniffers or agents-based EA discovery methods is, that it requires extensive access rights to the target system that might cause an increase security risks. As libraries code logic is loaded into the pipeline it allows to reuse the credentials that are needed to run the deployment (e.g. SCM credentials to checkout source code, credentials to BRM system and credentials to the target deployment platform). There are no additional access rights required for such systems. However, this requires the library to be trusted by its users.

- **Reuse of code:** libraries are ideal to provide code in a reusable manner. It also allows to modularize the EAD process logic contained and provide tailored sequences according to actual need.

This decision, however, implied to rewrite major parts of the prototype's source code. Along with this, also other issues previously mentioned were handled. This includes the following improvements:

- **Exchange CLI by API calls:** With the goal the make the scripts more robust and prevent labour intensive handling of text-based CLI responses, the *ead-script* and *ead-crawler-script* were rewritten to use API calls. The JSON-based responses also allowed for easier and faster handling of retrieved data.

- **Refactoring robustness:** Name-based matching was replaced by ID-based matching wherever possible

- **Crawling scope:** The *ead-crawler-script* was extended to also document applications not yet registered with the backend instead of ignoring them. As such applications are not equipped with an *ead.json* file the documentation scope is limited, however, gives the opportunity to notify relevant stakeholders about this.

- **Consistency:** To avoid duplicates, the backend's functionality was extended to validate whether a posted element is already registered (based on its GUID) before any *POST* action is executed.

Deficiencies that remained unsolved are:

- **ead-crawler limitations:** For documenting an application along with its deployment it is sufficient to reuse the credentials being provided to Jenkins. For crawling, however, these access rights typically only cover a certain subset of the cloud platform. For this reason, crawling should on the long run be centralized with global reading access to the entire cloud platform. Such access rights were not granted at the industry partner.
- **Detection of communication relationships:** As previously mentioned network policies do not make a reliable source for the discovery of communication relationships between applications. To cover this functionality, the use of distributed tracing is required throughout the entire platform. This precondition is not met at the industry partner.

4.4.3. Pipeline integration procedure

This section describes the concept of how the redesigned prototype can be integrated into the CI/CD pipelines. It is essential to keep this a fast and easy executable task to keep the roll-out costs low and the acceptance by development teams on a high level.

This is achieved by providing the solution in form of a Jenkins Shared Library. To integrate the solution only two steps are necessary:

1. Add the library to the CI/CD server.
2. Import the library into the pipeline script with a pre-configured stage for EAD documentation.

Adding a library to the CI/CD server is a task that can be achieved using the system settings menu. This will make the library publicly available to all jobs on the server. The library can be integrated into the pipeline script as depicted in 4.9. The code snippet has to be pasted into the pipeline script after the deployment stage. The highlighted keywords are parameters that need to be replaced by the developers. Table 4.1 explains the meaning and purpose of these parameters.

This completes the integration the *EAD-library*.

4.4.4. Automated, deployment-driven EA documentation process

This section describes the process of automated EA documentation which functionality is provided by the *ead-library*. Figure 4.10 depicts a sequence diagram of the EA documentation process. The process relies on standard REST APIs provided by the

```
// import the ead−library into the pipeline script
\@Library(['ead−jenkins−library@master']) _

// original pipeline stages (checkout, build, test, deploy)
[...]

// pipeline stage to be added
stage('EAD␣documentation␣process') {

    steps {
        script {

            // read manifest.yml
            def manifest = readFile "${PATH_TO_MANIFEST.YML}"

            // read ead.json to a JSONObject
            def eadjson = readJSON file: "${PATH_TO_EAD.JSON}"

            // method call to the EAD process
            eadprocess.ead(eadjson_file: eadjson, manifest_file: manifest, pcfApiUrl: "${
                CLOUD_PLATFORM_API_URL}", pcfCredentialsID: "${PCF_CREDENTIALS_ID}", org
                : "${PCF_ORG}", space: "${PCF_SPACE}")
        }
    }
}
```

Figure 4.9.: Pipeline integration code snippet

used information sources. This basic example features the API endpoints used for Cloud Foundry (cloud platform) and GitHub (VCS). Functional components provided by the suggested solution concept are highlighted in blue color.

Process trigger: The overall process starts with an event triggering the deployment pipeline. This either can be a human or an automatism, depending on the pipeline's configuration. The solution does not require a certain interval of execution. Agile teams are free to run the deployment pipeline whenever there is a need to.

1) Load library: At the beginning of the pipeline job, the *ead-library's* EA documentation logic is loaded from the VCS system to the CI/CD servers environment.

2) Regular pipeline stages: Following to that the regular pipeline stages are executed. Typically such a pipeline consists of the following stages:

Parameter	Description	Purpose
PATH_TO_MANIFEST.YML	relative path to the manifest.yml config-file	delivers the artefact's technical name name and further configuration data
PATH_TO_EAD.JSON	relative path to the *ead.json* file	stores the *ead.json* file to a JSON object
CLOUD_PLATFORM_API_URL	API URL	provides the cloud platforms API base url
PCF_CREDENTIALS_ID	credentials ID	provides the credentials to be used for calling the cloud platform
PCF_ORG	PCF organization name	target organization in Cloud Foundry
PCF_SPACE	PCF space name	target space in Cloud Foundry

Table 4.1.: Explanation of ead() function parameters

- **2a) checkout source code:** loads the artifact's source code from the VCS system into the CI/CD server's working directory
- **2b) build artifact:** in this step the CI/CD server builds the source code into an executable artifact
- **2c) run tests:** functional tests such as unit tests or integration tests are executed to ensure the built artifact works properly
- **2d) deploy to operating environment:** this stage finally deploys the executable artifact to a defined target operating environment (e.g. cloud platform). The deployment must take place before the EA documentation stage is started.

3) EA documentation stage: The actual EA documentation starts with a method call to *ead()*. The logic for this method is provided by the *ead-library*.

- **3a) *ead.json* validation:** The process starts with validating the *ead.json*. This stage checks for the existence of the file and the correctness of its content. This is important to ensure the success of the remaining process and the quality of EA documentation achieved. The details of this validation process are described in 4.3.3.

- **3b) extract static information:** This step consists of multiple tasks. First, after successful validation, the information provided in the *ead.json* is stored to variables. Secondly, descriptive information about the artifact contained in the *README.MD* file is extracted. And lastly, specific CI/CD server environment variables, including the URL to the source code repository, the URL of the deployment pipeline itself and, if applicable, URLs to the BRM system the built artifact was stored to and the code inspection system that stores the results obtained in the test stage.

- **3c) extract app name from manifest.yml:** This step serves to obtain the app's technical name, i.e. the name that is assigned to the artifact in the cloud platform. In the case of Cloud Foundry, a config file called *manifest.yml* contains this information. For other technologies, possible alternatives are the extraction from *POM.xml* files or other build scripts (e.g. Gradle files). If there is no option for automatic retrieval, the artifact's name has to be provided as part of the *ead()* methods parameters.

- **3d) get app GUID:** Using the app's technical name identified in the previous step, the cloud platform API can be called to retrieve the GUID assigned to the app. This is a necessary step to use further API resources that require the specification of a GUID as a query parameter and to gain robustness against later refactoring

- **3e) get app runtime information:** this step fetches runtime data about the deployed app from the cloud platform. This includes status information, running instances, cloud platform information, etc.

- **3f) get app environment:** The app stores various environmental information including assigned routes (i.e. URL that are externally available to call the app remotely) and used cloud services (databases, auto-scaling services, monitoring agents, etc.). This information is also of interest for EA.

- **3g) get programming languages:** This step serves to retrieve used programming languages the application is implemented with from the VCS's API.

- **3h) post collected EA information to *ead-backend*:** This final step of the EA documentation stage is responsible to store the information gathered throughout the steps 3b) to 3g) to the *ead-backend*. Depending on whether the application is already known (based on matching its GUID), existing entries are updated or new ones generated.

- **3i) forward EA data to EAM repository:** As described in previous sections the *ead-backend* takes over EA model transformation functionality. If applicable, the data is forwarded to and stored by a central EAM repository. The HTTP response object contains the ID of the created or updated object. *ead-backend* stores this ID back to its own database to link it with the app's cloud platform GUID. By the help of this ID a direct link to the EAM repository element can be generated. The link is then pushed back together with additional information about the success or failure of the EA documentation stages to the CI/CD server users. This ends the overall deployment pipeline.

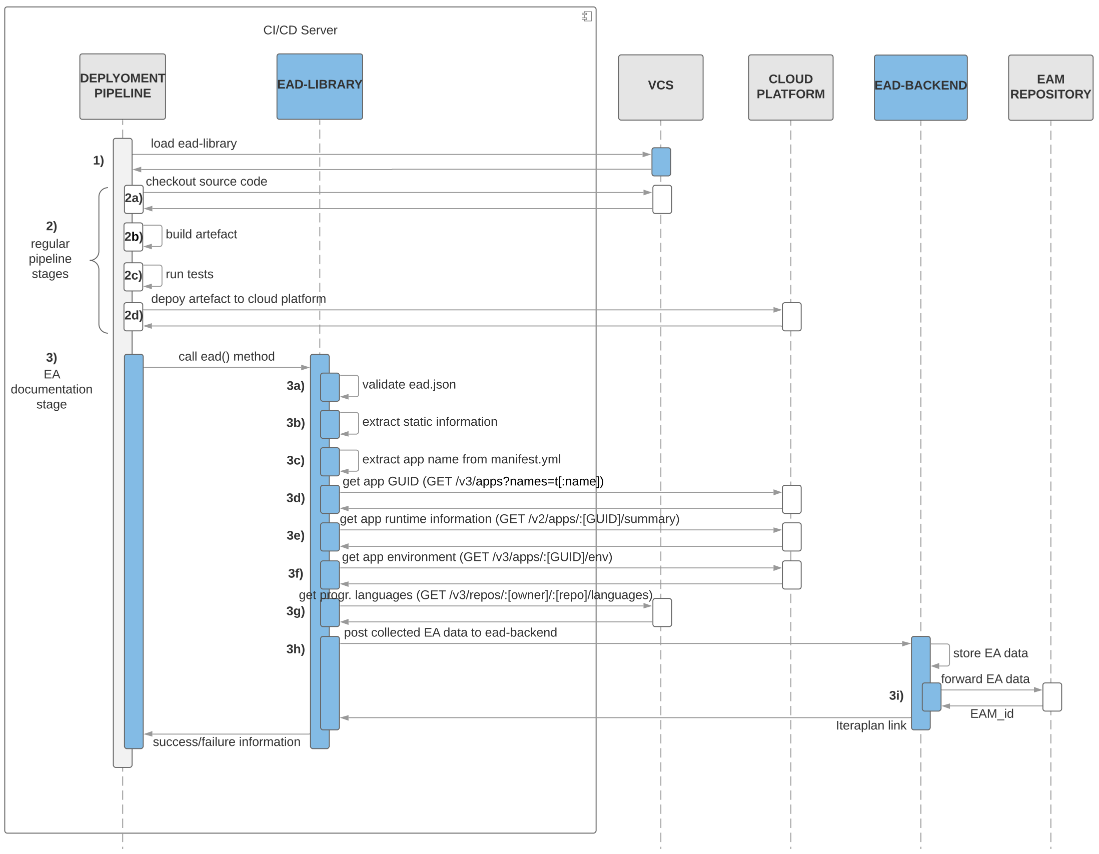

CI/CD Server
DEPLYOMENT PIPELINE
EAD-LIBRARY
VCS
CLOUD PLATFORM
EAD-BACKEND
EAM REPOSITORY
1)
load ead-library
checkout source code
2a)
2) regular pipeline stages
2b)
build artefact
2c)
run tests
2d)
depoy artefact to cloud platform
3) EA documentation stage
call ead() method
3a)
validate ead.json
3b)
extract static information
3c)
extract app name from manifest.yml
3d)
get app GUID (GET /v3/apps?names=t[:name])
3e)
get app runtime information (GET /v2/apps/:[GUID]/summary)
3f)
get app environment (GET /v3/apps/:[GUID]/env)
3g)
get progr. languages (GET /v3/repos/:[owner]/:[repo]/languages)
3h)
post collected EA data to ead-backend
store EA data
forward EA data
3i)
EAM_id
Iteraplan link
success/failure information

Figure 4.10.: Ead-library - Process sequence diagram

5. Evaluation design

This chapter describes the objectives, scope and structure of the case study as well as data collection methodologies used. The design follows the guidelines and best practices suggested by [34].

5.1. Case study design

5.1.1. The case

The case study is conducted in cooperation with a large German enterprise with an international market presence. The German organization employs almost 30,000 employees and generates a yearly revenue in a mid-range double-digit billion number of revenue. Its IT organization which, with some exceptions, follows a federal structure with a central IT governance consists of up to two thousand employees. Around 1,900 applications of different kinds are in use to directly or indirectly support business processes and the daily work of employees. Approximately 200 applications decomposed into more than 2,000 components are operated upon Platform-as-a-Service (PaaS) based cloud environments (Pivotal Cloud Foundry (PCF) [1] and Redhat OpenShift Container Platform [2]).

The case is considered the Enterprise Architecture documentation process with regards to the applications operated on these cloud platforms. This includes the initial situation, which refers to the current state of enterprise architecture documentation (Status Quo), and the target situation, describing the state that can be achieved by applying the suggested solution (Target state). The unit of analysis is the suggested EAMM solution concept and its capabilities assessed within a real-case environment which also includes the analysis of costs and benefits implied from a monetary and non-monetary perspective.

[1]https://pivotal.io/de/platform
[2]https://www.openshift.com/

5.1.2. The evaluation environment

In a narrow sense, the evaluation environment is the part of the IT landscape which Enterprise Architecture should be automatically discovered and documented by the suggested solution. This encloses all applications and their components that are operated on the industry partner's cloud platforms. This also includes the application's ecosystem, meaning other applications or microservices that are interconnected with one another. Within this scope, two agile teams were selected to integrate and test the suggested solution productively. In a broader sense, the solution has to be reflected in its entire ecosystem. This is necessary because it is not working on its own but does affect surrounding processes, tools and actors. As the solution does integrate into deployment pipelines, the software development process and the teams driving it are an important part of the broader evaluation environment. Moreover, also other information sources serve as EA information supplier and import data into the central EAM repository (Iteratec's Iteraplan [3]. These tools should of course work collaboratively rather than impairing each other and introducing data conflicts and inconsistencies.

Having said this, the evaluation environment can be split into a technical perspective and a socio-technical perspective. The technical evaluation environment takes the following components into account:

- The cloud environments (Cloud Foundry, OpenShift) used by the industry partner to operate its applications
- The CI/CD infrastructure (Jenkins) used to deploy application components
- A selection of federated information systems that contain enterprise architecture relevant data and which are integrated into the documentation process
- The central EAM repository (Iteraplan)

The socio-technical evaluation environment consists of:

- The software development process, as it is directly impacted by adopting the solution
- The agile development teams (including product owners and development teams) who take a central role in order to make the solution work
- The central Enterprise Architecture Management department, that is both, provider of the EAM repository and key consumer of EA information
- Domain architects, that currently bear the most manual EA modeling activities

[3]https://www.iteraplan.de/

Central department of Enterprise Architecture

The central department of Enterprise Architecture consists of to focus groups, technology-oriented and business-oriented, where each is comprised of six enterprise architects. The department strives to develop and optimize over-arching architecture solutions which focus on the enterprise as a whole. This includes all business units and the comprehensive integration of economic, business and technology aspects. The most important goals are to leverage synergies, improve re-usability and gain higher flexibility and pace with implementing new endeavors. The department also supports and promotes business-unit and platform-overarching collaboration. Its operational responsibilities include

- Architectural committees
- Definition and communication of architectural directives
- Architectural decision making
- Provision and operation of the EAM repository
- Integration and automation of EA documentation
- Training on the EAM repository and architectural directives

IT landscape overview

The industry partner's IT landscape can be generalized as depicted by 5.1. It consists of an Enterprise Private Cloud layer that is subdivided into a *militarized zone* (Zone A) and a *de-militarized zone* (Zone B). Internet-facing front-end applications located in the *militarized zone* are decoupled from the backend application components located in the *de-militarized zone*. In each of these zones dedicated cloud platform instances are located which host the industry partner's applications. Communication across these zones is either based on message queues or the central API gateway. The industry partner's core business systems and data stores mainly reside in proprietary, traditional data centers. This part of the IT landscape is out of scope of the evaluation. In scope of the evaluation are both, *militarized zone* and *de-militarized zone*, as well as the central API gateway. The message broker is not considered in this work as it does not suit as an EA information source as explained in the later course. The two agile teams that were selected to integrate the solution, develop and operate applications that consist of multiple components which are spread over both zones.

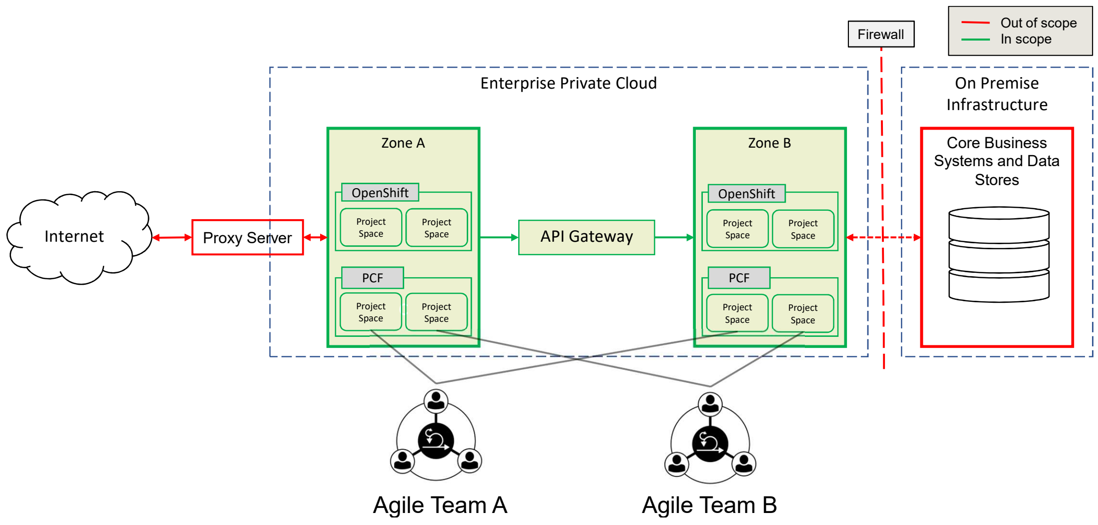


Figure 5.1.: IT landscape overview

5.1.3. Evaluation objectives and structure

Generally speaking, the case study's objective is to assess the suggested solution's capabilities applied to a real case enterprise context. To achieve this, the case study is structured along the following elements:

1. **Analysis of the *Status Quo*:** The goal of this phase is to determine the industry partner's current state of enterprise architecture with regards to documentation completeness, documentation weaknesses and documentation gaps. This should serve as a baseline to measure achieved improvements after applying the suggested solution. Data collection methods used in this phase are quantitative analysis' of the EAM repository's data stock in comparison with the CMDB and cloud platforms as well as the conduction of a questionnaire which gathers expert's perception of EA elements importance, completeness, actuality and change frequency.

2. **Conduction of a *Requirements Analysis*:** The goal of this phase is to derive EA automation requirements and priorities based on the findings in the previous phase. In addition, a semi-structured interview was conducted to record further functional and non-functional requirements.

3. **Analysis of the *Target State*:** This phase adapts the suggested solution to the industry partner's context and qualitatively assesses its capabilities with regards

to the automated discovery and generation EA documentation. It answers the question to what degree the industry partner's target EA meta model (EA model coverage) can be covered and how much of it can be automated (degree of automation).

4. **Adoption and *Productive use*:** This phase validates the qualitative assessment against results obtained under productive use of the solution. For this purpose, the prototype is integrated into the pipelines and applications operated by two agile teams.

5. **Analysis of cost and savings:** The goal of this phase is to determine the solution's costs (implementation, roll-out and operation) and compare it against the saving potential (reduced manual EA modeling efforts) that can be realized in case the solution would be rolled out to the entire organization. Both, expected cost and savings, are based on expert estimations using the three-point.estimation technique.

6. **Evaluation interviews:** This last phase bundles all findings acquired in the previous phases and presents them to experts in the field. The goal is to obtain their feedback about the solution's capabilities, ideas for improvement and their judgment whether or not they would support to roll-out the solution.

Figure 5.2 depicts the individual phases of the case study including in- and outputs.

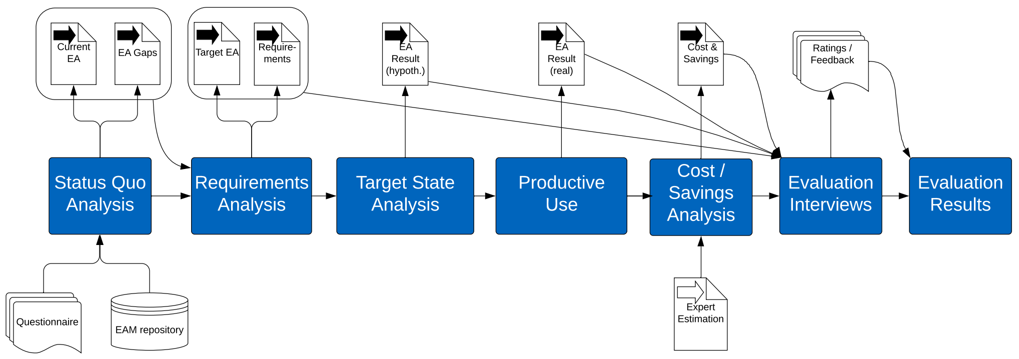


Figure 5.2.: Case study overview

Out of Scope:

Some aspects of EAMM functionality that typically are of relevance are out of scope of this work

- EA change propagation
- Synchronization with other independent data sources
- EA model visualization

5.1.4. Evaluation criteria

The authors in [28] tackled the question of what and how to evaluate information system artifacts in the design science research area. The authors compiled their findings based on over ten years of literature in a taxonomy of evaluation methods [28]. The hierarchy of evaluation criteria comprised by this taxonomy sets the basis for the evaluation at hand. The following subset of criteria is covered by this work:

- Goal:
  - **Efficacy:** The degree to which the artifact achieves its goal considered narrowly, without addressing situational concerns
  - **Effectiveness:** The degree to which the artifact achieves its goal in a real situation
  - **Utility:** Utility measures the value of achieving the artifact's goal, i.e. the difference between the worth of achieving this goal and the price paid for achieving it
  - **Validity:** Validity means that the artifact works correctly, i.e. correctly achieves its goal
  - **Technical feasibility:** Evaluates from a technical point of view, the ease with which a proposed artifact will be built and operated
  - **Generality:** Refers to the scope of the artifact's goal. The broader the goal scope, the more general the artifact
- Environment:
  - **Usefulness:** The degree to which the artifact positively impacts the task performance of individuals
  - **Ease of use:** The degree to which the use of the artifact by individuals is free of effort

  - **Fit into technical IS architecture:** The degree to which the artifact integrates into the technical IS architecture of the organization
  - **Alignment with business:** The congruence of the artifact with the organization and its strategy
  - **Alignment with IT Innovation:** The degree to which the artifact uses innovative IT
  - **Absence of side effects:** The degree to which the artifact is free of undesirable impacts on the technical IS architecture of the organization in the long run

- Activity:
  - **Accuracy:** The degree of agreement between outputs of the artifact and the expected outputs
  - **Performance:** The degree to which the artifact accomplishes its functions within given constraints of time or space. Speed and throughput (the amount of output produced in a given period of time) are examples of time constraints. Memory usage is an example of space constraint
  - **Functionality:** The capability of the artifact to provide functions which meet stated and implied needs when the artifact is used under specified conditions
- Structure:
  - **Correspondence with another model:** The degree to which the structure of the artifact corresponds to a reference model
- Evolution:
  - **Adaptability:** The ease with which the artifact can work in contexts other than those for which it was specifically designed. Synonym: flexibility

5.2. Data collection methods

5.2.1. Quantitative analysis of archival data

Archival data analysis was used for several purposes and included data exports from the central EAM repository, the CMDB, the API gateway and cloud platform data. Cross comparison of the data contained in these sources allowed to determine the current EA documentation completeness as well as to localize and quantify the documentation gap. The obtained figures play an important role in the further course of this work. The figures obtained also allowed to determine potential cost savings based on the interpolation of registered modeling activities and per element efforts estimated by experts.

5.2.2. Semi-structured interviews

Semi-structured interviews are an important instrument during all phases of the case study. 5.1 depicts the interview subjects and the interviews conducted separated by topics. The acronyms contain the function (Enterprise Architect, Software Developer, etc.) an interviewee works in.

	EA1	EA2	EA3	EA4	EA5	EA6	EA7	EA8	EA9	DA1	DA2	DA3	DA4	PO1	PO2	DEV1	DEV2	#
Years Of Experience	8	1,5	11	9	2	5	2,5	5	7	8	10	1	2	3	2,5	6	7	
AS-IS EA Documentation Survey	X	X	X	X	X	X	X											7
Requirements Analysis Interview	X	X																2
Cost & Savings Estimation		X	X				X			X								4
Evaluation Interview		X	X			X	X	X	X	X	X	X	X	X	X	X	X	14
																	TOTAL:	27

EA	Enterprise Architec	6
DA	Domain Architect	4
PO	Product Owner	2
DEV	Software Develope	2

Table 5.1.: Interview register

Interviews conducted are mostly semi-structured, including open questions paired with survey-like questions that asked ratings on a Likert scale. This allows for easier comparison among and between different functions. The following shortly describes the goal and content of interviews conducted as part of the evaluation.

Questionnaire AS-IS EA Documentation: This questionnaire was targeted at the industry partner's central EA department. The goal was to get a quick insight into the current situation and perception about existing EA documentation along different criteria and architectural elements. The questionnaires results are covered in 6.2.

Requirements Analysis Interview: An extensive analysis of challenges and requirements for automated EAMM has already been conducted by research [8], [10], [15], [41]. For this reason, only two selected enterprise architects were interviewed in order to validate whether the requirements identified by literature persist or deviate in the context of the industry partner. Also additional requirements were recorded.

Estimation/Validation of Cost & Savings: To determine the solution's economic feasibility, expert estimations were conducted to (1) approximate the implementation, roll-out and operating costs which go along with the suggested solution and (2) the current efforts spent for manual EA modeling activities. The recording of estimations about activities in a pre-defined structure primarily formed the core of these interviews.

Evaluation Interview: The final evaluation interviews served to obtain independent expert's judgment about the solution's capabilities and value proposition. At the beginning of these interviews, experts were introduced to the solution and presented with the findings acquired throughout the prior phases of the case study. Experts were then asked to give their rating about diverse aspects on a Likert scale and explain their choice. Open questions took an important role in these interviews to get more insight about interviewee's opinions and to gather inspiration for further improvement. Finally, experts had to judge whether they would roll-out the suggested solution.

5.2.3. Three-point-estimation

The three-point-estimation initially was introduced and made popular in the project management discipline. The method is used to estimate the duration of individual activities as part of the program evaluation and review technique (PERT) [1]. The name comes from the estimation of three values (worst case, likely case and best case estimate) that together define the so called PERT distribution. E_e denotes the weighted average that takes into account all three estimates. Formula 5.1 depicts the calculation of the expected value E_e.

$$E_e := \frac{(a + 4 \times m + b)}{6} \tag{5.1}$$

where
E_e = estimation value based on double-triangular distribution
a = the best case estimate
m = the most likely case estimate
b = the worst case estimate

The technique is especially suited in case estimations are linked to high risk and uncertainty. As opposed to the two-point-estimation, the additional estimation of a likely case meets the assumption that the likely case is not centered between worst and best case estimates but has a bias towards one the extremes. This characteristics also apply to the manual modeling of EA. In the context of this work, this method was therefore used to estimate manual EA modeling efforts as well as implementation cost.

Advantages of the technique are that it considers extreme values on both sides of an estimate. It does not force experts to decide for a single best estimate but rather allows them to specify a range based on their experience. The uncertainty inherent to estimations can be made transparent and taken care of respectively.

A disadvantage, as typical for all estimation techniques, is a lack of objectiveness. Therefore one should not rely on the estimate of a single expert. If required, noteworthy deviations can be sorted out in a subsequent discussion. For this reason, all estimations conducted in this work were obtained independently by multiple experts and cross validated afterwards.

6. Evaluation - Case study at a German enterprise

This chapter reports on all activities and findings obtained throughout the case study at a large German enterprise. The sections are organized along the case study elements introduced in section 5.1.3. At the end of this chapter, the key findings and suggestions obtained during the evaluation interviews are compiled and processed to a preliminary, revised approach that serves as a recommendation for future development.

6.1. Definition and mapping of EA modeling concepts

This section shortly explains the most important mappings between EA meta model elements used. This serves a better understanding of this chapter as many different concepts and wordings are in use. Table 6.1 maps the three groups of EA meta model elements that need to be differentiated. The elements used by the industry partner are given in the first column of the table. These elements are mapped against the ArchiMate notation as well as against Iteraplan's specific building blocks, which is in use as the central EAM repository. Special attention has to be put on the threefold classification of applications into *Applications*, *Application Components* and *Services* that are all modeled using Iteraplan's building block *Information System*. The element *Interface* is used to model a relationship between a target and a source *Application*. *Business Channels* are business actors from an ArchiMate perspective and correspond to the industry partner's various user groups. *Business Mappings* are a specific building block in Iteraplan that is used to model multidimensional relationships between an *Information System* and business layer elements including *Business Processes*, *Business Units*, *Business Functions*, *Business Domains* and *Business Objects*. Only one element per type can be part of such a tuple.

Figure 6.1 depicts the industry partner's EA meta model in ArchiMate notation. The element hierarchy of *Business Domain* > *Business Subdomain* > *Business Capability* is one special characteristic of this EA model. Domains and subdomains group any business layer elements. All business layer elements can have connections to each other except for *Product Owners*. *Business Applications* make the most central EA element. It can be further specialized into *Application Components* and *Services*. The definition and

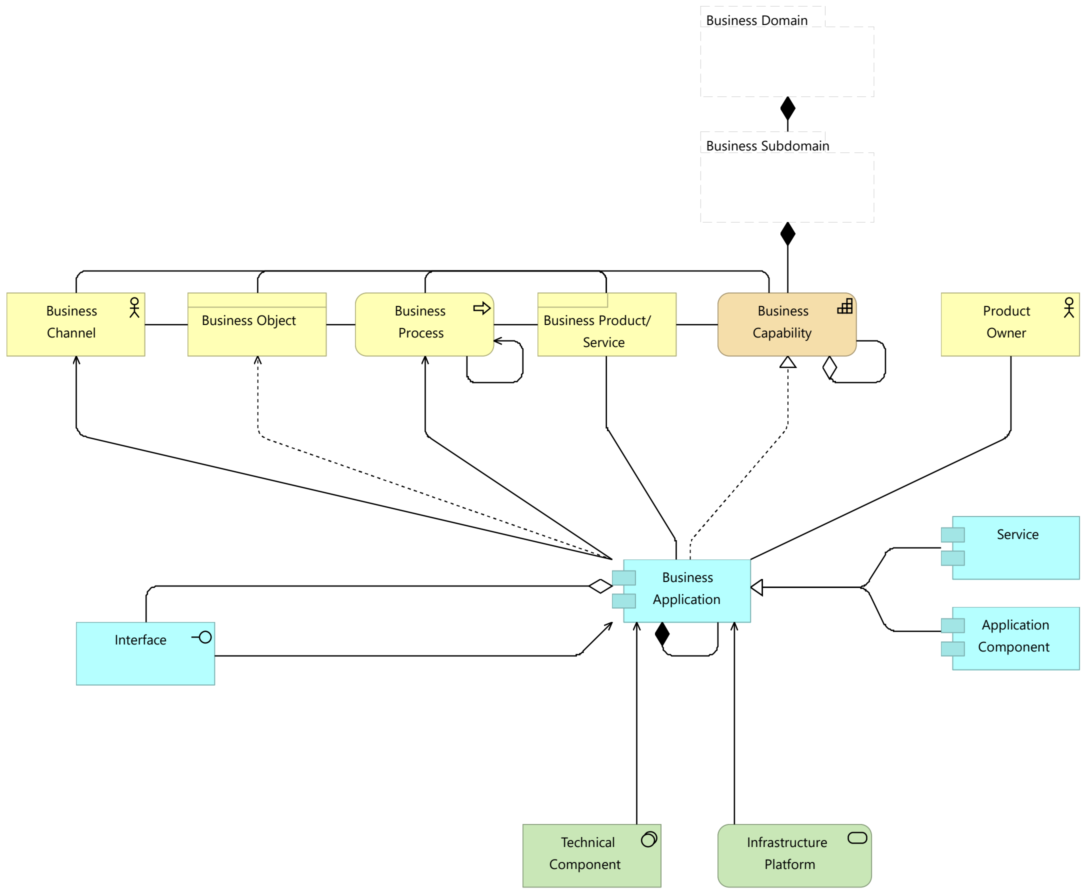
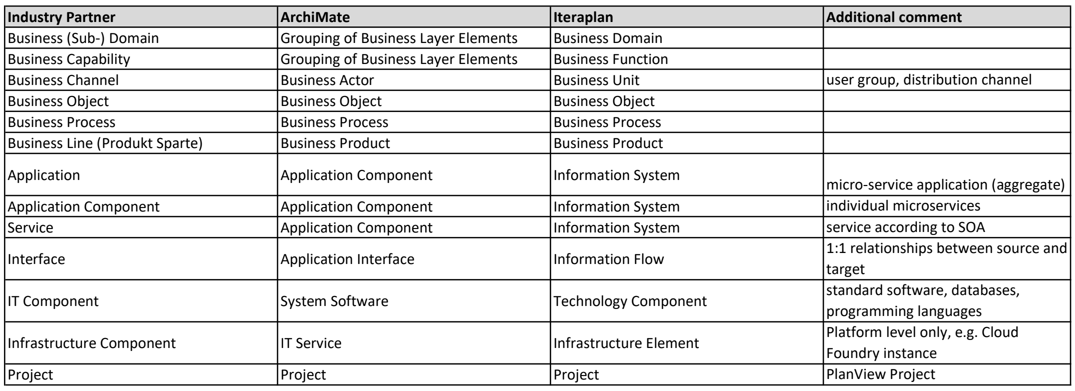


Figure 6.1.: The industry partner's EA meta model

modeling of *Services* is oriented on SOA, that specifies a service as an integral part of a Business Application that makes a specified business logic remotely available to consumers (i.e. Applications or Application Components). *Application Components* are defined as an integral part of a Business Application which provides a specific part of the functionality of the overall Business Application. *Interfaces* are used to model 1:1 communication flows between a source and a target application, enriched by further attributes. *Technical components* serve to model different kinds of technologies used. This includes programming languages, standard software, databases, etc. *Infrastructure Platforms* solely model operating environments on a platform level. There is no further break down into more fine-grained elements such as virtual machines, nodes or similar.

For the remainder of this chapter, the industry partner's wording has to be used from time to time as the differentiation at an *Application Component* level often plays an important role.



Industry Partner	ArchiMate	Iteraplan	Additional comment
Business (Sub-) Domain	Grouping of Business Layer Elements	Business Domain	
Business Capability	Grouping of Business Layer Elements	Business Function	
Business Channel	Business Actor	Business Unit	user group, distribution channel
Business Object	Business Object	Business Object	
Business Process	Business Process	Business Process	
Business Line (Produkt Sparte)	Business Product	Business Product	
Application	Application Component	Information System	micro-service application (aggregate)
Application Component	Application Component	Information System	individual microservices
Service	Application Component	Information System	service according to SOA
Interface	Application Interface	Information Flow	1:1 relationships between source and target
IT Component	System Software	Technology Component	standard software, databases, programming languages
Infrastructure Component	IT Service	Infrastructure Element	Platform level only, e.g. Cloud Foundry instance
Project	Project	Project	PlanView Project

Table 6.1.: Mapping of EA elements

6.2. Status Quo

The purpose of this section is to give an overview of how enterprise architecture documentation is performed at the industry partner. It covers the most important roles and procedures. It also analyzes the current documentation weaknesses and quantifies the documentation gap.

6.2.1. EA documentation responsibilities

Enterprise architecture documentation is mostly performed on a manual basis. Exceptions are automated import interfaces that connect data from the CMDB and the PPM system. The data imported from these systems, in turn, are itself manually documented. Various functions play their part in the documentation process. Figure 6.2 shows the split of responsibilities laid upon the EA meta model.

Enterprise Architects: The central EA department has the overall accountability for the EAM repository, its operation and the EA meta model. In terms of EA documentation enterprise architects are responsible for defining central guidelines and rules for modeling. The department centrally defines and models the most central business layer elements including *Business Domains and Subdomains*, *Business Functions* (i.e. Business Capabilities) and *Products* in close collaboration and alignment with the industry partner's international group level organization. In addition, they develop and maintain interfaces for automatic data import into the EAM repository (CMDB and PPM system).

Domain Architects: Domain Architects work on an overarching level with responsibility for a certain domain or sub-domain. They cover most of manual modeling efforts and ensure that the business and application layer are reasonably documented and interconnected by the most important relationships. This information is mainly obtained by the means of meetings and intensive collaboration with agile teams.

Product Owners: Product Owners should take the responsibility for the documentation of the application they own including the relationships to other applications, provided and consumed interfaces, transferred business objects and the related infrastructure platforms they are operated on. Documentation is limited to an application level, skipping to model the more fine-grained application components. Important to state is, that the application owners responsibility is only mandatory for the CMDB but not for the EAM repository. In practice most application owners never directly use the EAM repository for modeling. The CMDB data is being imported on daily basis on element level. Relationships are not imported which is one of the main reasons for a lack of documentation at this place. In general the CMDB is considered the leading system over the EAM repository.

Development Team: The development team does not have a defined responsibility for documentation in the CMDB or EAM repository. In practice, they often support product owners to create the CMDB model.

Projects: After its introduction, the EA department quickly started to support overarching projects by allowing them to add project specific attribute groups to the EAM repository's meta model. This means that a range of projects and their members act cross-cuttingly on most of the application layer elements. As a result a considerable amount of attributes is project-specific.

CMDB Import: Before the EAM repository has been introduced, the industry partner already modeled its applications as so called *Business Services* in the CMDB (ServiceNow[1]). This is part of a mandatory process to application owners which serves to obtain approval to bring their application to the production stage. As the CMDB is closely connected to operative IT service management processes such as change, incident or request management, the main purpose is, to ensure that all necessary preconditions are met to ensure a successful go-live. The modeling requirements also include up- and downstream dependencies (*used by*, *uses)*. In total 50 attributes are being imported from the CMDB into the EAM repository on a daily basis. This integration is unidirectional, meaning that if the data in the EAM repository is changed manually, there will be no synchronization back to the CMDB. There is also no name- or ID-based merging with existing elements but only a check for the CMDB's proprietary ID.

PPM Import: Currently Iteraplan's building bock *Project* is solely used by PPM system data imports. This reflects a project from a mostly financial perspective. Typically such elements include dates, descriptive texts and attachments that originated from the financial legitimization process. The import does not include any relationship information to other EA elements.

Naming conventions and global identifier: For applications, no naming conventions or global, system-independent identifiers are in place that would allow for a clear matching of an application's representation in different systems. As different systems are typically used by different roles, the used naming most often does not match across these systems.

6.2.2. EA documentation weaknesses

At the beginning of the case study, a questionnaire was conducted with the goal to analyze the perceived as-is situation with regards to the following criteria:

[1]https://www.servicenow.com/products/servicenow-platform/configuration-management-database.html

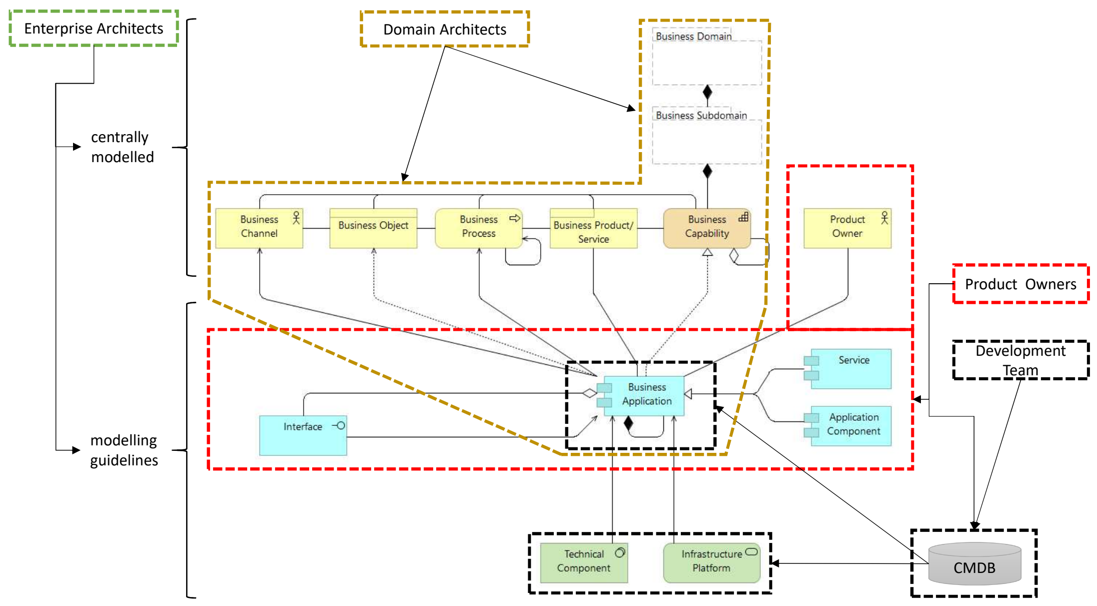


Figure 6.2.: EA documentation responsibilities

- Importance of a given EA meta model element
- Completeness of the data stock of a given EA meta model element
- Actuality of the data stock of a given EA meta model element
- Change frequency of a given EA meta model element
- Degree of automation of a given EA meta model element

These criteria were structured along the three main architecture layers (business layer, application layer and technology layer) and the inter-relationships between these layers. In addition to that some specific attributes were also included so that in total 41 architecture elements and attributes were judged on a Likert scale with regards to each of the specified criteria. The results were used to identify the perceived documentation gaps most painful to the organization but also to derive requirements and priorities with regards to the enterprise architecture model elements the automation solution should cover in first place. Table 6.2 shows an extract of the survey results including the average rating per asked element and rating variance.

In addition to that the EAM repository was analyzed quantitatively to validate the questionnaire findings and to reveal issues with the current data stock, that might be unknown to date. Steps that were undertaken are

- Analysis of quantities and relative amount of inter- and intra-specific relationships
- Analysis of attribute usage, the degree to which modeled attributes are actually filled with values
- Analysis of the time since last update
- Analysis of modification users and modification rates (automatic vs. manual)

As compared the to the other architectural layers the **business layer's** documentation is the one perceived most complete, even though an average rating of 2.59 indicates that there are still deficiencies. Whereas business domain's and project's documentation completeness is rated highest, business roles and actors have the lowest rating within the business layer. It needs to be mentioned that both, roles and actors are modeled as attributes not as own elements in the industry partners meta model. Business domains, subdomains and capabilities as well as products and business units are centrally defined and modeled by the EA department in close alignment with the global group organization. Products currently are only modeled at a very high level. It is subject to future development and concrete use cases whether this is broken down into more fine grained structures.

At the **application layer**, the highest documentation deficiencies are with interfaces, data flows and dependencies and intra-specific relationships. In contrast, these elements are rated as highly important (average importance rating ranges from 1.1 to 1.8) which makes this gap more severe to the organization. Despite the automated data import from the CMDB, applications are also perceived to be rather incomplete (avg. completeness rating 2.60) due to the fact, that the CMDB is itself modeled manually. There is a need to improve this value as all interviewees rated this EA element as essential (avg. importance rating 1.0). So far, it has been a conscious decision by the industry partner, not to model application components as this would cause too much manual modeling effort. This also explains the low average rating of 3.2 for this element. Due to the high number of project-specific attributes the overall degree of attribute usage is just around 18 percent. Deducting the project specifics, the percentage is still only 19.92 percent. On average the last update took place 56.36 days ago, whereof approximately 16 percent of all elements were updated within the last quarter, app. 30 percent within the last half a year and 40 percent within the last year. The portion of applications that is being imported from the CMDB is 28.57 percent.

The **technology layer** is of least relevance to the industry partner (average importance rating of 2.54). Still, its documentation completeness is perceived slightly better as compared to the application layer, which is in contrast way more relevant (average

	Importance		Completeness		Actuality		Change frequency	
	Variance	Average	Variance	Average	Variance	Average	Variance	Average
Business Layer								
Business processes	0,12	1,14	0,69	2,86	0,24	3,57	0,57	3,00
Business capailities	0,00	1,00	0,56	2,20	0,56	2,20	0,80	3,00
Business objects	0,00	2,00	0,24	2,60	0,16	2,80	0,96	3,20
Business domains and subdomains	0,20	1,29	0,20	1,71	0,49	3,29	0,20	1,71
Actors (customers, partners, employees)	0,69	1,86	0,53	3,57	0,20	3,71	2,00	3,50
Roles (product owner, developer, unit head, etc.)	1,27	2,17	2,82	3,00	1,96	2,83	2,82	3,60
Product domain	0,49	1,50	1,92	2,67	3,27	4,00	2,69	2,60
Product	0,24	1,43	1,63	2,71	0,53	3,57	1,10	1,67
Department	0,78	2,29	0,98	2,14	0,98	2,86	0,57	2,00
Business functions (marketing, accounting, etc.)	1,14	2,33	1,84	3,67	2,49	3,20	2,78	3,20
Projects	0,82	2,57	0,49	1,71	0,98	1,86	0,29	2,00
Intrapecific relationships (within business layer)	0,00	1,00	0,78	2,29	0,78	2,71	0,20	2,71
Application Layer								
Application (logical aggregate of components)	0,00	1,00	0,24	2,60	0,64	2,40	0,24	2,40
Application component	0,16	2,80	0,16	3,20	0,56	2,80	0,24	2,40
Interface (external application behavior)	0,12	1,14	0,41	3,14	0,78	2,71	0,00	3,00
Technical domain	2,78	3,00	2,82	3,60	3,55	3,75	3,55	3,75
Data flow and dependencies	0,24	1,43	0,20	3,29	0,53	2,43	0,00	3,00
Intrapecific relationships (within application laye	0,78	1,80	2,82	3,60	1,84	2,60	1,84	3,00
Technology Layer								
Instance (running process)	0,49	1,71	1,67	2,83	1,71	2,33	1,63	3,17
Database (Mysql, MongoDB, etc.)	1,39	2,57	2,20	2,67	1,35	3,17	1,67	3,00
Runtime environment (OS, host, cloud platform)	1,55	2,14	2,12	2,50	1,84	3,33	2,49	2,67
Virtualisation technique	2,12	2,50	2,82	3,00	1,71	3,50	2,82	3,00
Communication technology (e.g. protocols)	1,63	3,17	1,71	3,50	1,71	3,50	1,92	3,83
Physical IT resource (server, router, network devi	2,12	2,17	1,96	3,00	1,84	3,33	2,49	2,67
Intrapecific relationships (within technology laye	3,43	3,50	3,92	4,00	3,14	3,50	3,55	3,75
Interspecific relationships between business and application layer								
Business domain - application component	0,00	1,00	0,12	2,86	0,78	3,29	0,20	2,71
Product - application component	0,12	1,14	0,12	3,14	1,92	3,17	1,10	2,83
Business process - application component	0,12	1,14	0,12	2,86	0,69	3,14	0,12	2,86
Project - application component	0,20	1,71	0,12	3,14	0,98	2,86	0,12	3,14
Actor - application component	0,98	1,86	0,24	3,43	1,67	3,00	1,63	3,17
Business function - application component	1,10	1,43	1,10	3,00	1,67	2,83	1,06	2,67
interspecific relationships between application and technology layer								
Application component - instance	0,29	1,17	1,35	3,17	1,06	2,67	1,35	3,17
Application Layer (Attributes)								
Version	0,86	2,33	2,20	2,67	2,00	2,33	2,00	2,33
Lifecycle state	0,78	2,00	1,35	2,67	0,98	2,50	1,35	2,67
Compliance and data protection	0,82	1,83	1,35	2,67	0,98	2,50	1,14	2,33
Last deployment/update	1,67	3,00	1,67	3,00	1,67	3,00	1,67	3,00
Technology Layer (Attributes)								
Technology (NodeJs, JEE, .Net, etc.)	1,67	2,83	1,71	3,50	1,92	3,83	1,71	3,50
Runtime data (saturation, availability, requests, e	2,29	2,33	2,82	3,00	1,96	4,00	2,82	3,00
Event data (Incidents, MTTR, MTTF, etc.)	2,53	2,20	2,78	3,20	2,82	3,60	3,14	2,80
Complexity	1,67	3,00	2,00	3,50	3,55	3,75	1,92	3,83
Cost structure (TCO, running costs, licenses)	1,39	1,83	1,92	3,17	1,92	3,83	1,96	3,00
Usage classification (business vs. technical, orche	2,49	3,17	2,41	3,33	3,14	3,50	2,00	3,50
Software dependencies	0,98	2,50	1,71	3,50	2,24	2,83	1,84	3,33

Table 6.2.: Extract of the conducted survey regarding current EA documentation

importance rating of 1.86). It's notable, that the variance of received importance rating is highest for technology layer elements, which indicates that the interviewees do not have a uniform perception here. In the EAM repository *Technical Components* typically cover technologies used (e.g. databases, protocols, programming languages) as well as organization-specific or generic reference architectures followed. *Infrastructure Elements* that in total have a two-digit number only, only cover operating platforms such as cloud platforms. During the interviews, it was stated that this is sufficient for the industry partner's EAM use cases. Therefore this is explicitly not considered a weakness. In case details about infrastructure elements are required, the CMDB is being consulted.

Inter-Relationships, i.e. relationships between elements of different architectural layers, have an average importance rating of 1.35 and thus, have the highest importance to the industry partner. Also **Intra-Relationships**, i.e. relationships between elements within an architectural layer, are perceived important (avg. importance rating of 2.10). Hence, it is all the more severe that the rating of documentation completeness only ranges between 3.09 to 3.30 for these EA model elements. As all kinds of relationships are modeled fully manually, interviewees stated doubts regarding the actuality of the data stock (average actuality rating of 2.97). The EAM repository confirms that only a third of registered applications are equipped with a relationship to surrounding applications or a superior or inferior system. Only 15 percent bear a relation to a (*Technical Component* and only 27 percent to an *Infrastructure Element*). 29.2 percent of applications are connected over the element *Information Flow*. 54.2 percent of information systems on average have 5.9 relationships to business layer elements whereas 45.8 percent do not have any. Intra-specific relationships at the technology layer are at a even lower level. However, this is not perceived a weakness considering an average importance rating of 3.50. At the business layer the numbers for intra-specific relationships looks more satisfactory even though an average rating of 2.29 indicates that there are weaknesses.

During the case study, it was found that there are numerous **modeling errors**. EA model elements were used incorrectly and relationships were modeled inconsistently. This especially is an issue for EA reporting as filter results might either be falsified by unnecessary elements or miss parts of the desired result. This leads to a low level of data reliability and slows down decision making. A reason for this issue is different user groups with different skills and knowledge about EA modeling. The EAM tool provides many different possibilities to connect items with each other but doesn't give guidance how to correctly use them.

Table 6.3 summarizes the figures stated in the section above.

	Importance	Completeness	Actuality	Change Frequency
Business Layer	1,71	2,59	3,05	2,68
Application Layer	1,86	3,24	2,78	2,93
Technology Layer	2,54	3,07	3,24	3,15
Intra-Relationships	2,10	3,30	2,94	3,15
Inter-Relationships	1,35	3,09	2,99	2,94

Table 6.3.: Rating per layer averaged over elements

6.2.3. Documentation gap

To quantify and validate the perceived documentation gaps, figures from the EAM repository, CMDB and cloud platforms were obtained and compared to each other. The comparison allows to draw conclusions and assumptions to determine the expected documentation gap. However, this needs to be interpreted with special care as only for few architecture elements the *ground truth*, i.e. the total amount, is known with certainty.

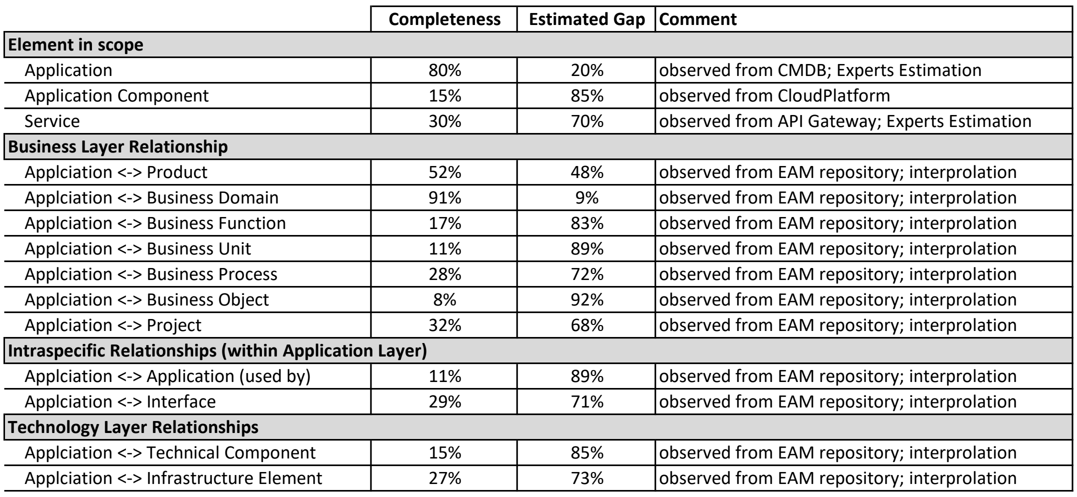
Application Layer Elements: As a common baseline, the scope is restricted to applications, application components and services operated on cloud platforms and being in production stage. The cloud platform delivered the total amount of deployed applications. This is considered the ground truth. Based on this figure the documentation gap of application components is almost certain. The CMDB delivered to total amount of cloud-based applications. The data contained is modeled manually as demanded by an obligatory process. Expert estimate this to be be 80 percent complete. Thus, an application would aggregate approximately nine application components each. For services, there is a lack of a reliable source, therefore the total amount is unknown except for a subset that is governed by the central API gateway. Based on feedback received in the questionnaire and further estimations by enterprise architects the current data stock of services is expected to be 30 percent complete.

Application Layer Relationships: The ground truth for such relationships is unknown. The completeness of relationships between EA elements is almost impossible to judge as most of them reflect one-to-many or even many-to-many relationships. In order to approximate the gap, the average amount of relationships per existing application based on the current EAM data stock was used for interpolation. This constitutes a rather pessimistic approximations as experts perceive the current data stock as incomplete.

Business Layer Relationships: The ground truth for these relationships is also unknown. Still, based on the analysis of the EAM repositories' data stock it was possible to determine the ratio of unconnected applications (i.e. application that do not connect to a single business layer element of a kind). The underlying assumption is that an application must at least be connected with one of the business layer elements each (e.g. at least one process must be supported, at least one domain must be assigned, etc.). The ratios calculated were used for interpolation to approximate the documentation gap.

Technology Layer Relationships: In order to approximate the documentation gap, the same approach as for business layer relationships was used.

Table 6.4 shows the results for this approximation. In the later course of this work these approximations will play an important role to determine the cost and savings potential for the documentation gap to be closed by the help of the suggested solution.



	Completeness	Estimated Gap	Comment
Element in scope			
Application	80%	20%	observed from CMDB; Experts Estimation
Application Component	15%	85%	observed from CloudPlatform
Service	30%	70%	observed from API Gateway; Experts Estimation
Business Layer Relationship			
Applciation <-> Product	52%	48%	observed from EAM repository; interprolation
Applciation <-> Business Domain	91%	9%	observed from EAM repository; interprolation
Applciation <-> Business Function	17%	83%	observed from EAM repository; interprolation
Applciation <-> Business Unit	11%	89%	observed from EAM repository; interprolation
Applciation <-> Business Process	28%	72%	observed from EAM repository; interprolation
Applciation <-> Business Object	8%	92%	observed from EAM repository; interprolation
Applciation <-> Project	32%	68%	observed from EAM repository; interprolation
Intraspecific Relationships (within Application Layer)			
Applciation <-> Application (used by)	11%	89%	observed from EAM repository; interprolation
Applciation <-> Interface	29%	71%	observed from EAM repository; interprolation
Technology Layer Relationships			
Applciation <-> Technical Component	15%	85%	observed from EAM repository; interprolation
Applciation <-> Infrastructure Element	27%	73%	observed from EAM repository; interprolation

Table 6.4.: Quantified EA documentation gap

6.3. Requirements analysis

The requirements analysis is fed by three sources

1. Questionnaire regarding the perception of the as-is EAM documentation (see previous section)
2. Quantitative analysis of the EAM repository
3. Semi-structured interview with two EA department representatives

Whereas the questionnaire and the quantitative analysis served to inductively derive requirements from identified documentation weaknesses and gaps, the interview directly asked the experts for requirements they pose. As a basis, requirements identified by literature were used.

6.3.1. Automation priorities

With the goal to derive priorities for the automation of EA documentation, a score was build over the questionnaire's criteria importance, completeness, actuality and change frequency. Weights were aligned with enterprise architects and applied to the criteria importance (factor 2.0) and completeness (factor 1.5) so that the scoring calculates as depicted in formula 6.3.1.

$$score_e := 2 \times \underbrace{|i \bmod -5|}_{\text{rating reversal}} + 1.5 \times c + a + \underbrace{|f \bmod -5|}_{\text{rating reversal}} \qquad (6.1)$$

where:
$score_e \in \{5.5, ..., 22.5\}$, denotes the need for automation for an architecture element e
$i \in \{1 \textit{ (essential)}, \ldots, 4 \textit{ (irrelevant)}\}$, denotes the importance rating
$c \in \{1 \textit{ (complete)}, \ldots, 4 \textit{ (not documented)}\}$, denotes the completeness rating
$a \in \{1 \textit{ (up-to-date)}, \ldots, 4 \textit{ (not documented)}\}$, denotes the actuality rating
$f \in \{1 \textit{ (very often)}, \ldots, 4 \textit{ (seldom)}\}$, denotes the change frequency rating

The equation's part "$|i \bmod -5|$" serves to reverse the ratings from e.g. "1" to "4", "2" to "3", etc. This way, all criteria follow the same schema from "1" as *low* to "4" as *high*. The score reflects the need for automation. The higher the value the higher the automation priority. In case of the EA element *Interface* the following value is calculated:

$$score_{interface} = (2 \times |1 \bmod -5| + 1.5 \times 3 + 3 + |4 \bmod -5|) = 16.5$$

Figure 6.5 shows the entire list of EA elements and attributes, ranked by the described score. As enterprise architects follow two different focuses (IT focus and business focus) the table includes an extra column for these focus groups. The column *difference* denotes the deviation between them. It can be stated that most higher deviations appear for business layer elements where business-oriented enterprise architects tend to give a higher rating regarding the importance and a lower rating for completeness. The technology layer attribute *cost structure* can be interpreted as business-oriented. For all other elements and attributes asked, there is rather small deviation between the two groups.

Business Layer Elements were excluded from the ranking as these elements are modeled centrally by the EA department. Thus, there is no need for an automated discovery for these elements. This, of course, does not include relationships towards the business layer. Still, table 6.5 depicts the score values. An important remark is that the industry partner's meta model currently only supports the elements *Business Domains and Subdomains, Business Processes, Business Capabilities* and *Product*.

An eye-catching finding is that all **Inter-Relationships** belong to the Top10 within the ranking. This indicates that there is a high need for automating the discovery and the documentation of relationships between architectural elements. Intra-specific relationships within the application layer (rank 5), i.e. communication dependencies between applications or application components also takes a high position.

Most **Application Layer Elements** elements are placed within the Top10 due to the high importance and the poor documentation. An interesting exception from this are *Application Components* that only rank on position 24. This corresponds to the current circumstance that the industry partner focuses on the modeling of applications only, and skips to modeled more fine-grained components. Components are perceived as being too expensive for manual modeling.

Despite being perceived as poorly documented, **Technology Layer Elements**, except for *Instance (running process)*, take lower positions in the ranking. The reason behind this are rather low ratings for the criteria *importance*, which has an average of 2.54.

6.3.2. Functional and non-functional requirements

The requirements analysis also comprised a semi-structured interview with two representatives of the EA department responsible for the EAM repository. Its structure and content is set up oriented on requirements to EAMM formulated by [8] and potential

EA Element / Attribute	EA Layer	Avg. Score	Score EA - IT focus	Score EA - Business focus	Difference	Rank
Application component - instance	Relationship	17,92	17,5	18,3	0,83	1
Interface (external application behavior)	Application	17,71	17,2	18,1	0,96	2
Data flow and dependencies	Application	17,64	17,8	17,5	0,33	3
Business function - Application	Relationship	17,33	16,5	18,2	1,67	4
Intrapecific relationships (within application layer)	Application	17,20	18,0	17,0	1,00	5
Product - Application	Relationship	17,17	16,5	17,8	1,33	6
Instance (running process)	Technology	17,08	17,3	16,8	0,50	7
Application (logical aggregate of components)	Application	17,00	17,5	16,8	0,67	8
Actor - Application	Relationship	16,83	15,5	18,2	2,67	9
Business domain - Application	Relationship	16,71	16,5	16,9	0,38	10
Business process - Application	Relationship	16,71	16,0	17,3	1,25	10
Project - Application	Relationship	16,57	16,0	17,0	1,00	12
Software dependencies	Technology	15,75	16,0	15,5	0,50	13
Cost structure (TCO, running costs, licenses)	Technology	15,25	12,3	18,2	5,83	14
Lifecycle state	Application	15,17	14,3	16,0	1,67	15
Compliance and data protection	Application	15,17	14,3	16,0	1,67	15
Event data (Incidents, MTTR, MTTF, etc.)	Technology	14,60	15,7	13,0	2,67	17
Physical IT resource (server, router, network device	Technology	14,50	14,0	15,0	1,00	18
Version	Application	14,33	13,3	15,3	2,00	19
Technical domain	Application	14,25	16,5	13,5	3,00	20
Communication technology (e.g. protocols)	Technology	14,25	13,5	15,0	1,50	20
Intrapecific relationships (within technology layer)	Technology	14,25	13,0	14,7	1,67	20
Technology (NodeJs, JEE, .Net, etc.)	Technology	14,25	14,2	14,3	0,17	20
Application component	Application	14,13	13,5	14,3	0,83	24
Virtualisation technique	Technology	14,00	13,0	15,0	2,00	25
Complexity	Technology	13,88	15,5	13,3	2,17	26
Database (Mysql, MongoDB, etc.)	Technology	13,83	13,5	14,2	0,67	27
Runtime data (saturation, availability, requests, etc	Technology	13,83	14,0	13,7	0,33	27
Runtime environment (OS, host, cloud platform)	Technology	13,75	13,0	14,5	1,50	29
Usage classification (business vs. technical, orchest	Technology	13,75	13,5	13,8	0,33	29
Last deployment/update	Application	13,50	11,7	15,3	3,67	31
Business processes	Business	16,43	14,5	17,9	3,38	---
Use cases	Business	16,07	16,2	16,0	0,17	---
Business Capabilities	Business	17,25	18,0	17,0	1,00	---
Business Objects	Business	15,50	16,5	15,2	1,33	---
Business domains and subdomains	Business	13,43	12,2	14,4	2,21	---
Actors (customers, partners, employees)	Business	16,58	14,7	18,5	3,83	---
Roles	Business	15,60	10,5	19,0	8,50	---
Product domain	Business	14,90	11,0	17,5	6,50	---
Product	Business	13,92	12,0	15,8	3,83	---
Department	Business	12,79	9,5	15,3	5,75	---
actors (customers, partners, employees)	Business	16,40	17,0	16,0	1,00	---
Project	Business	12,57	12,5	12,6	0,13	---
Intraspecific relationships (within business layer)	Business	16,43	14,5	17,9	3,38	---

Table 6.5.: Automation priorities - ranking by score

EA information sources analyzed by [10]. The interview consists of the following sections:

- Usage and importance of potential information sources
- Review of the questionnaire results and automation priority ranking
- Recording of function and non-functional requirements to the prototype

Usage and importance of potential information sources

As the solution concept puts runtime data in a central position, the question about the current use and perceived importance of such information sources with regards to EA documentation purposes was asked. The answers are depicted in Table 6.6.

	In use	Not used	Very important	important	Less important	Not involved
Cloud APIs	☐	☒	☒	☐	☐	☐
Tracing Data	☐	☒	☐	☐	☒	☐
Monitoring Data	☐	☒	☒	☐	☐	☐

Table 6.6.: Usage and perceived importance of runtime information data sources

Cloud platform data is judged highly important for EA documentation as it can provide a full inventory of deployed application components. The enterprise architects considered this data the ground truth and a powerful information source due to the rich information that is being exposed by APIs.

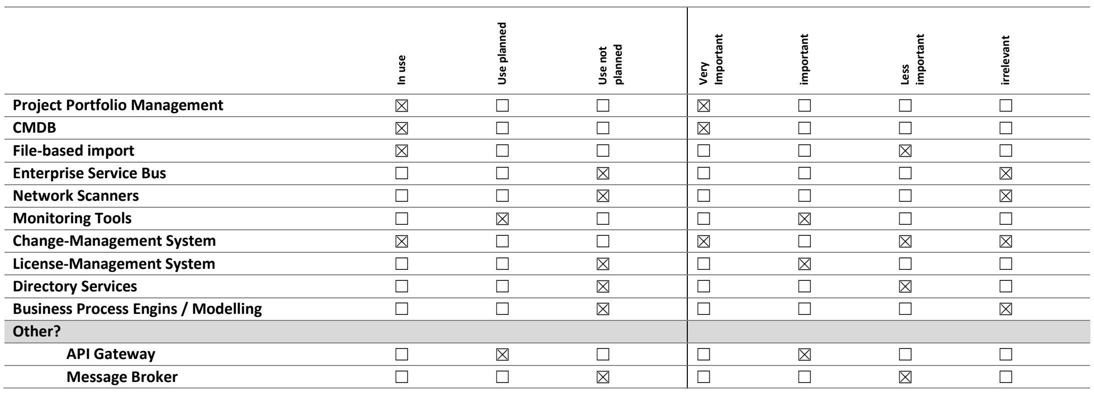
In terms of **Tracing data**, the tool Dynatrace[2] is being used at the industry partner. Its usage is fully optional and up to the agile team's individual decision. Usually, it is not used continuously but rather temporarily in the context of troubleshooting or performance analysis. Due to this, Dynatrace doesn't make a reliable source for EA documentation as only fractions could be covered.

The industry partner uses Prometheus[3] for metrics collection and Grafana[4] for visualization purposes. **Monitoring data** is judged as highly important mainly due

[2]https://www.dynatrace.de/
[3]https://prometheus.io/
[4]https://grafana.com/

to EA use cases that might be (partly) automated based on monitoring data. EA use cases that could be driven by the incorporation of monitoring data are the validation of 12-factor-app[5] criteria for a given application and operational stability assessments (e.g. fulfillment of resilience patterns).

Next, the experts were asked about their judgment of federated information sources that could be integrated by the prototype. The answers are depicted in figure 6.3.



	In use	Use planned	Use not planned	Very Important	important	Less important	irrelevant
Project Portfolio Management	☒	☐	☐	☒	☐	☐	☐
CMDB	☒	☐	☐	☒	☐	☐	☐
File-based import	☒	☐	☐	☐	☐	☒	☐
Enterprise Service Bus	☐	☐	☒	☐	☐	☐	☒
Network Scanners	☐	☐	☒	☐	☐	☐	☒
Monitoring Tools	☐	☒	☐	☐	☒	☐	☐
Change-Management System	☒	☐	☐	☒	☐	☒	☒
License-Management System	☐	☐	☒	☐	☒	☐	☐
Directory Services	☐	☐	☒	☐	☐	☒	☐
Business Process Engins / Modelling	☐	☐	☒	☐	☐	☐	☒
Other?							
API Gateway	☐	☒	☐	☐	☒	☐	☐
Message Broker	☐	☐	☒	☐	☐	☒	☐

Figure 6.3.: Usage and perceived importance of federated information sources

Automated data import from the CMDB, PPM system and file-based imports are already used by the industry partner. The option *Change Management System* was also checked as *used* as changes are handled along with the CMDB (same tool). Still, no change data is being imported as there is no demand for such information. While Buschle et. al. [4] already examined the value of an Enterprise Service Bus for EA documentation purposes, the industry partner judges it as irrelevant in their context as it is about to be decommissioned. The department already thought about integrating the license management system, however, the endeavor failed due to issues in establishing a reliable mapping between the system's elements. There is lower to no interest in network scanners, business process modeling systems and directory services. As the enterprises cloud architecture blueprint makes use of a layered structure, communication between them, are passed either through central API gateways (ApiGee[6]) or message queues (Active MQ[7]). For this reason these systems were added to the list. The message broker however only covers a small portion of traffic between the layer and therefore is of less relevance for EA documentation.

[5]https://12factor.net/de/
[6]https://docs.apigee.com/
[7]https://activemq.apache.org/

Review of the questionnaire results and priority ranking:

This part of the interview served to review the questionnaire results and derived automation priorities as presented in the previous section. Experts were asked whether the calculated score and raking meets their expectations towards the automation of EA documentation. The weights applied in the score calculation formula 6.3.1 were adjusted accordingly to reflect the experts' requirements.

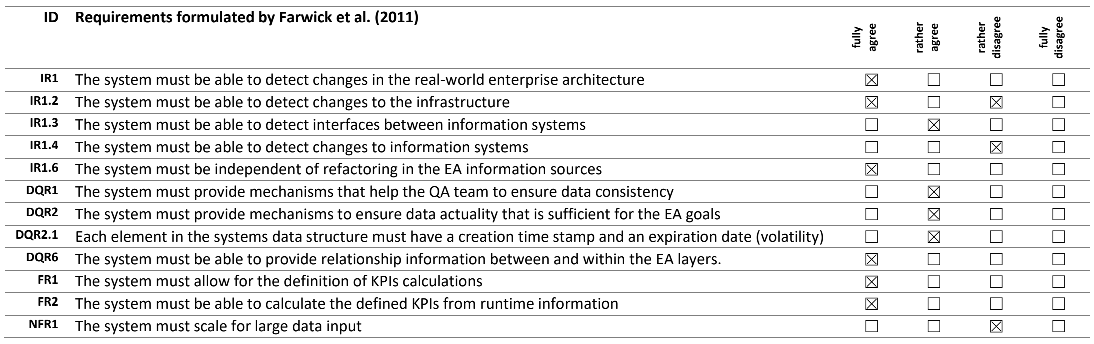
Recording of functional and non-functional requirements to the prototype

In this section the enterprise architects were presented a list of requirements suggested by literature. In addition, they had the opportunity to formulate requirements freely.

With regards to functional requirements, a subset of the elements compiled by Farwick et al. [8] were used as a baseline for this section of the interview. Figure 6.4 displays the consent of the enterprise architects with these requirements. As presumed, high consent was stated for the capability to detect changes to the real-world enterprise architecture, the capability to detect relationships among architecture layers and robustness against refactoring in used EA information sources. These are considered fundamental functionalities. Also the ability to detect interfaces between information systems, mechanisms to assure data consistency and appropriate data actuality as well as time-stamping belongs to the functionalities considered basic. The definition of KPIs and calculation based on runtime data are features that would cause excitement about the solution but are not considered mandatory. The reason behind this judgment is that the industry partner has a set of architecture assessments in place that are performed regularly. Currently they mainly consist of the fulfillment of checklists and adherence to best practices and reference models. The opportunity is, to partly automate these procedures and become capable of continuously monitor applications for their compliance with given standards. With regards to the detection of changes to the infrastructure and information systems rather disagreement was expressed. As the industry partner is not interested and also does not import infrastructure elements into the EAM repository, this functionality is not required. In general, is was stated that it is sufficient to know which platform (e.g. OpenShift, Cloud Foundry instance) an information system is operated on. Changes to information system are of low relevance if interpreted according to IT Infrastructure Library (ITIL)[8]. Relevant changes

[8]http://www.officialitil4.com/

to information systems are only those that actually have an impact to the architecture. Such changes are covered by *IR1*.



ID	Requirements formulated by Farwick et al. (2011)	fully agree	rather agree	rather disagree	fully disagree
IR1	The system must be able to detect changes in the real-world enterprise architecture	☒	☐	☐	☐
IR1.2	The system must be able to detect changes to the infrastructure	☒	☐	☒	☐
IR1.3	The system must be able to detect interfaces between information systems	☐	☒	☐	☐
IR1.4	The system must be able to detect changes to information systems	☐	☐	☒	☐
IR1.6	The system must be independent of refactoring in the EA information sources	☒	☐	☐	☐
DQR1	The system must provide mechanisms that help the QA team to ensure data consistency	☐	☒	☐	☐
DQR2	The system must provide mechanisms to ensure data actuality that is sufficient for the EA goals	☐	☒	☐	☐
DQR2.1	Each element in the systems data structure must have a creation time stamp and an expiration date (volatility)	☐	☒	☐	☐
DQR6	The system must be able to provide relationship information between and within the EA layers.	☒	☐	☐	☐
FR1	The system must allow for the definition of KPIs calculations	☒	☐	☐	☐
FR2	The system must be able to calculate the defined KPIs from runtime information	☒	☐	☐	☐
NFR1	The system must scale for large data input	☐	☐	☒	☐

Figure 6.4.: Consent of enterprise architects with requirements formulated by Farwick et al. [8]

In addition to this, the following requirements were demanded explicitly:

- Capability to detect and document deployed applications (inventory)
- Capability to group application components to a logical super-ordinate application
- Capability to detect hierarchies and dependencies between applications
- Smooth integration of the solution into the software development process and pipelines
- Capability to export the data into the existing EAM repository
- Data collection for the assessment of architecture policies and guidelines

As the industry partner does not wish to have an additional front-end next to the existing EAM repository, explicitly not demanded were the following functionalities:

- Visualization of deployed artifacts
- Visualization of communication dependencies
- Visualization of architectural changes over time (history)

Summarizing the requirements analysis that incorporates the information gathered by questionnaires and semi-structured interviews the following key requirements are captured. Generally speaking, the industry partner primarily requires a solution for the automated EA discovery and data integration into an existing EAM repository. A separate front-end for architecture visualization purposes is not desired. With the scope of used cloud environments this solution is required to solve the following challenges:

- **R1 - Application Inventory:** Closure of the the EA documentation gap in application inventory in an automated manner
- **R2 - Application Relationships:** Closure of the EA documentation gap in application inter-relationships in an automated manner
- **R3 - Business Layer Relationships:** Capability to connect discovered applications with respective business layer elements
- **R4 - EA Model Maintenance:** Capability to detect changes to the real-world EA and ensure sufficient actuality of EA models
- **R5 - Ease of Use:** Impairment to the agile development process needs to be kept as low as possible, the solution itself must be centrally manageable and maintainable
- **R6 - Data Integration:** discovered EA data must be importable into the existing EAM repository
- **R7 - Automation Priorities:** Consideration of automation priorities as per Table 6.5

6.3.3. Ead.json adjustment

The first adoptions of the prototype quickly revealed limitations with the initial scope and usage of the *ead.json* configuration file. The original version only expected singular business domain assignments while the industry partner required the possibility for multiple assignments and a set of other business layer assignments. The *ead.json* therefore needed to be adapted to the industry partner's needs. The revised version of the configuration file consisted of the following specifications:

The fields *business_domain, business_subdomain, business_product and business_units* reflect the industry partner's most important business layer assignments. Theses elements are centrally defined in the EAM repository and are almost static. It is therefore possible to incorporate them into the configuration file without the risk of outdating soon. Referenced federated information systems contained are the CMDB and the central API gateway. Both fields are optional as there might be applications that are not registered in these systems. The reference to the CMDB allows to consolidate and merge the EA documentation imported by *EAD* into the EAM repository with the EA data imported by the CMDB. The reference to the API gateway allows to discover which APIs a given application consumes over the API gateway.

```
{
  "application_component_name": "a human understandable name of the application component (
        mandatory)",
  "description": "short description of the application component (optional)",
  "superordinate_application": "top level application name the application component belongs to (
        mandatory)",
  "product_owner": "email of the product owner (mandatory)",
  "business_domain": "the business domains the component belongs to (mandatory)",
  "business_subdomain": "the business subdomain the component belongs to (mandatory)",
  "business_product": ["array – set of products the application component supports (optional)"],
  "business_unit": ["array – set of business units the component supports (optional)"],
  "cmdb_id": "the ID of the corresponding element in the CMDB (optional)",
  "api_gateway_client": "the unique name of the corresponding client registered in the API gateway (
        optional)"
}
```

Figure 6.5.: Adjusted ead.json according to the industry partner's needs

6.4. Target state

After analyzing the as-is situation and gathering requirements from the industry partner, an in-depth analysis was performed to identify the degree to which the requirements can be fulfilled by the use of the suggested solution concept. This includes observations from the practical adoption and use of the solution in a real case setting as well as qualitative analysis of EA information sources that are required to satisfy the information demand. In this section, the following questions will be answered

1. What are the information sources required to satisfy the information demand (section 6.4.1)?
2. To what extent can they contribute to the EA model (section 6.4.2)?
3. To what extent can this be automated (section 6.4.3)?

6.4.1. EA information sources

In the first step, potential information sources are analyzed which could satisfy the industry partner's information demand. The goal of this analysis is to identify the most valuable EA information sources that finally can be incorporated into the suggested solution approach. The baseline for the selection of tools to be analyzed is that they contain data that most closely reflects the current EA. The data contained should be in close relation with productive systems. Moreover, it should not be documented manually as this would increase the risk to build upon outdated and incomplete data. Therefore, the focus was put on platforms used to run and operate applications including the development tool-chain.

The potential information sources are categorized as **config files** which are static files that provide configuration and other meta information about an application, **software development tools** and **runtime information** that originates from the operating environments. A rating $r \in \{1, ..., 4\}$ was applied where "1" stand for highest adequacy and "4" for very low adequacy. This rating refers to the adequacy of a given information source to retrieve data about the EA element in question. Cells that remain empty stand for *not suited* or *no information contained* about the respective EA element. The ranking at the lower end of figure 6.6 was calculated by weighting the reversed rating by its automation priority rank which resulted in a score, the higher the better. This score finally was ranked. For time reasons only one representative system of a kind could be analyzed. It is assumed that the tools of different vendors contain similar information.

The ratings are based on analyzing the individual data source publicly available documentation and real-case examples assessed at the industry partner. Not all ratings can be justified at this point but the most important clusters are explained in the following paragraph.

Config files can hide surprisingly much EA relevant information, especially within the application layer and even about some relationships. Examples are container configurations such as *Docker-files*[9] *and Docker-compose files*[10] or Kubernetes resource configuration files which are used to define *Services*[11], *Deployments*[12] *or Pods*[13]. They allow to enrich their content by meta information which can be added using tags or labels. This feature could be used to set the artifact into relation of business layer elements. The authors in [13] demonstrated the value of extracting information from *Docker-files* and *Docker-compose* files in combination with runtime information to recover microservice architectures. Config files used to be contained in the source code repository and thus, are directly accessible from the CI/CD servers working directory after being checked out. This makes them directly available within the CI/CD pipeline for further processing. Relationship information is held by config files insofar that they reference other components they depend on (e.g. Docker-compose file attribute "depends on"). Assessing real-case examples, it was found that these files often contain variables instead of being hardcoded. As variables cannot be resolved when being read

[9]https://docs.docker.com/engine/reference/builder/
[10]https://docs.docker.com/compose/compose-file/
[11]https://kubernetes.io/docs/concepts/services-networking/service/
[12]https://kubernetes.io/docs/concepts/workloads/controllers/deployment/
[13]https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/

		Automation Priority Rank	ead.json	manifest.yml	docker file	docker compose file	SCM (GitHub)	PPM (JIRA Sowftware)	code inspection (SonarQube)	CI/CD pipelines (Jenkins)	BRM (Artifactory, Docker Hub)	cloud platforms (CloudFoundry)	api gateway (ApiGee)	distributed tracing (Dynatrace)	service mesh (Istio)
EA artefacts / attributes	**Layer**		**config files**				**software development tools**					**runtime information**			
actors (customers, partners, etc)	L1	11					2	2				2	2		
business processes	L1	14													
intraspecific relationships (within business	L1	14	3				3	3				4			
business functions	L1	16													
use cases	L1	17													
roles (product owner, developer, unit hea	L1	19					2	2				2			
product domains	L1	23													
products	L1	32					3	1				4			
business somains and subdomains	L1	39													
deparments	L1	40													
projects	L1	41					2	1							
interface (external application behavior)	L2	2		4								3	1	1	1
data flow and dependencies	L2	3		4		4						3	2	1	1
intrapecific relationships (within applicati	L2	5	4			4						2	1	1	1
application (logical aggregate of compone	L2	13	2		4	2	3					2		2	2
application component	L2	16	4		3	3	3					1		1	1
Compliance und Datenschutz	L2	21	3												
lifecycle state	L2	21	2										2		
version	L2	26			1		1				1	2			
technical domain	L2	27	1		4							4			
last deployment/update	L2	38								1	1	1			
application component - instance	Rel	1										1		1	1
business function - application componen	Rel	4	1		4							4			
product - application component	Rel	6	1		4			1				3			
actor - application component	Rel	8	1				2	2				2	2		
business domain - application component	Rel	9	1		4							4			
business process - application component	Rel	9	1									4			
project - application component	Rel	12	3				3	3							
instances (running process)	L3	7										1		1	1
software dependencies	L3	18					1								
cost structure (TCO, running costs, license	L3	20										3			
event data (Incidents, MTTR, MTTF, etc.)	L3	24					3	3	2	2		1	1	1	1
physical IT resource	L3	25												1	
communication technology (e.g. protocols	L3	27												1	1
technology (NodeJs, JEE, .Net, etc.)	L3	27			3	3	2					2			
intrapecific relationships (within technolo	L3	27												1	2
virtualisation technique	L3	31										1			
complexity	L3	33					3		1						
database (Mysql, MongoDB, etc.)	L3	34										2	1	2	2
runtime data (saturation, availability, requ	L3	34										1	1	1	1
runtime environment (OS, host, cloud plat	L3	36										1		1	1
usage classification (business vs. utility)	L3	36	1												
average adequacy:			2,00	4,00	3,38	3,20	2,36	2,00	1,50	1,50	1,00	2,31	1,44	1,14	1,23
unweighted Rank:			7	12	11	10	9	7	5	5	0	8	3	1	2
Rank weighted by automation priority:			**4**	**12**	**8**	**9**	**6**	**7**	**10**	**13**	**11**	**1**	**5**	**2**	**3**

Figure 6.6.: EA information sources - coverage matrix

from the CI/CD servers working directory, this makes this source less reliable (therefore rated with "3" or "4"). The *ead.json* is exempt from this limitation as it is an integral part of the suggested solution concept and is dedicated to hold static information with regards to EA. It is an ideal vehicle to map business-related information such as domain assignments, supported business units, etc. to the application. They are also easier to maintain than labels that are scattered across multiple docker files.

Software development tools are a valuable source for identifying actors and their roles which are involved in the development process (e.g. developers, project leads, etc.) and putting them into relation to the application. Given that the respective features are used, software products and related components can be precisely represented (e.g. by JIRA components or GitHub Releases). In GitHub, application components often have an individual repository and are grouped to a logical application by GitHub Organizations. GitHub Projects and JIRA Projects can be interpreted as *Projects* form an ArchiMate perspective as they store work packages intending to transform a given EA element. Software characteristics such as software versioning, complexity and technologies being used can be found within VCS system. Continuous Code Inspection tool such as SonarQube[14] used for static code analysis and testing, store software test results, vulnerabilities and other code issues which indicate the software's quality and complexity. Details about deployments and releases versions can be extracted from BRM system (e.g. Arfifactory[15] or DockerHub[16]) or the CI/CD server. Such tools often provide their objects, metrics and event data via REST APIs that could be incorporated for EA purposes.

The group of **runtime information** sources consists of the operation environment itself (e.g. cloud platforms), tools that govern services and their communication such as API gateways and service meshes and finally, distributed tracing technologies which allow to analyze communication flows. These tools provide runtime information, metrics, actors and relationships of different kinds. Still, there are differences in adequacy to be considered. Cloud platforms reliably reflect the ground truth about deployed applications (application inventory) and a broad range of information across the application and technology layer. Also some relationships, for instance, bound cloud services, exposed URLs accessible for external applications and related actors can be retrieved from these platforms. Application logs produced are often stored in central logging solutions such as the ELK-Stack[17] that can be further analyzed to

[14]https://www.sonarqube.org/
[15]https://jfrog.com/artifactory/
[16]https://hub.docker.com/
[17]https://www.elastic.co/de/what-is/elk-stack

discover communications between individual applications deployed. Though, cloud applications have dynamic IP addresses which frequently change. This implies that there is usually a small time window where communication paths can be discovered based on simple logs. This issue was exemplary observed using Cloud Foundry which logs reference the IP address of an incoming request's source only. Mapping to the requesting application is only possible as long one of its instances bears the respective IP address. An easier solution to the discovery of communication paths, therefore, is the analysis of distributed tracing data. This technology (cf. OpenTracing[18]) equips request headers with certain IDs that can be matched across all logs of involved applications. Therefore, it allows to recover the exact end-to-end path a request took through the IT landscape. Dependent on the level of instrumentation, such tracing solution reach from application-level elements down to infrastructure-level devices. A potential alternative are service meshes. For instance, Istio[19] places a sidecar-proxies along with deployed services which allows to govern inter-service communication across the mesh uniquely and can visualize and provide this information via REST APIs. Such a mesh can be spanned across multiple cloud platforms.

Regarding this set of potential EA information sources, it can be stated that there is a gap in the field of business layer elements. No tool is a direct representative for one of the business layer elements, instead, objects contained only be interpreted as such. For example, a JIRA project owner or a GitHub team could be interpreted as *Business Actors* from an EA perspective. A Jira project can be interpreted as an IT product. Whether this is suited or not depends on the individual companies EA meta model and requirements. Resulting to this, another weakness is with relationships between business and application layer. The only vehicle found suited for such assignments is the solution concept's *ead.json* file. Compared to tagging or labeling it has the advantage that it can be maintained at a single central location in the code repository. However, the number of required entries should be kept on an acceptable level to ensure its maintainability.

Based on the calculated ranking which considers the industry partner's automation priorities and information demand, the following tools were selected to be integrated into the suggested solution as they contribute the most information to the target EA model.

1. Cloud Foundry as **cloud platform**
2. ApiGee as **API gateway**

[18]https://opentracing.io/
[19]https://istio.io/

3. GitHub as **SCM**
4. A set of config files including
   - *ead.json*
   - readme file as an alternative source for components descriptions
   - manifest.yml files as a source to extract the artifacts name in the cloud platform

Despite their promising ranking, service meshes and distributed tracing are dropped from the selection of information sources. At the industry partner, Dynatrace is not used for continuous tracing but on demand only in context of troubleshooting or performance analysis. As of now, Dynatrace therefore does not make a reliable source for EA documentation. Service meshes are not in use at the industry partner.

6.4.2. EA element and attribute coverage

This section analyses in detail which parts of the industry partner's EA meta model (cf. 6.1) can be satisfied by the help of the suggested solution approach and the set of selected EA information sources.

Mapping Cloud Foundry to the EA meta model

Figure 6.7 illustrates the fundamental concepts of Cloud Foundry interpreted as ArchiMate elements. This should facilitate the mapping from a simple object model to an EA meta model. All of the elements are retrievable using the cloud platform's exposed REST APIs. A *Droplet* is a OCI[20]-compliant container which results from the staging process. It contains the executable application from the build process and a *Container Image / Buildpack* (defines the runtime environment) and a *Stack* (specifies the base operating and file system being used). These elements are interpreted as *System Software*. There can be multiple instances (interpreted as *Node*) of an app which run on a cluster. Cloud *Services* can be bound by an app. They provide standard software functionalities such as databases, autoscalers and monitoring agents. Thus, cloud services are interpreted as *System Software*. *Apps* are the most central element in Cloud Foundry around which most other element are organized. Apps represent an ArchiMate *Application Component*. The Cloud Foundry allows to organize apps in two hierarchies. *Organizational* and *Space* level while a space is part of an organization. *Apps* can be reached remotely using defined *Routes* which belong to a specified *Domain*. In other words, *Routes* and *Domain* represent an application's fully qualified domain name (FQDN) and

[20]https://www.opencontainers.org/

a certain endpoint URL. Both are interpreted as *Application Interfaces*. With regards to the industry partner's meta model *Apps* are considered *Application Components* whereas bound *Service Instances* are modeled as *Technical Components*. *Droplets, Container, Stack and Packages* are handled as application attributes. The cloud platform itself is modeled as *Infrastructure Platform*. Routes and Domains are modeled as application attributes.

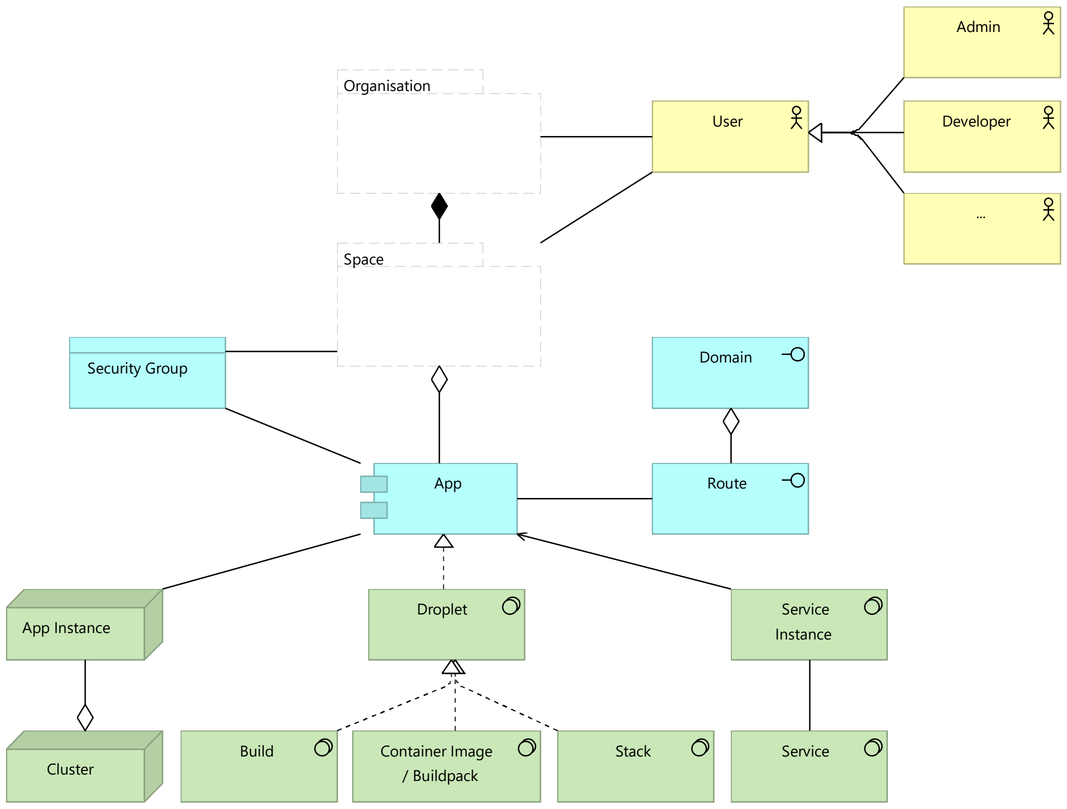


Figure 6.7.: Reverse engineered Cloud Foundry meta model outlined in ArchiMate notation

Mapping ApiGee to the EA meta model

ApiGee's and GitHub's meta model (simplified) interpreted in ArchiMate notation are depicted in 6.8. API providing applications are represented by *API Products* that bundle a set of *API Proxies* which cover individual *API resources*. Both can be modeled as *Application Service* from an ArchiMate perspective. The API providing application itself is not directly represented in ApiGee. Still, a mapping between API proxies and the origin API is easy to achieve based on the FQDN both elements have in common. API Consumers (also called Clients) are managed as *Companies* and their *Developers*. Both have to be registered in the ApiGee Developer Portal. In ArchiMate, they are represented by a *Business Actor* element. Developers and Companies can register their applications as *Consuming Apps* in order to obtain access to a desired API Product published in ApiGee. In the industry partner's meta model, each published API proxy is considered a *Service* exposed by an given application component. API providers and API consumers are interconnected using the *Interface element*.

Mapping GitHub to the EA meta model

The source code versioning system GitHub essentially is organized around repositories of source code where different branches and releases of code reside. These elements are interpreted as *Artifacts* from an ArchiMate perspective. Individual *Users* or user groups, bundled by *Teams*, contained in GitHub are represented as *Business Actors*. They work on one or more repositories. Activities and tasks around one or more repositories can be managed as part of a *Project*. A set of repositories can be grouped using a GitHub *Organizations*.

Linking EA information sources

For all information systems integrated into *EAD*, the questions needs to be answered how objects and information contained in these systems can uniquely be linked to each other. As elaborated by [20], business layer related information can be linked to a concrete cloud application along with the deployment pipeline by passing the *ead.json* as additional input. This works for any kind of data as well as references to federated EA information systems by including a target URL or ID which references the given artifact. Figure 6.8 shows this part of the process in the upper center. The repository that contains the source code of an artifact can automatically be mapped to the deployed cloud application as it passes the CI/CD pipeline, where the repository is checked out, then build and deployed to the cloud platform. In case of ApiGee, one needs to differentiate API providers from API consumers as the mapping works differently. In case an application consumes APIs published via ApiGee, the *ead.json* must reference the registered *Consuming App* that represents the application. This reference makes it possible to retrieve information about consumed APIs and their providers by calling ApiGee's REST API. For API providing applications the mapping process can be fully automated. Registered *API resources* can directly be matched based on their base URL that equals the FQDN of the providing application. This means that a particular Cloud Foundry application can be looked up in ApiGee based on its FQDN. Matching API Proxies are then considered to be exposed *Application Services* by this application. For API consuming applications this is not possible as ApiGee registers applications based on certificates[21]. An automatic matching would therefore require to integrate the certificate store to match an application with its certificate name followed by a lookup in ApiGee. Thus, adding a reference to the *ead.json* is a more lightweight approach.

In the appendix, Figure A.1 gives more details about what the relevant API endpoints for information retrieval are.

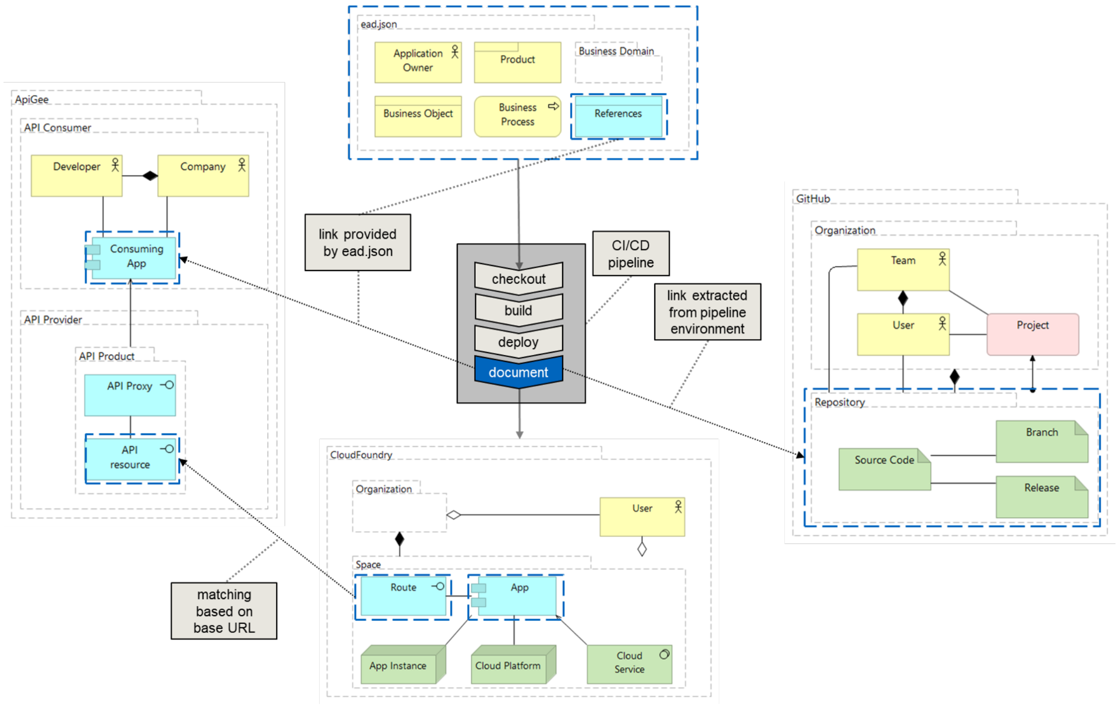


Figure 6.8.: Linkage of EA information sources

[21]this might deviate from organization to organization

6.4.3. Automation potential

Figure 6.9 puts the contribution of these EA information sources together, transformed to the industry partner's target meta model. The percentages given on the left reflect the *Status Quo* state of EA repository completeness (cf. section 6.2.3) versus the *Target state* of EA repository completeness. The baseline was determined by the documentation gap analysis in section 6.2.3. As the ground truth of what *complete* actually means is for some element (*Technical Components, Interfaces and Services* is unknown, corresponding values are approximations. The same is valid for *Relationships* that have a many-to-many or one-to-many multiplicity.

Business layer relationships, i.e. relationships of an application or application component towards business layer elements, can be created up to 100 percent. Though, this relies on the availability and completeness of the *ead.json* provided. Therefore the creation and maintenance of these relationships are considered semi-automatic. To ensure completeness it is indispensable that all agile teams properly adopt the solution. Therefore, this should be enforced by governance or technical means to ensure the adoption.

The same is valid for **Applications** where the approach relies on the specifications made in the config file. At the industry partner, there is no reliable way to technically discover logical applications, i.e. the product of several deployed application components. An aggregation based on *Spaces* (in Cloud Foundry) or *namespace* (in Kubernetes) that are used to organize application components makes a feasible solution. However, this was rejected due to a lack of reliability, as the naming of such spaces does often not correspond to a certain application's name or other components are contained that would falsify the result. As no naming conventions or other global identifiers are in place that would allow to map components and applications, the *ead.json* was the only possible solution.

Application Components can be discovered automatically a 100 percent using the cloud platform's REST API. Provided that the *ead.json* config file is provided along with the deployment process, an attribute coverage of up to 60 percent can be achieved. Otherwise only limited attribute documentation is possible.

Services and **Interfaces** can only be discovered in case the application component is registered in the API Gateway. As previously mentioned, a fully automated discovery is only possible for API providing applications. For API consuming applications, a unique reference has to be added to the *ead.json* file. Based on the analysis published APIs and clients registered in ApiGee, the current data stock of *Services* could be doubled from 30 to 62 percent. Attributes can be filled up to 48 percent automatically. The total amount for *Interfaces* can only be approximated. Based on interpolation, the integration of ApiGee would result in a plus of approximately 21 percent of elements.

Relationships that connect source and target application over these elements would increase proportionally. For *Interfaces* the attribute coverage percentage is up to 75 percent.

Most important **Technical Components** an application component is build upon can be discovered from Cloud Foundry and GitHub. This includes bound cloud services (e.g. databases, autoscaler, monitoring agents, etc.), technology stack, programming languages and software dependencies. The portion of element coverage, however, cannot be determined as there is no fixed definition of what technical components an application consists of. Nevertheless, this can cover approximately up to 80 percent of relevant technical components which are of relevance to the industry partner.

Relationships to the **Infrastructure Platforms** a given application component is operated on, can be determined in any case fully automated.

Summarizing this section one can state that with regards to the **Business Layer**, the suggested solution strongly depends on the availability and correctness of the *ead.json* file which has to be created manually. The process should therefore be enforced by governance or technical means to ensure the success of the solution. Also quality assurance mechanisms are crucial to validate the quality of information contained (cf. section 4.3.3). Regarding the **Application Layer**, the solution is able to provide full transparency on application components including most important **Technology Layer** elements. The API gateway as a source to discover *Services* and application relationships proofed to be valuable, however, is far off providing on overall picture. In order to close the remaining gap in application relationships, the analysis of distributed tracing data is indispensable. The element and attribute coverage ratios will further be used to calculate the cost savings potential in section 6.6.

6.5. Productive use - adoption to real-case projects

In total, the prototype was rolled-out to two projects with the goal to get insights on the solution's behavior under real-case settings. Another goal was to get qualitative feedback from Software Developers and Product Owners after integrating the concept into their environment. This qualitative feedback is included in section 6.7 together with all other findings gathered as part of the evaluation interviews.

6.5.1. Integration efforts and perception

From a processual perspective, the teams did integrate the solution using known Scrum procedures. First, a user story was created which stored the integration requirements and then was implemented as part of a regular sprint. The first adoption (i.e. the first

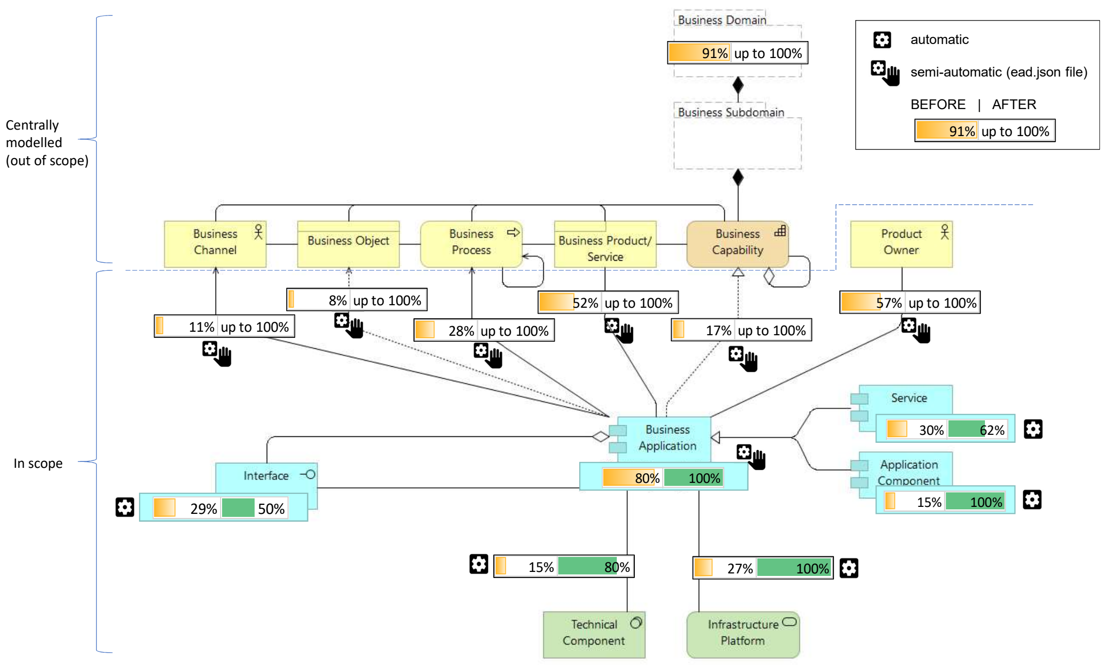


Figure 6.9.: Automation potential

ead.json to be created and the first pipeline to be instrumented) took between four to five hours effort for both, the pipeline integration and initialization of the *ead.json*. One software engineer had issues with adding the *ead-library* due to missing admin rights which caused a delay (3 hours for pipeline integration) whereas the other team was able to integrate and test the functionality within a single hour. Difficulties with the creation of the *ead.json* were caused by a lack of knowledge about business assignments but by specifying the correct references to federated information systems. The teams expected to require between 0.5 to 1.0 hours for each further artifact to be integrated due to learning effects. The effort needed is perceived as acceptable. None of the team did report impairments to the execution of the deployment pipelines beside a longer but acceptable duration. Similar to the *ead-libraries* test environment, the documentation stage took around 1.5 to 1.75 minutes to complete. Table 6.7 summarizes the figures.

All involved subjects perceived the integration procedure as easy to implement. Still, four to five hours for the initial adoption is a fairly long time which indicates that the process should be further improved and facilitated. Especially additional support to correctly fill the *ead.json* file is necessary as software engineers claimed that the manual was not always clear enough about how to specify the required values. This mainly was attributed to references to federated information systems. There were uncertainties of what the correct path should be. Developers also requested a more meaningful console based feedback about the success or failure of the EA documentation stage.

	Team 1		Team 2	
	number of services			
	5 micro-services		9 micro-services	
	integration efforts			
	Product Owner	Developer	Product Owner	SW-Architect
effort for pipeline integration (1st Adoption)		< 3,0 h		< 1,0 h
effort for ead.json initialization (1st Adoption)	< 2,0 h			< 3,0 h
average for further adoptions	< 1,0 h	< 1,0 h		< 1,0 h
	pipeline duration			
EAD stage duration	1:33 min		1:47 min	
	perception			
percieved complexity	very low	very low	very low	low
effort for ead.json is acceptable	fully agree		fully agree	fully agree
effort for pipeline integration is acceptable	fully agree	fully agree	fully agree	fully agree

Table 6.7.: Feedback received from pilot teams

6.5.2. EA documentation results and findings

This section shows exemplary results of one of the selected projects. The main difference between the current EA model and the one created by *EAD* is a deviation in abstraction levels. The current, manual documentation is based on application level whereas *EAD* documents EA based on application component level. Figure 6.10 shows the project's manually modeled EA on the left hand side. The right hand side depicts the EA modeled by *EAD*. The comparison brought forth the following findings:

1. From a business layer perspective, the solution is capable to provide an equal EA documentation as the existing one, thanks to the *ead.json* file.

2. The solution was able to identify all application components. Compared to the preexisting model, there is a mismatch in the level of abstraction (application level vs. application component level). Therefore both models can complement each other. To achieve this, the *ead.json* should reference the identifier of the preexisting EA element. Based on this, *EAD* can connect discovered application components to the preexisting application-based EA models they belong to. As a result, current EA documentation is enriched rather than substituted. From the industry partner's perspective this is a reasonable approach as the preexisting documentation contains many attributes that cannot be covered by *EAD*.

3. The solution was able to discover exposed services and their clients from the Api Gateway. A considerable mismatch was identified when comparing the resulting elements with the preexisting EA model. It was previously known that the API gateway only governs a certain part of the communication traffic between applications. Therefore, the assumption was, that a subset of interfaces and relationships modeled in the current EA could be automatically detected. Instead, almost none of the relationships contained in the API gateway mapped to the preexisting documentation. Following to this, a cross comparison between application relationships modeled in the EA repository, the CMDB and the API gateway was conducted. It revealed that between all of these sources only few entries matched. The reason identified behind this finding, is that manually modeled relationships abstract much of the details of a communication path. They rather are logical end-to-end aggregation of the first and last node of a path. Opposed to that, the relationships discovered by *EAD* refer to technical interfaces being published and consumed. This used to cover a segment of an end-to-end communication flow and therefore most often cannot be matched to existing EA documentation. Still, *EAD* was able to identify a number of relationships that were not at all documented in the current EA model.

4. On the technology layer a considerable information surplus could be achieved. All discovered components could be related to the *Infrastructure Platform* they are operated on. Additional details about bound cloud services, the technical stack and used programming languages could be documented which was absent in the existing EA models.

5. Other elements modeled in the current EA, such as retired applications and projects are not covered by *EAD*.

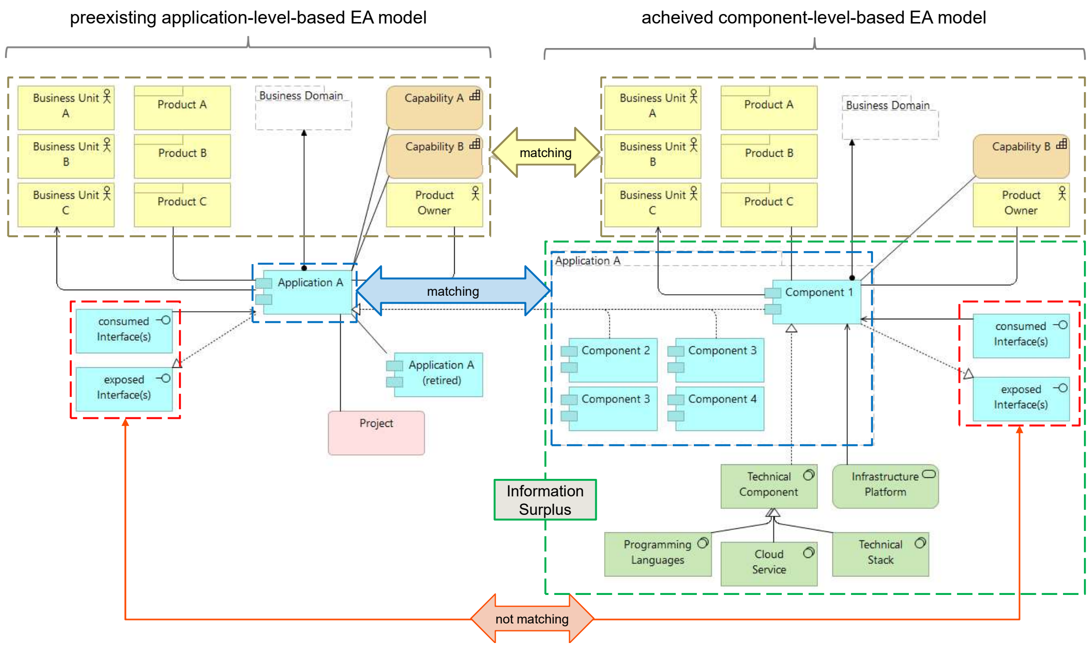


Figure 6.10.: Schematic comparison of current and achieved EA model

As summarized in section 6.2, the EA documentation of *Business Applications* is perceived 80 percent complete due to the automated import from the CMDB and a mandatory modeling process for this system. Over a 150 fields are attributed to the EA element *Application*. This includes architectural assessments and decisions, organizational details, project related information and attributes relating to regulatory and compliance requirements. As such attributes can hardly be automated, it is clear that the preexisting documentation cannot be substituted by *EAD*. Still, *EAD* is capable to contribute much relevant EA information and close current documentation gaps to large extent. Therefore a consolidation of the preexisting and the achieved EA models is suggested as depicted in figure 6.11.

To achieve this, **GUID based matching** needs to be used to ensure a unique mapping of application components to the preexisting documentation on application level. For the industry partner's case the CMDBs unique identifier is used. This identifier is stored in the EAM repository along with the data import. Agile teams have to add the identifier to the *ead.json*. This allows to connect application components discovered by *EAD* to their respective super-ordinate application. The questionnaire presented in section 6.2 revealed that the application component level is of less relevance than the application level. For this reason, information gathered on the more fine-grained component level should be aggregated and replicated to the application level. All relationships and interfaces should be visible on this level. The same applies for the *Infrastructure Platforms* (i.e. cloud platform) the application is operated on. Optionally, also discovered technical components could be aggregated. However, in this example this isn't the case due to the relatively low importance of these elements. As a result of this aggregation the application level documentation can be completed and further enriched while a drill down to the more fine-grained component level is still possible. Benefits of this approach are (1) that preexisting documentation is retained and further enriched and (2) enterprise architects as well as agile teams can find the elements in the EAM repository they are familiar with.

Drawbacks of this, however, are blurred boundaries between manual and automatic modeling. This bears the risk of modeling conflicts that need to be handled. For instance, recurring updates from *EAD* could override manual changes. The EAM repository supports the reservation of writing access to individual meta model objects and attributes to specified users and roles. This could solve some of the conflict potential on an element level but is insufficient for an attribute level. The EA repository should still allow for the flexibility of manual modeling. Therefore, an appropriate trade-off is necessary.

Concluding this section, it can be stated that all application components were successfully identified by *EAD*. This is a considerable information surplus compared to existing EA models which do not contain application components. The *ead.json* fulfilled its purpose and delivered relationships to business layer elements and the super-ordinate application. On the technology layer the existing EA model can be enriched with information previously not documented. The API gateway delivered a number of exposed services and consumed services and allowed to connect providing and consuming applications. Figure 6.12 depicts the difference in modeled EA elements between the preexisting and the achieved EA model for both agile teams. The percentages given indicate the completeness compared to the documentation contained in the applications' IT concept. For some elements the percentage could not be determined. This comparison revealed that gaps in relationship documentation are still considerable.

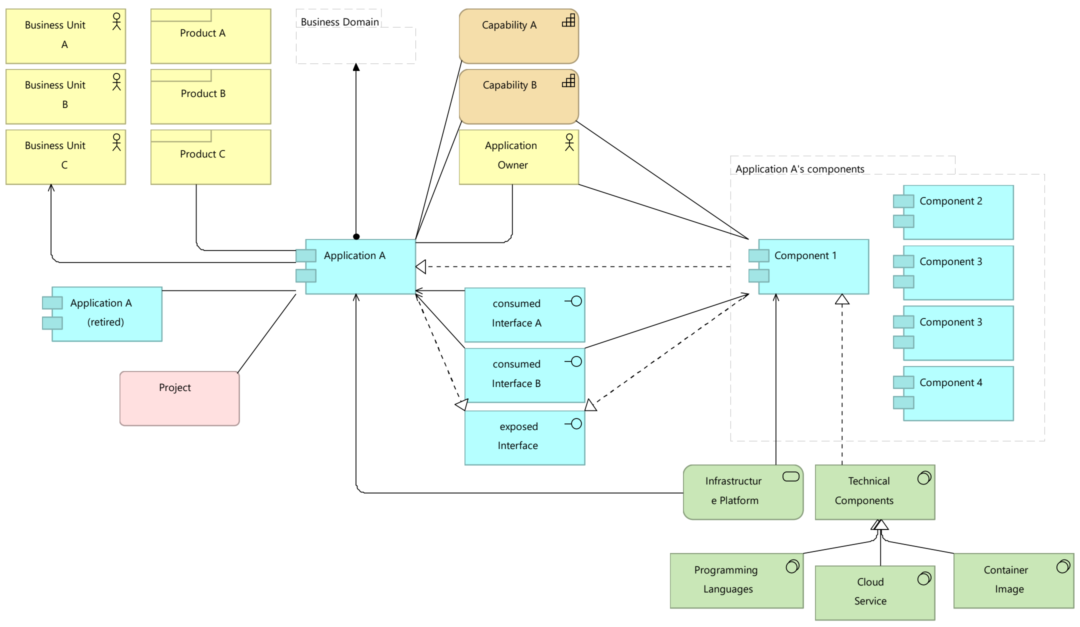
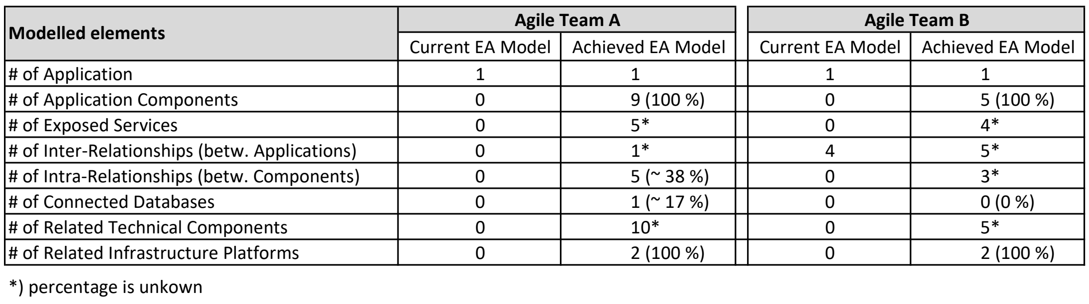


Figure 6.11.: Consolidation of preexisting and achieved EA models

Without the analysis of distributed tracing data, communication flows among application components and to surrounding applications cannot be fully discovered except for the subset which passes the API gateway. Simply speaking, *EAD* in its current state is capable to white-box the preexisting application-level EA documentation by its application components and enrich it by technology layer relationships as well as a certain part of relationships. In the context of the overall *MICROLYZE* concept, tracing data is crucial to fully discover inter-relationships between applications.



Modelled elements	**Agile Team A**		**Agile Team B**	
	Current EA Model	Achieved EA Model	Current EA Model	Achieved EA Model
# of Application	1	1	1	1
# of Application Components	0	9 (100 %)	0	5 (100 %)
# of Exposed Services	0	5*	0	4*
# of Inter-Relationships (betw. Applications)	0	1*	4	5*
# of Intra-Relationships (betw. Components)	0	5 (~ 38 %)	0	3*
# of Connected Databases	0	1 (~ 17 %)	0	0 (0 %)
# of Related Technical Components	0	10*	0	5*
# of Related Infrastructure Platforms	0	2 (100 %)	0	2 (100 %)

*) percentage is unkown

Figure 6.12.: Modeled EA elements - preexisting vs. achieved EA documentation

6.6. Analysis of cost and savings

As discussed by [22] it is not reasonable to judge an IT investment from a purely monetary viewpoint as the value propositions often cannot be directly mapped to financial impact. Instead IT systems often have an indirect influence which can hardly be measured. Besides, non-monetary aspects have to be considered.

This section discusses a business case which was calculated with the industry partner. The first subsection estimates the costs it takes to implement, roll-out and operate *EAD*. The second subsection estimates the current expenditures for manual modeling efforts and derives the cost savings potential. The third subsection compares the determined investment against the estimated cost savings. This results in a net benefit from a monetary perspective only. Other value propositions of the suggested solution are not monetized in this work as they cannot be directly related to costs and efforts. Still, such factors have to be considered on top of the identified financial net value. A final judgment is left to experts in the field and is a part of the evaluation interviews which results are presented in section 6.7. However, this section aims to make the financial impacts transparent in order to facilitate such a judgment.

6.6.1. Running cost and savings for manual EA modeling

The following steps were undertaken to determine the cost savings potential

1. Determine quantity structure and monthly amount of *create* and *update* activities in the EAM repository
2. Calculate (where possible) or approximate the documentation gap of EA elements
3. Conduct expert estimations regarding manual modeling efforts per EA element using the three-point-estimation method
4. Validate estimations with experts
5. Calculate the savings potential by the help of the EAMM automation degrees determined in section 6.4.3

First, the current quantity structure of existing elements in the EAM repository was identified, describing the total amount of elements per type, the amount of attributes per element type and the number of relationships that connects them. Moreover, the monthly average of manual modeling activities was determined as the average of *create* and *update* activities in 2019. Previous years were ignored as there were many bulk activities that would have falsified the value. These quantities serve to determine manual efforts caused by the current data stock by linear extrapolation (cf. Table A.5).

Second, the estimated gap of elements not yet documented in the EAM repository was analyzed. To achieve this, extracts from the CMDB, the API gateway and the cloud platforms were used to get the total number of applications, application components and services. This was compared to the current data stock in the EAM repository. From this comparison the degree of completeness of the current data stock was derived. Further details about this analysis are described in section 6.4. These quantities serve to determine manual efforts that would be caused by closing the documentation gap (onetime efforts) and maintaining the additional data volume (running efforts).

Third, the effort estimations were conducted independently with three Enterprise and Domain Architects using the three-point-estimation methodology described in section 5.2.3. For each type of EA element estimations were recorded differentiated in *create* and *update* activities for a single unit. This estimations include the time necessary to look up information and fill the attributes as well as setting of relationships to other existing EA elements. Explicitly not contained in these estimations are efforts which are needed to obtain information by the means of meetings, calls or email based communication due to the immense variance this would cause. Such efforts are considered in form of a percentual supplement. This process resulted in a PERT estimation value E as per three-point-estimation that is required to create or update a given EA element. Multiplied by the quantities determined in the first two steps, it is possible to calculate the expected manual modeling efforts per EA elements and time period.

Forth, the resulting estimations were cross validated by the experts.

Fifth, after calculating the total efforts by multiplying the quantities determined in the first two steps by the estimated efforts, the saving potential can be derived. This was achieved by multiplying the efforts by the determined automation degree in 6.4.3 on element level as well as on attribute level.

After supplementing the total efforts with 10 percent surcharge to approximate meeting and alignment efforts the total effort and savings potential is determined. The most important figures that were revealed by this calculation are depicted in table 6.8. In the appendix, the corresponding calculations can be found:

- Table A.2 summarizes the effort estimation process. It contains the averaged three-point-estimations obtained by three experts (red colored figures) and calculates the PERT value E from these estimations. E is then multiplied by the monthly quantities to determine the total monthly effort per element shown at the left-hand side of the table. The right-hand side of the table first states the degree of automation that is then multiplied by the total PERT values *E* for *create* and *update* activities.

- Table A.3 is concerned with cost and savings that emerge from the current data stock. It starts with the sum of monthly efforts determined and multiplies them by an assumed internal cost rate of 100 EUR. By applying the degree of automation determined onto the yearly costs, the savings potential is calculated and finally expressed as a percentage.
- Table A.4 is concerned with cost and savings that emerge the EA documentation gap to be closed. On the left-hand side, onetime costs and savings are displayed that would occur if the documentation gap was closed manually. The right-hand side covers cost and savings that would emerge from the additional data in stock.

		current data stock	documentation gap	total data stock
		PERT SUM	PERT SUM	PERT SUM
COST	monthly efforts in hours	39,96	16,69	56,65
	monthly Cost in EUR	3.995,56 €	1.669,30 €	5.664,86 €
	yearly Cost in EUR	47.946,70 €	20.031,63 €	67.978,33 €
SAVINGS	monthly savings in hours	24,40	8,57	32,96
	monthly savings in EUR	2.439,69 €	856,72 €	3.296,40 €
	yearly savings in EUR	29.276,22 €	10.280,62 €	39.556,84 €
		PERT SUM	PERT SUM	PERT SUM
	Savings Potenial	61%	51%	58%

Table 6.8.: Total operation savings

Important remarks and assumptions

1. Monthly average modeling activities are considered to be steady in this calculation as no prediction about the future is possible.
2. The monthly update rate does not differentiate between application classifications, i.e. applications, application components and services are treated identically.
3. The efforts for *updates* for *application components* were considerably lowered as compared to *applications* due to the fact that most attributes for this element type are sufficient to be filled on a application level. Therefore fewer attributes

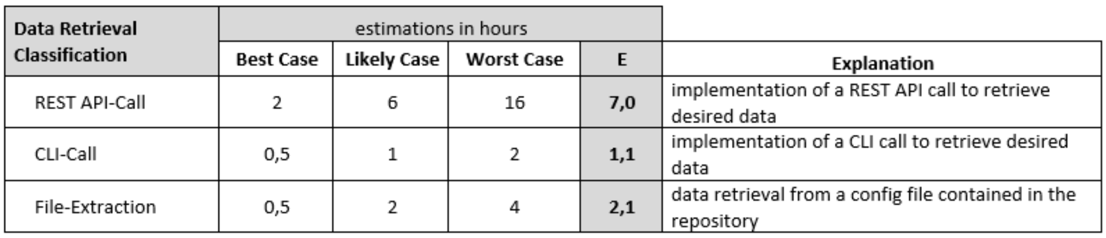
are necessary for application components which results in a higher degree of automation in these cases but also in considerably less saving potential.

6.6.2. Implementation, roll-out and running cost for *EAD*

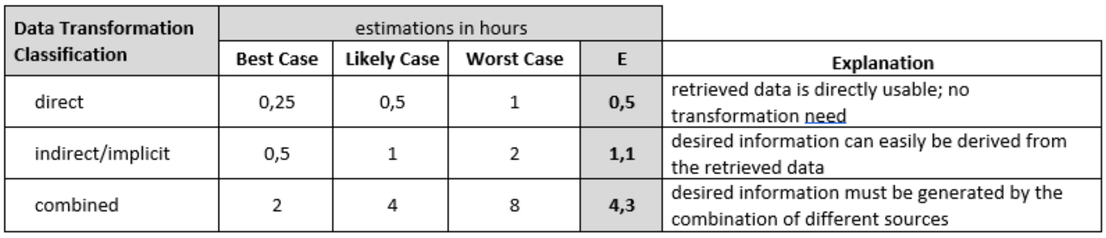
In this section, the cost for the solution's implementation, roll-out and operation are determined. The following aspects are taken into consideration:

1. Preparation efforts (concept, planning and documentation)
2. Implementation efforts (solution implementation, integration with EAM repository)
3. Cross-cutting activities such as project management, enabling and quality assurance
4. Roll-out efforts (*ead.json* creation, pipeline integration and troubleshooting)
5. Operation and maintenance efforts for the suggested solution

The solution's **Implementation efforts** are estimated using the following schema: First, the approach of how relevant data is retrieved from a given data source is determined and classified in a set of predefined retrieval methods. Next, the approach of how data is transformed into information that is actually valuable for EA documentation is determined. Again this is classified in a set of predefined transformation methods. The classification sets were derived from practical experience collected during the implementation of the prototype. Two experienced software developers were asked to give their efforts estimation for each of these classifications using the three-point-estimation methodology. For instance, the developers estimated the effort in hours it takes to implement an REST-API call (Data Retrieval Classification) and to transform the retrieved data into useful information (Data Transformation Classification). Figure 6.9 shows the result of this process. The expected effort E is the actual estimation used for further processing. It is determined as per three-point-estimation technique. The total effort is determined by multiplying the expected effort E by the number of occurrences of the respective classification in the solution's source code. For instance: In total two CLI-Calls and three API-Calls need to be implemented to get the most valuable data out of Cloud Foundry where in nine cases the data returned can directly be used, a single case that requires mapping logic (implicit/indirect) and two cases where the results have to be combined with data originating from a secondary information source (e.g. matching Cloud Foundry applications with related API proxies published in ApiGee). These numbers are partly directly observable from the prototypes source code. For all other cases, the numbers were determined by analyzing the API documentation (parts of GitHub and ApiGee). The values for Kubernetes are assumed to be similar to the ones of Cloud Foundry. Initial project ramp up, concept, planning and documentation was included as a lump sum estimated by the software developers. Efforts for testing and integration with the central EAM repository calculated as an estimated lump sum per source.



Data Retrieval Classification	estimations in hours				
	Best Case	Likely Case	Worst Case	E	Explanation
REST API-Call	2	6	16	**7,0**	implementation of a REST API call to retrieve desired data
CLI-Call	0,5	1	2	**1,1**	implementation of a CLI call to retrieve desired data
File-Extraction	0,5	2	4	**2,1**	data retrieval from a config file contained in the repository



Data Transformation Classification	estimations in hours				
	Best Case	Likely Case	Worst Case	E	Explanation
direct	0,25	0,5	1	**0,5**	retrieved data is directly usable; no transformation need
indirect/implicit	0,5	1	2	**1,1**	desired information can easily be derived from the retrieved data
combined	2	4	8	**4,3**	desired information must be generated by the combination of different sources

Table 6.9.: Data retrieval and transformation classifications and effort estimations

Cross-cutting activities are included as percentual supplement applied to the implementation costs. These are five percent for project management activities during the roll-out, ten percent for quality assurance and ten percent for enabling of and communication with agile teams. **Roll-out costs** essentially consist of onetime efforts it takes for agile teams to create the *ead.json* for each of their artifacts and to integrate the *ead-library* into the deployment pipelines. The first adoption per team is estimated to take two hours and each further adoptions half an hour. Additionally, one hour of troubleshooting per involved team is calculated. Running costs for **Operation and maintenance** were estimated to be one working day per month plus a small amount of infrastructure costs for the *ead-backend* component.

Figure 6.10 summarizes all estimated onetime and running cost positions. Onetime costs in total accumulate to more than 210,000 EUR. Implementation efforts only cause a quarter of this amount. Key cost drivers are the number of application components and deployment pipelines to be integrated. These activities make up to 68 percent of the overall cost.

6.6.3. Total cost of ownership

Concluding the monetary assessment the figures are brought into context of time. Figure 6.13 shows the timely evolution of costs and savings. The lines depict running yearly costs/savings whereas the bars cover onetime effects and the progressing amortization. Year t_0 reflects the current state, without the suggested solution in place. Yearly running costs for manual EA modeling are approximately 50,000 EUR as determined in section 6.6.1 based the current data stock of approximately 2,000 application components (i.e. approximately 25,- EUR per unit and year). In year t_1 the solution is implemented and rolled out (onetime effect costs as per section 6.6.2). It is assumed that only a little portion of the identified savings potential can be realized during the first year. Instead, run costs will first increase by the amount of operation costs of the solution. Realized savings will then start to increase until year t_3 while the amount of registered application components in the EAM repository's data stock will grow up to approximately 4,000 elements as the documentation gap is closed. During the first year upon the invest, a large amount of onetime savings is realized. This essentially reflects the value it would take if the documentation gap was closed on a manual basis (see section 6.6.1). Starting from year t_2, run costs are decreasing due to increased savings and stabilize itself with a reduction of 3.4 percent as compared to year t_0. While this is an almost neglectable amount, the run costs per registered application component in the EAM data stock, however, is reduced by 52.2 percent (approximately 10,- EUR per unit and year). The suggested solution is expected to be fully amortized in year t_5.

The so called *Total Cost of Ownership* (TCO) is a concept that was developed and made popular by the Gartner Group. It targets at determining the actual cost that emerge from an IT investment across its entire life-cycle from acquisition until decommissioning [6], [22]. Figure 6.14 concludes this section by illustrating the total cost of ownership over a runtime of five years. In this case, the solution's decommissioning cost are omitted. From a monetary perspective, after five years runtime the solution is fully amortized. As already argued at the beginning of this section, a purely monetary view is not reasonable when evaluating IT investments. According to Krcmar, value proposition potentials of IT systems can include *automation*, *information provision*, *continuous measurements and surveillance*, *complex analytics*, *overcoming of geographical barriers*, *integration of heterogeneous activities*, *knowledge creation* and *facilitation* of processes [22]. Many of these potentials are also valid for the suggested solution. The following lists a set of additional value propositions that were identified during interview sessions.

- Avoid manual reviews/rework: As the data quality is at an insufficient level, EAM repository-based reports currently often need manual reviews and in some

		Effot in hours	Cost in €	Percentage
IMPLEMENTATION	**GRAND TOTAL - IMPLEMENTATION**	**557,00**	**55.700,00 €**	**26%**
	TOTAL - Preparation	**50,67**	**5.066,67 €**	**9%**
	RampUp / Documentation	25,33	2.533,33 €	50%
	Concept / Planning	25,33	2.533,33 €	50%
	TOTAL Cost - EAM Integration	**52,00**	**5.200,00 €**	**9%**
	Backend Extension	0,00	- €	0%
	Iteraplan Integration	52,00	5.200,00 €	100%
	TOTAL Cost - EA Source Integration	**454,33**	**45.433,33 €**	**82%**
	CloudFoundry	37,79	3.779,17 €	8%
	Kubernetes	37,79	3.779,17 €	8%
	ApiGee	31,29	3.129,17 €	7%
	Github	19,42	1.941,67 €	4%
	Jenkins	11,79	1.179,17 €	3%
	ead.json	38,92	3.891,67 €	9%
	Testing	277,33	27.733,33 €	61%
ROLL-OUT	**TOTAL Cost - Roll-out**	**1460,00**	**146.000,00 €**	**68%**
	ead.json creation	1080,00	108.000,00 €	74%
	pipeline integration	328,00	32.800,00 €	22%
	troubeshooting	52,00	5.200,00 €	4%
CROSS CUTTING	**TOTAL Cost - Cross Cutting Activities**	**139,25**	**13.925,00 €**	**6%**
	Project Management	27,85	2.785,00 €	20%
	Quality Assurance	55,70	5.570,00 €	40%
	Communication / Information	55,70	5.570,00 €	40%
	TOTAL ONE TIME COST	**2156,25**	**215.625,00 €**	**100%**

		hours / year	Cost in € / year	PERCENTAGE
RUN COST	**TOTAL Cost - Operation & Maintenance**	**96,00**	**11.400,00 €**	**100%**
	Maitenance	96,00	9.600,00 €	84%
	Infrastructure Operation	n.a.	1.800,00 €	16%
	TOTAL YEARLY RUNNING COST	**96,00**	**11.400,00 €**	**100%**

Table 6.10.: Estimated implementation onetime and running costs for *EAD*

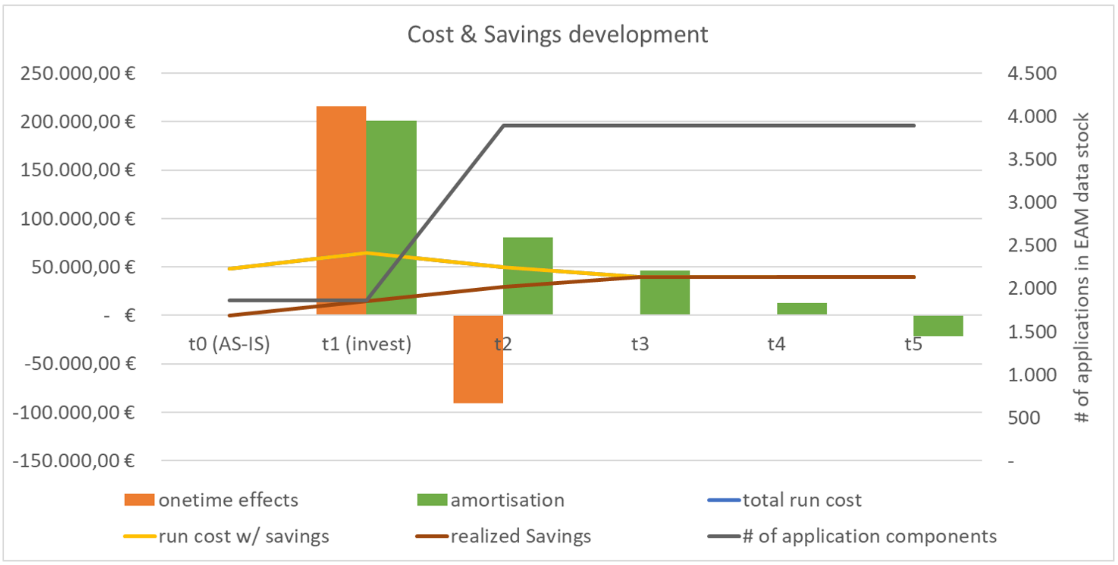


Figure 6.13.: Development of cost and savings

cases manual rework to ensure that the data contained is reliable and correct. The suggested solution can help to increase the data quality to a level that makes manual reviews and reworks unnecessary.

- Avoid unnecessary inventories: During the interview sessions, it was claimed, that despite continuous EA modeling efforts, there are still spreadsheet based inventories as the trust in the current data stock is insufficient. For the scope of automation, such inventories can be avoided.

- Facilitation of projects and decision making: While the documentation gap is closed and EA model maintenance becomes more and more automated, the overall reliability and data quality in the EAM repository will increase. Projects and decisions that rely on this information provided can progress faster.

- Enabling of new use cases: Once being rolled out, it is feasible to extend the functionality as the connection to important information sources is already established. For instance, architectural KPIs could be calculated based upon runtime data or architectural assessments (currently conducted manually) automated.

All in all, such additional non-monetized value has to be incorporated as illustrated in Figure 6.14. Expert's judgment whether the solution is worth being rolled out from the industry partners perspective is given as part of the evaluation interviews which results are described in the following section.

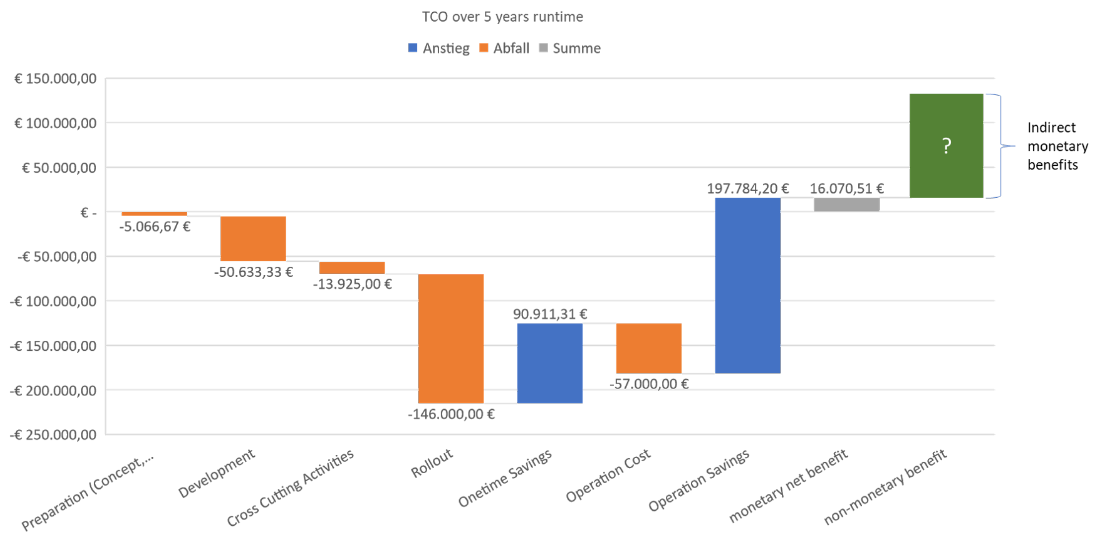


Figure 6.14.: Total cost of ownership over a five years runtime

Digression: Generic formulas for the approximation of cost and amortization

As the calculations in the previous section are specific to the industry partner, this section provides the basic formulas that can be used to approximate the roll-out cost and amortization of the suggested solution in general.

The first formula 6.2 approximates the total cost for implementation and roll-out C. The implementation costs I are considered fixed and have to be determined by experts estimation (cf. section 6.6.2). The roll-out cost R are driven by the total amount of *ead.json* files q_{apps} to be created, the number of deployment pipelines q_{pipes} to be instrumented with the *ead-library* and the number of involved agile teams q_{teams}. For each agile team, the first adoption (i.e. first ead.json to be created $c_{a_{1st}}$ and first pipeline to be instrumented $c_{p_{1st}}$ is assumed to take more time than succeeding adoptions that become faster due to learning effects ($c_{a_{2nd,..,n}}$ and $c_{p_{2nd,..,n}}$). Per team, a certain cost c_t for resolving issues is calculated in addition as a risk factor. By adapting the variables in the formula 6.2, the total costs for implementation and roll-out can be determined depending on the size of variables chosen.

$$\begin{aligned} C = I + & \underbrace{\left(\left(q_{pipes} - q_{teams}\right) \times c_{p_{2nd,..,n}} + q_{teams} \times c_{p_{1st}}\right)}_{\text{integration cost for pipelines}} \\ + & \underbrace{\left(\left(q_{apps} - q_{teams}\right) \times c_{a_{2nd,..,n}} + q_{teams} \times c_{a_{1st}}\right)}_{\text{integration cost for \textit{ead.json} files}} \\ + & \underbrace{\left(q_{teams} \times c_t\right)}_{\text{troubleshooting cost per team}} \end{aligned} \tag{6.2}$$

where:
C = Total one-time cost including roll-out
I = Implementation Cost (Preparation, Development, Cross Cutting Activities)
q_{pipes} = total number of pipelines to be instrumented
q_{apps} = total number of apps to be equipped with *ead.json* files
q_{teams} = total number of agile teams involved
$c_{a_{1st}}$ = monetary effort for first *ead.json* creation
$c_{a_{2nd,..,n}}$ = monetary effort for second to all further *ead.json* creation
$c_{p_{1st}}$ = monetary effort for first *ead-library* integration
$c_{p_{2nd,..,n}}$ = monetary effort for second to all further *ead-library* integrations
c_t = monetary effort per team for troubleshooting

Formula 6.3 allows to determine the remaining value after y years of amortization. The total cost C determined previously is deducted by the onetime savings $S_{onetime}$ that are realized due to automated closure of the EA documentation gap and the sum of generated operating savings across all elements and y years of runtime. Quantities and individual savings potential of course cannot be generalized and has to be estimated for a given organization.

$$A = C - S_{onetime} - \underbrace{\sum_{e \in E} \left(y \times S_e \times q_e\right)}_{\text{sum of yearly operation savings per EA model element e}} \tag{6.3}$$

where:
A = the remaining cost to be amortized after y years
C = Total one-time cost including roll-out
$S_{onetime}$ = Total onetime savings due to closing documentation gap
e = EA element e in set of EA Element $e \in E \{e_1, ..., e_n\}$
S_e = operating savings realized for automated maintenance of EA model element e
q_e = total amount of EA model element e in the EAM data stock

y = years of runtime
S_e = operating savings per element e
q_e = total amount of element e

Digression: Alternative scenario including distributed tracing

As previously mentioned, distributed tracing could not be evaluated with the industry partner. This technology has a significant contribution to the overall solutions value proposition as it is allows to discover application relationships based on the analysis of tracing data produced. The degree of automation for the detection of *Services*, *Interfaces* and *Intra-relationships* at the application layer is assumed to be considerably higher than the current state of implementation. Gathering relationship information is currently one of the major effort drivers for manual EA modeling. Therefore, the savings potential for meetings, calls and email communication is considered to be at a higher level. This, in turn, would lead to a higher cost savings and faster amortization. Figure 6.15 approximates the impact of distributed tracing on the development of running cost and savings. The implementation cost that would be caused on top cannot be determined. For this reason the gray colored bars should indicate that there is a certain supplement to the invest and amortization to be considered.

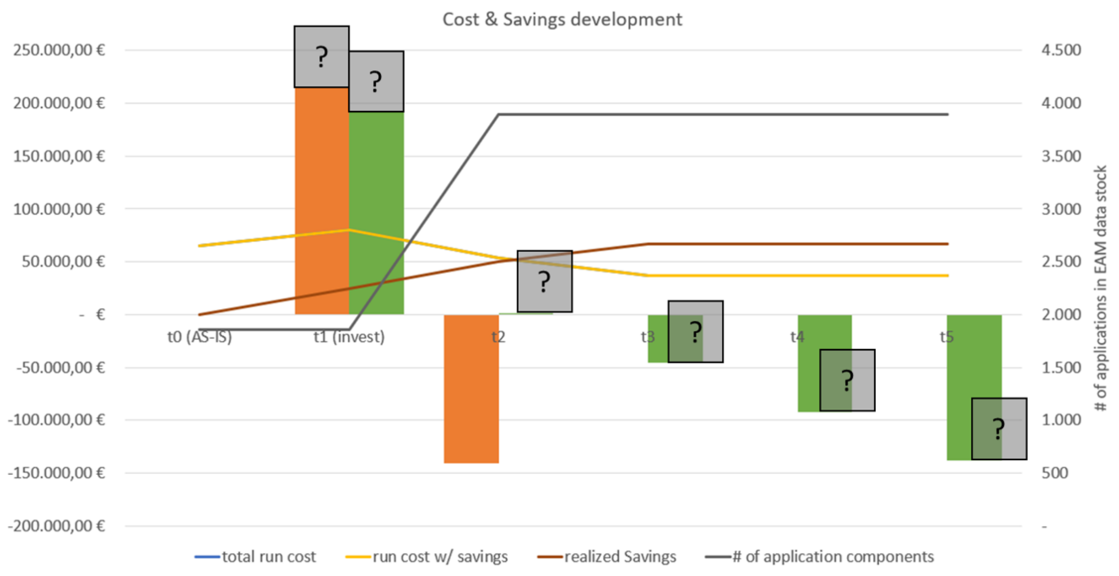


Figure 6.15.: Development of cost and savings including distributed tracing

6.7. Evaluation interview results

At the end of the case study, 14 semi-structured expert interviews including six enterprise architects, four domain architects, two product owners and two software developers ware conducted to obtain qualitative and independent judgment from different perspectives and roles directly affected by the suggested solution.

At the beginning of each session, the interviewees were given an overview of the solution. This included a problem statement, motivation and goals to be achieved. Afterwards, the process of automation of the EA documentation was presented by leading the interviewee through the individual touch points stakeholders encounter while working with the solution. This started with the preconditions which have to be created upfront by agile teams and how the solution affects their daily business (positive and negative). At the end, the visualization of specific examples resulting in the EAM repository were presented. The results were compared to the current EA models. Once the experts have heard and understood the solution in its various aspects, the actual interview started.

In total the interview consisted of 29 questions that were organized along different aspects of the solution. At the beginning, general questions about the concept were asked before details about more specific aspects as the pipeline integration procedure and the *ead.json* were tackled. Experts were required to express their agreement or disagreement with a certain statement on a Likert scale. Additionally, they were given room to explain their choice and add additional feedback in response to open questions. The interviews took between 1.5 to 2.0 hours each. Depending on their role, interviewees were sometimes not able to judge certain questions. The full interview is available in the appendix.

Terminology used: The following specifications correspond to the table 6.11 in section 6.7.6 that aggregates all spoken feedback received.

- **Appreciations:** Key *Appreciation* recorded during the interviews are marked with an unique identifier: $A1, ..., An$
- **Concerns:** Key *Concerns* recorded during the interviews are marked with an unique identifier: $C1, ..., Cn$
- **Limitation:** Key *Limitations* recorded during the interviews are marked with an unique identifier: $L1, ..., Ln$
- **Suggestions:** Key *Suggestions* recorded during the interviews are marked with an unique identifier: $S1, ..., Sn$

These items were compiled based on the analysis of spoken feedback received during the interviews. The items correspond to the most important feedback received repeatedly and independently by interviewees or which is most promising to the further development of the solution.

6.7.1. Evaluation aspect - Solution approach

The first set of questions focused on the overall concept of *MICROLYZE* as presented in chapter 4. Experts were asked to express their opinion if the core foundations of the solution are reasonable and practicable.

Q_1 *The instrumentation of deployment pipelines to drive EA documentation is a practicable and reasonable approach?*: Most of the interviewees fully agreed with this statement (cf. figure 6.16). The highest appreciation was expressed about the fact, that the solution brings the EA documentation process closet to the actual knowledge carriers, the agile teams (cf. Appreciation A1). The idea of using configuration files and pipelines to pass additional information (as suggested by the *ead.json*) was judged as very useful (cf. Appreciation A3). Experts also highlighted that deployment-driven EA documentation would ensure EA updates at the most important points in time (cf. Appreciation A7) as architectural changes usually are introduced along with the deployment process. Two enterprise architects claimed that EA documentation at deployment-time would be too late. They demanded that the solution should consider preceding EA phases in future work (cf. Concern C11). Another enterprise architect feared that the approach could lead to a complex fragmentation of the overall EA documentation process. Therefore a roll-out would require to properly re-define the overall process which clearly defines the roles and responsibilities of different stakeholders and systems (cf. Concern C2).

Q_2 *The discovery of EA elements from runtime data (cloud platforms, API gateways, distributed tracing) is a practicable and reasonable approach?*: All experts either agreed or fully agreed with this statement (cf. figure 6.17). They highlighted that runtime data as an ideal source for the discovery of the as-is IT landscape which at any time reliably reflects the current state (cf. Appreciation A2). Drawbacks mentioned by the experts (and that prevented some of higher agreement) are issues with data granularity which used to be too fine-grained and not matching the one required for EA purposes (cf. Concern C4). Another claim frequently made is that this kind of data often is difficult to interpret from an EA perspective (e.g. due to technical names). Therefore runtime data alone is of little value for the practice as most of the current use cases are driven from the business layer. The combination with the static business-oriented data provided by the *ead.json* configuration file is therefore crucial (cf. Concern C1).

Q_3 *The use of a configuration file (ead.json) to bind static information (business layer rela-*

tionships, links) to an application is a practicable and reasonable approach?: All experts either agreed or fully agreed (cf. figure 6.18). The general idea was well appreciated by the interviewees. They especially highlighted the importance of the configuration file to link business layer information to the artifact but also that this could be further expanded to other use cases as well (cf. Appreciation A3). This adds a high value for EAM as it allows to interpret, filter and report on applications discovered at runtime from a business layer perspective and therefore overcomes aforementioned issues with pure runtime data based EA discovery (lack of interpretability, data granularity issues). However, a potential lack of knowledge about business layer assignments at agile teams is a drawback. Such knowledge would have to be build upfront to ensure correct assignments (cf. Concern C6). For some product owners, the handling of the JSON format based *ead.json* might cause troubles (cf. Concern C5). The idea, to have the configuration file generated by an interactive form, was named numerous times (cf. Suggestion S4). Some of the experts also see a risk in low acceptance of agile teams. However, both interviewed product owners did judge this risk as *minor*. Suggestions for further improvement made by the experts were (1) to bind the information contained in the *ead.json* to the runtime artifact either by the use of tags or environment variables (cf. Suggestion S3), (2) to enhance the automated detection of federated information systems based on static code analysis or extraction from the environment variables (cf. Suggestion S6) and (3) to bring the *ead.json* initialization forward to the EA planning phase and pass it through to the design and implementation phase in order to bridge the gap between these EA life-cycle phases (cf. Suggestion S1). These suggestions are explained in more detail in section 6.7.6.

Q_4 The shift in responsibility for EA documentation to agile teams is a practicable and reasonable approach?: This statement received the highest agreement, no matter the role (cf. figure 6.19). All experts agreed that this is the correct approach and constitutes a necessity. Despite risks regarding acceptance, one exceptional feedback was that the approach brings the EA documentation process closer to the agile teams by integrating into their natural process and tool environment (cf. Appreciation A5). This would lead to a higher acceptance than imposing them to use an EAM repository for modeling. By making them responsible for EA documentation, the current gap between agile teams and EA might be reduced in general and leads to increased understanding at both sides. Experts of all roles pointed out that the documentation responsibility should be as close to the knowledge carriers as possible which is enabled by the suggested approach (cf. Appreciation A1). In contrast, they stated, that this currently is not the case and most of EA modeling is covered by domain architects who have to gather information from agile teams in labor-intensive and error prone work.

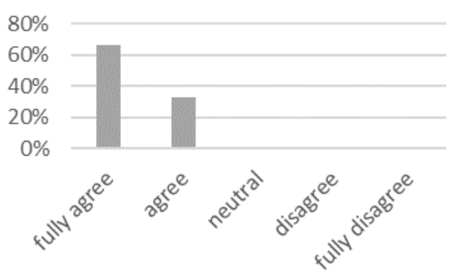


Figure 6.16.: Q_1: The instrumentation of deployment pipelines to drive EA documentation is a practicable and reasonable approach?

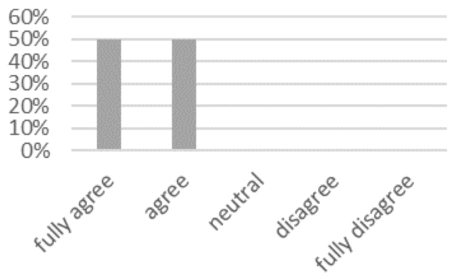


Figure 6.17.: Q_2: The discovery of EA elements from runtime data is a practicable and reasonable approach?

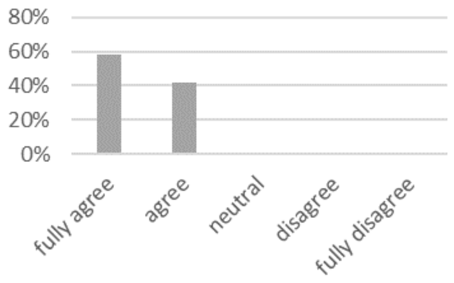


Figure 6.18.: Q_3: The use of a configuration file (ead.json) to link static information ([..]) to an application is a practicable and reasonable approach?

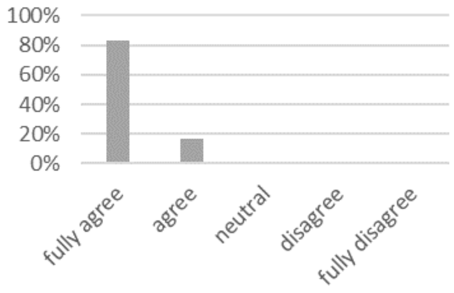


Figure 6.19.: Q_4: The shift in responsibility for EA documentation to agile teams is a practicable and reasonable approach?

Q5 The suggested solution is easy to integrate into the agile development process?: A central requirement of the industry partner is the solution's reasonable integration into the preexisting environment of processes, tools and actors. Additional efforts and potential side effects have to be reduced to a minimum. With regards to the agile development process experts see this requirement as fulfilled (cf. Figure 6.20 and Appreciation A4). All interviewees expressed their agreement or strong agreement. The solutions preconditions are easy to create as part of regular sprint planning and sprint execution processes as also described in section 4.3.1 which was inspired in exchange with the agile teams that adopted the solution. A product owner and a developer appreciated that only a onetime effort is needed. Additionally, the integration procedure is perceived as simple and time-efficient according to the two teams that adopted the solution to their applications (cf. Appreciation A8).

Q6 The suggested solution reasonably integrates into the EAM ecosystem (documentation processes, actors and tools)?: In the context of existing EA documentation processes around the central EAM repository, interviewees were cautious to answer this question. Most experts still agreed, two experts disagreed (cf. figure 6.21). The most mentioned challenge arises from multiple independent information sources which get automatically imported into the EAM repository. Conflicts and duplicates need to be mitigated and therefore proper mechanisms to ensure data consistency are required (cf. Concern C3). One enterprise architect claimed, that automated imports would reduce the flexibility for manual modeling and corrections using the EAM repository's user interface (cf. Concern C7). Attributes that are affected by automatic import would need to be protected against manual changes. Otherwise they would be overwritten by the next import. Corrections using the deployment pipeline, in turn, would lead to high overhead. In opposition, one expert argued that this is not an issue but only a matter of processual change. In his opinion, agile teams have to know what to what business layer elements their application corresponds to. In case such assignments are incorrect in the *ead.json*, agile teams have to be made aware of their mistake and correct it respectively. Some experts claimed that other lifecycle phases are not or poorly considered by the solution (cf. Limitation L1). The solution would concentrate too much on the current state of EA. They did not only demand to integrate further federated systems but suggested to have real incorporation of the EA planning phase from a process perspective (cf. Suggestion S1). Two experts suggested that the solution could even help to establish system-overarching unique identifiers (cf. Suggestion S2). These suggestion are further explained in section 6.7.6.

Q7 The suggested approach will improve the quality of data in the EAM repository?: Data quality can be broken down into *data granularity*, *data completeness*, *data consistency* and

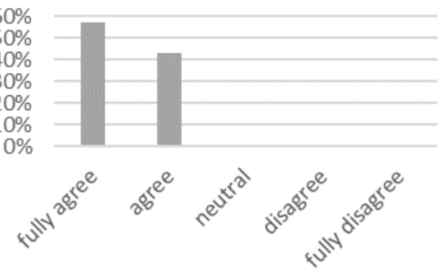


Figure 6.20.: Q_5: The suggested solution is easy to integrate into the agile development process?

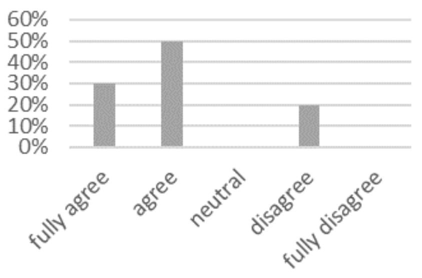


Figure 6.21.: Q_6: The suggested solution reasonably integrates into the EAM ecosystem (documentation processes, actors and tools)?

data actuality. High data quality is a key success factor and therefore a fundamental requirement to EAMM solutions [8]. Experts' judgments go in disperse direction with regards to what kind of impact the suggested solution would have on the data quality (cf. Figure 6.22). Regarding data completeness and data actuality experts are optimistic that the solution has a positive impact. Limitations regarding data completeness were mentioned as a significant part of the industry partner's IT landscape cannot be covered as it resides in legacy data center environments (cf. Limitation L3). Still, there was a high appreciation that cloud-based environments can reliably be covered what indeed is a valuable benefit to the organization. The biggest doubts expressed were regarding data consistency. Certainly, if used on its own, the solution would deliver a very high data quality. Putting it into the current environment where there is still manual modeling in the EAM repository and other independent imports from different sources in parallel, interviewees were skeptical whether an improvement could be achieved. Some expected it to decrease at the beginning and only recover and improve upon a time after processes stabilized. The answers to data reliability as an outcome of the three aforementioned data quality characteristics, in consequence, reflected this skepticism. Most mentioned challenges were (1) the assurance of data consistency and handling of conflicts across acting people and import mechanisms (cf. Concern C3) and (2) incorrect business layer assignments contained in *ead.json* (cf. Concern C10). However, experts did also see opportunities for increasing data quality. They argue that the trust into the data stock could be increased as manual modeling is reduced. Runtime data is generally perceived as highly credible and up-to-date. The result of this could manifest itself in reduced efforts for verification of reports generated from the data stock and time-consuming rework. A current issue to data quality are manual modeling faults where elements are incorrectly modeled and classified. This falsifies filters and reports applied to the EAM repository. Thanks to the suggested solution, the modeling error rate could be reduced and unified modeling of elements ensured due to automation (cf. Appreciation A6).

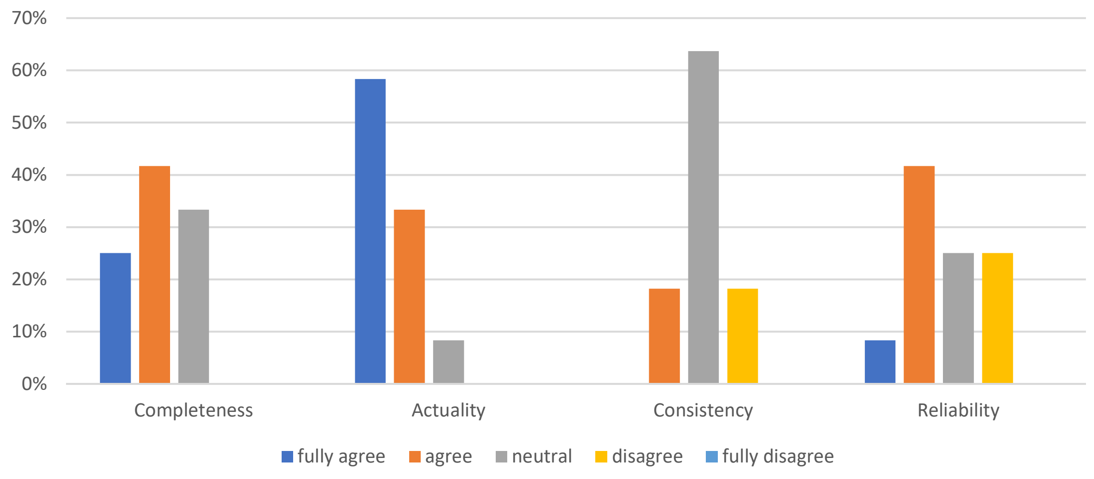


Figure 6.22.: Q_7: The suggested approach will improve the quality of data in the EAM repository?

Q_9 *The effort it takes to get the tool operational is manageable?* and Q_{10} *The effort it takes to get the tool operational pays off quickly?*: At this point, interviewees did know about the required preconditions (pipeline integration procedure described in section 4.4.3 and *ead.json* creation described in section 4.3.1) and were presented results about the cost and savings analysis covered in section 6.6. With this background information the questions about economic feasibility were asked. The majority of experts agreed or fully agreed to these questions (cf. figure 6.23 and figure 6.24). The determined roll-out cost was perceived plausible and low (relatively seen within the industry partner). Still, such a project could not be funded by the EA department on its own (cf. Concern C8). Other sponsors would need to be acquired. However, as the solution is easy to adopt with small amounts of onetime efforts, some experts argued that there is no need to fund a roll-out but only the implementation cost. Onetime efforts to the agile teams are reasonable and could be covered as part of their daily work without too much impact. Agile teams could be convinced about their intrinsic motivation as they can save time for meetings currently required to model the EA in the EAM repository (cf. Suggestion S8). On a mid-term period, they would therefore profit from the solution.

Other interviewees suggested to provide agile teams with standardized deployment pipelines which include the EA documentation logic per default and based on this technically enforce its usage (cf. Suggestion S6). Admittedly, they confirmed, that this is currently not possible due to the numerous independent CI/CD servers that limit the solution's scalability (cf. Concern C12). Section 6.6 resulted in an amortization period between four to five years, solely considering saved cost due to reduced manual modeling efforts. Some experts stated that this might constitute an issue as the organization usually demands a faster payback. They highlighted that the presented analysis did not include other value propositions beside time savings. Therefore, the results are rather pessimistic. They agreed that such beneficial aspects are difficult to monetize, however, they would expect the solution to pay off faster. Examples named for additional savings potential not included in the cost analysis are (1) reduction of manual verification/rework of EAM repository based reports due to higher data reliability, (2) saved efforts for alignment meetings and workshops with the purpose of manual EA modeling and (3) satisfaction of increased demand by running projects with regards to the application component layer that currently is not documented. An important note is that the aforementioned cost and savings analysis as well as experts' judgments consider the current state of implementation at the industry partner, i.e. without the use of tracing data. Almost all experts stated that the future inclusion of tracing, as a vehicle to discover application layer inter-relationships would make the solution from an economic and added-value perspective more attractive (cf. Limitation L4).

Q_{11} *Overall, the suggested approach is capable to reduce the amount of manual documentation effort?*: This questions was asked with regards to the (1) current state of implementation (i.e. without tracing data) (Q_{11a})and the (2) target state of implementation that includes tracing data (Q_{11b}). The key difference is that the current state only covers parts of application inter-relationships whereas distributed tracing could make all interrelationships transparent. With regards to the target state, all of the experts expressed their full agreement, as it currently makes one of the most time consuming factors to find out about applications communication relationships (cf. figure 6.26). So far, domain architects take over most EA modeling activities in the EAM repository. As they do not know about the details on an application layer it is necessary to request and clarify this information together with the responsible agile team. Meetings, calls and emails undertaken for this purpose are very time consuming and bind several resources at once. Such activities could be saved to a major extent by the help of automated relationship modeling based on tracing data. Development teams and domain architects would most profit from this situation. Two experts don't believe in distributed tracing to be the right instrument as it would cause too much costs and negatively impacts applications' performance if enabled continuously (cf. Concern C9). Instead, they promote the use of cloud native solutions to solve this challenge (cf. Suggestion S7). Product owners would profit to a little portion only as they still have to model their application in the CMDB in parallel. With regards to the current state of implementation, i.e. without tracing data available, interviewees still perceive the solution as valuable as explained in the previous questions, however, they doubt that this results in noticeable savings of modeling efforts (cf. figure 6.25). The main reason for this perception is that so far it was a conscious decision not to model the application component level due to additional extraordinary efforts that would be caused. A major part of time savings as calculated in 6.6 would therefore actually be imputed savings as the costs currently do not materialize.

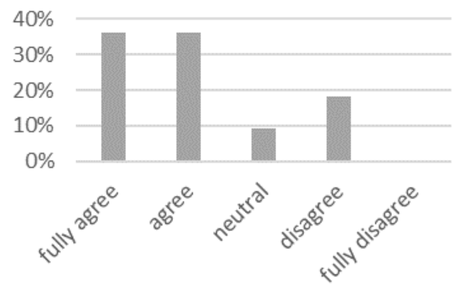


Figure 6.23.: Q_9: The effort it takes to get the tool operational is manageable?

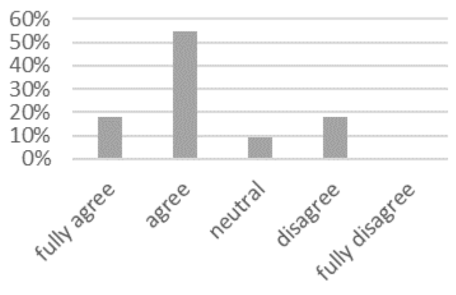


Figure 6.24.: Q_{10}: The effort it takes to get the tool operational pays off quickly?

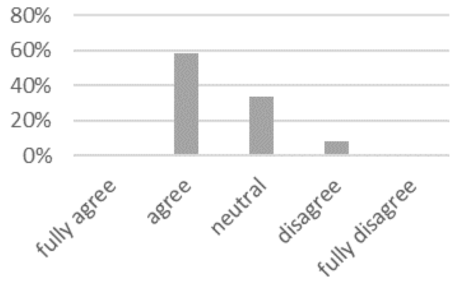


Figure 6.25.: Q_{11a}: Overall, the suggested approach is capable to reduce the amount of manual documentation effort? (excl. tracing)

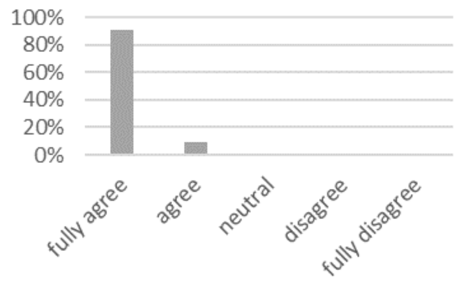


Figure 6.26.: Q_{11b}: Overall, the suggested approach is capable to reduce the amount of manual documentation effort? (incl. tracing)

Q_{12} *The suggested approach supports the strategy of moving legacy applications to cloud* *based environments?* and Q_{13} *The suggested approach is capable of being adapted to newly introduced technologies?*: With regards to Q_{12} all experts fully agreed or agreed (cf. figure 6.27). According to them, the suggested solution has a good fit to current strategies pursued. On the one hand side, cloud migration projects are still ongoing which leads to an increasing volume of IT landscape which can be handled by the help of the suggested solution. New applications are build per default on cloud environments. On the other hand, a major project regarding the SOA transformation has been launched. Its goals include to publish and manage all internal and external services on central API gateways. Both projects will therefore implicitly result in a higher value of the suggested solution as the importance of cloud platforms and API Gateway (the two core information sources to the suggested solution) is about to further increase. Regarding Q_{13}, the majority of interviewees agreed, that the solution is adaptable to technologies newly introduced (cf. figure 6.28). CI/CD pipelines are in use independently of the targeted cloud platform. The same applies to distributed tracing which today is already supported by most of the cloud platforms.

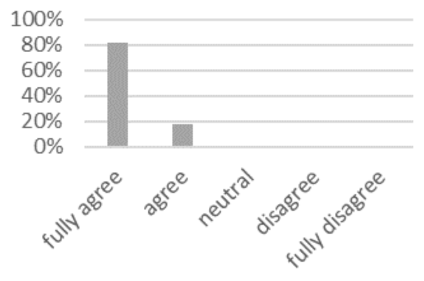


Figure 6.27.: Q_{12}: The suggested approach supports the strategy of moving legacy applications to cloud based environments?

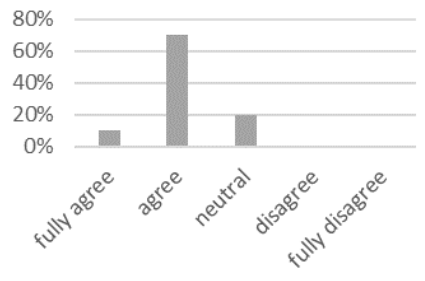


Figure 6.28.: Q_{13}: The suggested approach is capable of being adapted to newly introduced technologies?

A central idea of the suggested solution is to enable a federated EA strategy, by passing references to other systems which contain information about a given artifact along with its *ead.json* file. From an EAM perspective, this allows the retrieve additional information in real-time and on demand from these systems as described by [20]. Question Q_{14} (*The approach could enable/drive new EAM use cases. Which of the following suggestions are most promising/interesting to you?*) therefore asked experts to give their opinion to suggestions including (1) EAM KPIs retrieved from source system (e.g. complexity, performance or code quality metrics), (2) Architecture Evolution (i.e. visual representation of historic EA model evolution), (3) automation of Architecture Assessments (e.g. assessment of an applications operability, cloud readiness or technical debts) and (4) automated architectural guideline compliance checks. Responses recorded about this question are dispersed (cf. figure 6.29). Most agreement was attributed to the automatic calculation of EAM KPIs and architecture assessments. Architecture evolution and guideline compliance was perceived as less promising. Some experts claimed that architecture assessments and guideline compliance actually are interesting use cases. However, automation is only partially feasible as qualitative judgment by humans is necessary in many cases.

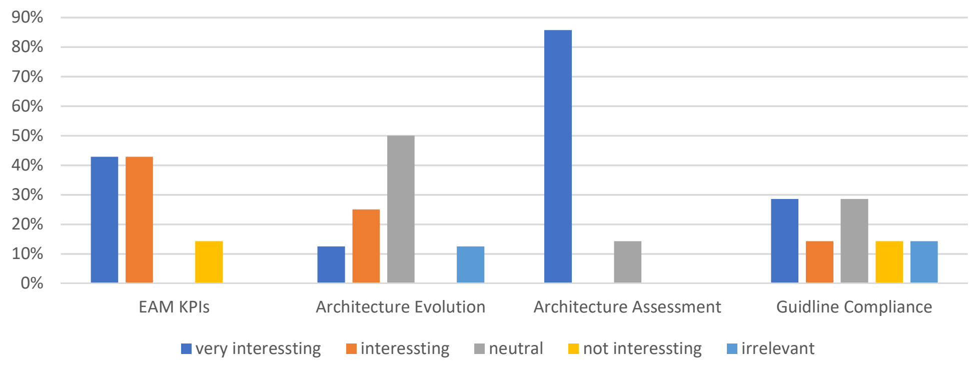


Figure 6.29.: Q_{14}: Which of the following suggestions [for enabled use cases] are most promising/interesting to you?

Q_{15} *How do you judge the severity of observed risks in the context of your company?* target at potential negative side effects of the proposed solution. Experts should judge the severity of a set of observed risks during the case study or name own concerns. Listed risks were:

1. Challenge of *Consolidation* of independent data sources delivering into the EAM repository
2. Challenge of *Conflict handling* between manual and imported data from source systems into the EAM repository
3. Too much *Overhead*, this means additional efforts caused to affected actors due to the suggested solution
4. Too much impact on deployment pipelines *Performance* caused due to the suggested solution

5. Potential *Security* caused due to the suggested solution
6. Lack of *Acceptance* with product owners (PO)
7. Lack of *Acceptance* with the development team (DEV)
8. Risk of *outdated ead.json* content
9. Risk of *inconsistent ead.json* content, i.e. mismatch of information across components of an application

Risks perceived most severe are consolidation and conflict handling of data sources which independently deliver into the EAM repository. All other risks were attributed to have a medium severity, except for security with a minor severity judgment (cf. figure 6.30). Experts recommended that acceptance with agile teams (product owners and developers) needs to be achieved by explaining the motivation and value of EAM. They should understand, they had an intrinsic interest in the solution and they can directly profit from its usage. First, they would benefit in saving time for meetings, calls and email currently needed for manual modeling of an application. Second, the solution supports bridging the current gap between EA models which use a very high level language and degree of abstraction and the application components that actually underlie these models and are developed by the team. As a result, better collaboration could be enabled.

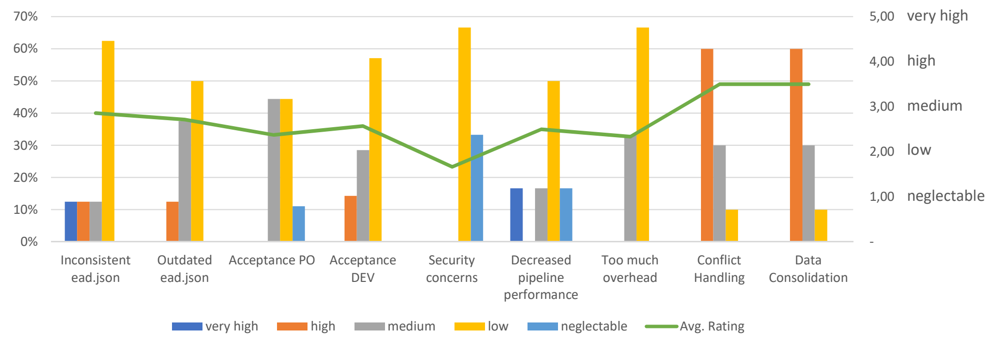


Figure 6.30.: Q_{15}: How do you judge the severity of observed risks in the context of your company?

6.7.2. Evaluation aspect - Pipeline integration

Interviewees with a software developer knowledge background (eight experts) were presented a dedicated section regarding the concept of pipeline integration in order to get more detailed feedback.

Q_{17} *The EAD-library can be integrated with reasonable effort?*: The first question was meant to find out whether the solution can be integrated with reasonable and acceptable effort. The majority of interviewees fully agreed, some agreed. There was no disagreement recorded. As already mentioned in section 6.5, the two teams that actually adopted and tested the solution, perceived its complexity as *low* or *very low*. They fully agreed that the required effort is at an acceptable level, especially compared to other existing governance regulations which require pipeline integration (cf. Appreciation A8). This was also confirmed by the answers received to Q_{17}, where all experts agreed and the majority fully agreed (cf. figure 6.31). Still, some suggestions for further facilitation were pointed out, e.g. to provide more detailed examples in the procedure's description and reduce the number of required method parameters. As mentioned at the beginning of this section, developers at the industry partner are very familiar with the concept and usage of Jenkins Shared Libraries. At the current state of implementation, the EA documentation stage executed in deployment pipelines takes approximately 1.0 to 1.25 minutes. Q_{18} asked, what duration would be acceptable to agile teams. One product owner had little willingness to accept more than 0.5 minutes of extra duration as productive deployments usually take much time even without the EA documentation logic. Other experts, in turn, stated that the duration should be kept as little as possible. However, no negative impact would emerge from the current duration. One interviewee suggested to separate the EA documentation logic into a dedicated CI/CD server job, that should be triggered from the origin deployment pipeline. Doing so would lead to no additional waiting time at all, it would however limit other functional aspects of the solution.

Q_{19} asked the experts to judge the severity of a list of risks and challenges that goes along with using pipeline integration with regards to (1) **Security**, (2) **Pipeline Performance**, (3) **Acceptance** by developers and (4) **Overhead** caused by this solution. Answers in the majority of cases resulted in a *low* to *neglectable* severity. Some interviewees attributed a *medium* severity to a potential lack of acceptance and too much overhead caused and in a single case *high* severity for decreased pipeline performance (cf. figure 6.30).

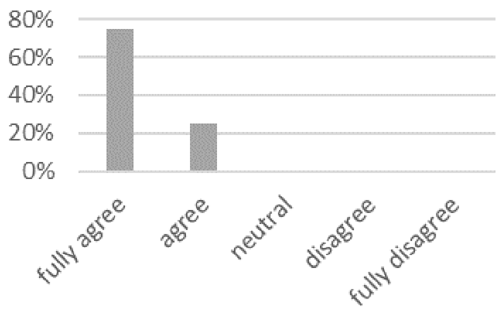


Figure 6.31.: Q_{17}: The EAD-library can be integrated with reasonable effort?

6.7.3. Evaluation aspect - ead.json

One section of the evaluation interview was dedicated to the concept of using a configuration file (*ead.json*) to link artifacts with business layer elements and references to federated information sources. As previous questions already revealed, experts do highly value this approach (cf. Appreciation A3 and Concern C1). Answers got more disperse as they were asked whether the source code repository would be a more convenient place to provide this information (contained in *ead.json* configuration file) than the EAM repository (Q_{20}). A slight majority fully supported this idea as software artifacts become directly associated with business layer relationships instead of having this information maintained decoupled in an EAM repository (cf. figure 6.32). Some experts even would go a step further. They suggest binding this information directly to the runtime artifact e.g. as environment variables or tags (cf. Suggestion S3). Doing so would allow to retrieve this information at any time from the artifact's runtime environment after deployment. Pipeline integration allows to technically achieve this. A similar use case based on this idea is already in use at the industry partner and proved its practicability. The remaining interviewees had a rather neutral and a single one a negative position about this idea. They see a lack of usability and reduced flexibility as compared to manual modeling in the EAM repository. Using the suggested solution would require to prohibit manual business layer assignments in the EAM repositories user interface due to conflicts and overriding situations by automated and manual modeling activities.

Q_{21} *What additional EA information sources should be included (as of now: Github, Jenkins, Jira, ApiGee)?*: Interviewees were also asked what kind of information actually should be included in the *ead.json*. As already mentioned a few times, experts highly valued the idea including business layer assignments in the *ead.json*. However, this should be limited to the almost static ones. As of now this includes *Business Domains, Products, Business Processes, Business Objects and Business Units*. **Business Capabilities** currently experience many changes so that they should be skipped from the configuration file. Regarding federated information systems, one expert demanded to assess all tools along the application lifecycle whether they can be integrated. This is not always an easy task, as there is no direct mapping between an application component and elements contained in other systems in many cases. For instance, during project portfolio planning it is not yet clear of what components an application will consist of. In CMDBs, application components might not be represented as well. Other tools might only register applications but not components. Therefore an individual assessment is necessary for a given tool. Other tools demanded, that reflect individual application components are Continuous Code Inspections Tools (e.g. SonarQube, Selenium), Binary Repository Managers (e.g. Artifactory, Docker Hub) and APM systems (e.g. Grafana, Prometheus). One expert suggested that dependencies towards systems outside the technical scope of the suggested solution (e.g. legacy systems) could be included in the *ead.json* file.

Two questions tackled the issue whether the configuration file can be created (Q_{22}) and maintained (Q_{23}) with reasonable effort. All experts agreed or fully agreed that the effort needed for the initial creation of the file is reasonable and acceptable (cf. figure 6.33). A majority thought the same about the maintenance of the information contained as it was almost static. Two interviewees (a product owner and an enterprise architect) were less optimistic and gave a neutral rating.

Q_{24} asked whether the current version of the *ead.json* is understandable to affected roles, responsible for the creation and maintenance of data contained. Experts agreed, however, also made suggestions for further improvement (cf. figure 6.34). Besides a more detailed manual including real-case examples, the idea most often mentioned was to introduce a form-based process for *ead.json* initialization. A web-based form should lead the user through the fields of the configuration file which gets finally generated automatically and provided as a download (cf. Suggestion S4).

Finally, Q_{25} asked the experts to judge the severity of a list of risks and challenges which go along with using the *ead.json* concept with regards to (1) ***ead.json* inconsistencies** that might occur across the configuration files of a set of components building an application together, (2) **outdated *ead.json* content** and (4) **acceptance** by agile teams. Data quality issues with the configuration file were judged with a *low* to *medium* level severity by most experts (cf. figure 6.30). Some expressed more skepticism. Still, this risk was attributed to be manageable as the information contained in the *ead.json* is of

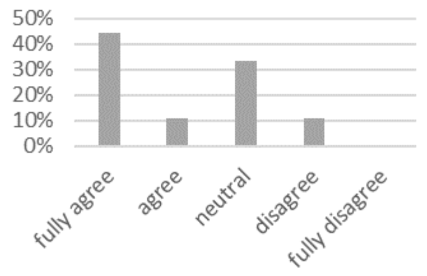


Figure 6.32.: Q_{20}: The code repository is a more convenient place to maintain the information asked by the ead.json file than compared to Iteraplan?

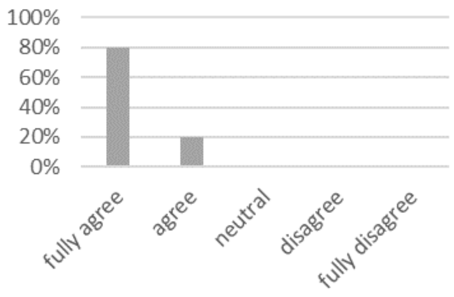


Figure 6.33.: Q_{22}: The ead.json template can be created with reasonable effort?

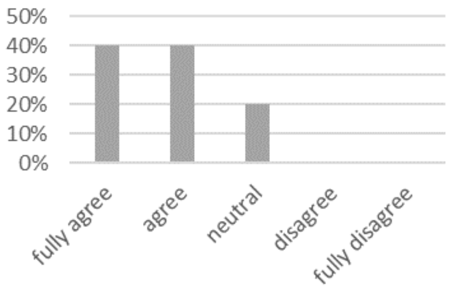


Figure 6.34.: Q_{23}: The ead.json can be maintained with reasonable effort?

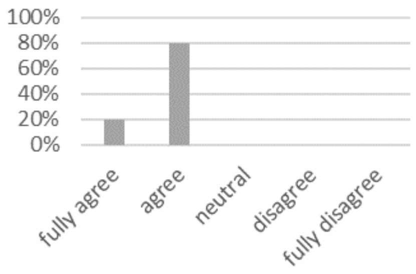


Figure 6.35.: Q_{24}: The ead.json is easy to understand?

very static nature. Figure 6.30 depicts the results.

6.7.4. Evaluation aspect - Documentation coverage

The last section of the interview focused on whether the industry partner's EA information demand can be satisfied by the suggested solution. **Business layer relationships** (Q_{26}) are very well covered by the concept of using configuration files as an addition to software artifacts source code repositories. Most experts therefore were *rather satisfied* to *fully satisfied* (cf. figure 6.36). All essential relationships can be covered at an acceptable and reasonable level of effort for creation and maintenance. A drawback mentioned by some of the experts obviously is that there is no real automated discovery of such information (cf. Limitation L2). They admitted that they currently see no solution to achieve such automation.

Application layer relationships, at the current state of implementation, are partly covered. In general, experts perceive this as insufficient (Q_{27a}, cf. figure 6.37). The reason is that no tracing data is incorporated which would allow discovering communication relationships and derive inter-dependencies among applications and applications components. At the industry partner, distributed tracing technology is not used consistently and therefore is not a reliable data source as of now. Given the case, that this gap is closed, experts in a majority are very satisfied with the solution's capabilities (Q_{27b}, cf. Figure 6.38).

The solution is capable to satisfy most of the experts demand for **technology layer relationships** (Q_{28}, cf. figure 6.39). This includes the cloud platform the application component is operated on, its technical stack and used cloud services (e.g. databases, autoscalers, monitoring agents, etc.) as well as the programming languages used for implementation. Some had no specific interest in this information and therefore took a neutral position.

Regarding the **coverage of attributes** (Q_{29}) for application components the majority of experts stated to be *rather satisfied* (cf. figure 6.40). They appreciated that there is a solid foundation of attributes covered. However, most attributes would remain to be handled manually.

6.7.5. Expert's conclusion and feedback

At the end of the interview, experts were asked, whether they would roll-out the suggested solution and, if applicable, what conditions they would impose (Q_{16} *Would*

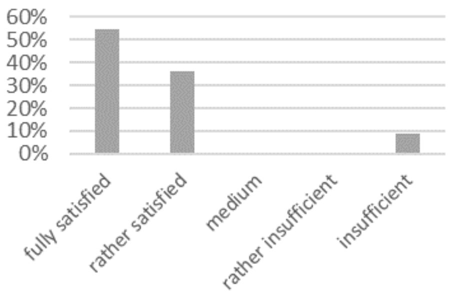


Figure 6.36.: Q_{26}: How satisfying is the coverage of business layer relationships?

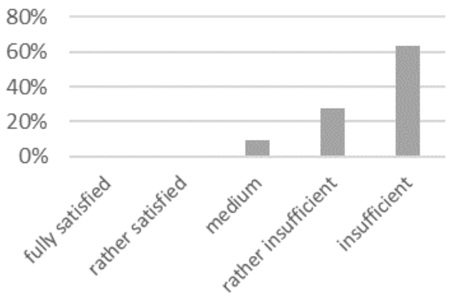


Figure 6.37.: Q_{27a}: How satisfying is the coverage of application layer relationships? (excl. tracing)

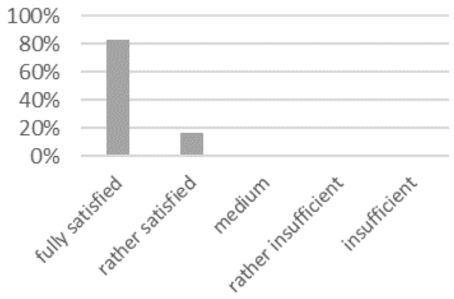


Figure 6.38.: Q_{27b}: How satisfying is the coverage of application layer relationships? (incl. tracing)

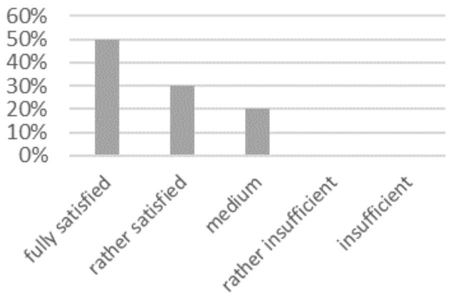


Figure 6.39.: Q_{28}: How satisfying is the coverage of technology layer relationships?

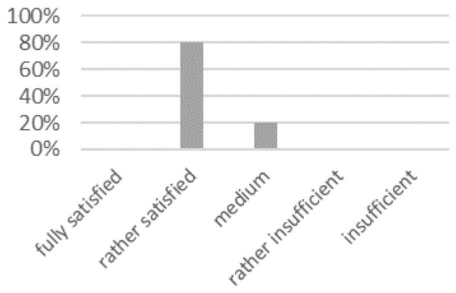


Figure 6.40.: Q_{29}: How satisfying is the coverage of attributes?

you roll-out the suggested solution? and *If not, why? What needs to be fixed?*). With only a single exception all interviewees agreed or fully agreed to roll-out the suggested solution despite its current state of implementation without tracing data integration (cf. Figure 6.41). However, most interviewees pointed out additional conditions to be fulfilled prior to the roll-out or to be caught up on as soon as possible.

1. **Process enforcement:** The adoption of the suggested solution needs to be mandatory, ideally enforced by technical means. A possible solution would be the use of a standardized, pre-configured deployment pipeline which fails as long as the EA documentation step cannot be executed due to missing *ead.json* file or incomplete properties. In such a case the validation of the configuration file must happen prior to the artifacts deployment stage (cf. Suggestion S6).
2. **QA measures:** Prior to the roll-out, further functionality needs to be implemented to resolve data consolidation and conflicts between overlapping data imports by different source systems. This corresponds to the specific situation at the industry partner where multiple tools independently import data into the EAM repository.
3. **Tracing data integration:** The automated detection and modeling of communication relationships between applications and application components is a key value proposition of the solution. Therefore tracing data must be incorporated on a mid-term plan as part of further development.
4. **Extension of the roll-out scope:** In order to increase the solution's profitability, a roll-out should be expanded to a cross-country or even group level, not only within a single country organization.
5. **Extension of the technical scope:** Parts of the current IT landscape are technically out of scope of the solution. To increase its value proposition, the solution's capabilities should be further extended to cover non-cloud environments.
6. **Standardized deployment pipelines:** Prior to the roll-out, distributed CI/CD servers should be consolidated to enable standardization of deployment pipelines which is a precondition to further reduce the pipeline integration efforts and allow technically effective process enforcement (see also 1)
7. **Inclusion of the EA planning phase:** EA elements that have been created during target IT landscape planning must automatically be linkable to discovered EA artifacts in order to avoid duplicates and enable EA life cycle management.

Figure 6.42 summarizes the mentioned conditions for a roll-out of the suggested solution including the number of independent mentions and the percentage of all responses. Most experts explicitly demanded a future integration of tracing data in order to further increase the automation potential and the value proposition of the solution. Process enforcement and consolidation of CI/CD servers are preconditions named second most frequent. The inclusion of the EA planning phase and implementation of reliable quality assurance mechanisms is also a key requirement to some of the experts. Only few experts demand for a organizational (roll-out scope) or technical extension (IT landscape coverage) of the solution's scope. Only a single experts required the EAD stage to be configured non-blocking as the deployment success goes over EA documentation success.

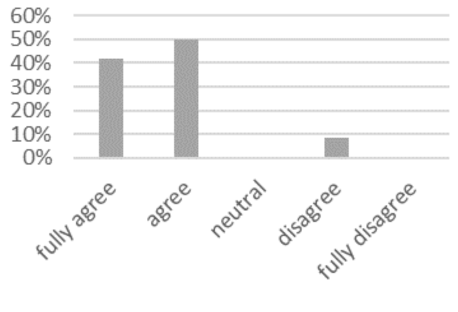


Figure 6.41.: Q_{16}: Would you roll-out the suggested solution?

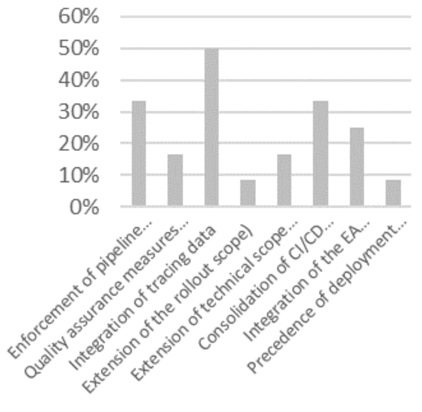


Figure 6.42.: Roll-out preconditions mentioned by interviewed experts

6.7.6. Key findings and suggestions

All feedback received as part of the open question was complied into a set of items classified as *Appreciations*, *Concerns*, *Limitations* and *Suggestions*. Table 6.11 shows the result of this aggregation. *Appreciations* denote aspects of the presented solution approach which were explicitly valued by experts. *Concerns* denote skepticism or doubts expressed regarding a certain topic. *Limitations* denote restrictions of different kinds which the solution currently does no cover. *Suggestions* denote ideas expressed by experts to overcome concerns and limitations named and/or for further improvement of existing functionality. The column *# of mentions* denotes the number an item was independently mentioned by an interviewee. I.e. the items were not presented to experts beforehand but identified from their individual and free feedback.

Appreciations, concerns and limitations were already mentioned in the previous section along with the analysis of interview responses. Therefore no further specification

Abbr.	Feedback Type	Topic	Description	# of mentions	Percentage
A1	Appreciation	Concept	Enables EA documentation closest to knowledge carriers	11	92%
A2	Appreciation	Concept	Always reflects a current as-is IT landscape due to the use of runtime data	8	67%
A3	Appreciation	Concept	Ead.json very important and well-suited to enrich static information	8	67%
A4	Appreciation	Process	Fits well into agile development process	7	58%
A5	Appreciation	Process	Involves agile teams into EA using their natural process & tool environment	7	58%
A6	Appreciation	Data Quality	Allows to reduces EA modelling errors due to automation	1	8%
A7	Appreciation	Concept	Pipeline-driven EA documentation ensures updates at most important points in time	6	50%
A8	Appreciation	Concept	Easy to apply integration procedure	5	42%
A9	Appreciation	Concept	Allows to overcome the gap between logical and technical EA modelling	3	25%
C1	Concern	Concept	Runtime data alone not valuable enough for EAM. Combination with ead.json is essential	5	42%
C10	Concern	Feasibility	Rollout difficulties due to a lack of technical process enforcement and acceptance	1	8%
C11	Concern	Scope	RUN-Phase is too late for EA documentation	3	25%
C12	Concern	Feasibility	Numerous CI/CD servers limit scalability of the solution	5	42%
C13	Concern	Data Quality	Risk of outdated and inconsistent ead.json files	2	17%
C2	Concern	Process	Leads to a fragmentation of the EA documentation process	2	17%
C3	Concern	Data Quality	Challenge of consolidation / harmonisation with other EA data sources	7	58%
C4	Concern	Data Quality	Challenge of providing appropirate data granularity of runtime information	7	58%
C5	Concern	Skills	Lack of skills in handling JSON file format by Product Owners	3	25%
C6	Concern	Skills	Lack of knowledge about business layer assignments in agile teams	6	50%
C7	Concern	Process	Reduced flexibility in manual EA modelling due to automation	1	8%
C8	Concern	Feasibility	Insufficient EAM budget	4	33%
C9	Concern	Adequacy	Tracing is too invasive and cost intensive	2	17%
L1	Limitation	Scope	Focus an RUN-Phase only - other EA lifecycle phases not covered	5	42%
L2	Limitation	Functionality	Lack of real automation for business layer assignments	1	8%
L3	Limitation	Scope	Scope not sufficient reg. IT landscape outside cloud environments	3	25%
L4	Limitation	Feasibility	Tracing is essential to make the solution more valuable	9	75%
S1	Suggestion	Scope	Integration of the EA PLAN-Phase	5	42%
S2	Suggestion	Functionality	Introduction of a system-overarching ID by the help if the solution	2	17%
S3	Suggestion	Functionality	Integration of ead.json contents into the artefact itself	2	17%
S4	Suggestion	Functionality	Form-based initialisation of ead.json file	9	75%
S5	Suggestion	Functionality	Automatic detection of federated information sources by static code analysis	1	8%
S6	Suggestion	Concept	Incorporate EAD pipeline stage into a standardised overarching pipeline	4	33%
S7	Suggestion	Concept	Usage of cloud native technologies instead of tracing	3	25%
S8	Suggestion	Feasibility	Voluntary adoption due to intrinsic motivation of agile teams	6	50%

Table 6.11.: Compiled feedback of expert interviews

is made at this place. For suggestions, additional explanation is required and provided in the following.

S1: Integration of the EA PLAN-Phase
Especially enterprise architects and domain architects claimed a lack of integration of the EA planning process, i.e. the planning of the target EA including the specification of new applications to be built (see concern C11 and limitation L1). At this phase architects already start to register and model elements in EAM repository in a *PLANNED* state. Once an application is implemented and the first components are build and deployed to the operating environment, the solution would be unable to link the EA elements created on its own with preexisting elements in a planned state. Today, this is already an issue as the same happens with imported EA elements originating from the CMDB that used to cover elements in a *CURRENT* state. Interviewees therefore promote to integrate the EA planning process into the solution.

S2: Introduction of a global unique application identifiers by the help if the solution
As of now no, naming conventions or global (i.e. system overarching) application identifiers exist at the industry partner. This constitutes an issue whenever elements from different information sources should be matched due to the lack of unique, shared attributes. Two enterprise architects saw the opportunity to overcome this gap by the help of the suggested solution. The pipeline integration technically allows to being extended to include such an identifier, for instance as part of the *ead.json*. Assuming a web formula based initialization process for the config file, a unique identifier could be generated and automatically placed into the configuration file generated. Along with the *ead.json* this ID becomes an integral part of an artifacts source code repository. Through its integrated nature with other federated systems, the solution could also be used to propagate unique IDs to other relevant systems it connects to (e.g. CMDB, EAM repository, etc.).

S3: Integration of ead.json contents into the artifact itself
According to two enterprise architects, a consequent improvement of the *ead.json* concept would be to build the information contained into an artifact instead of just extracting the file's information during the deployment process. This could be achieved either using application tags or environment variables that are automatically set based on *ead.json* information. Widely-spread cloud platforms including Cloud Foundry and Kubernetes feature such functionality based on CLI or API calls. Thanks to this, this suggestion could be realized within the *ead-library* with no further need to adapt the pipeline integration procedure. Resulting to this, business layer relationships could

also be retrieved directly from the cloud platform.

S4: Form-based initialization of *ead.json* file
As already suggested in the section 4.3.3 a form-supported initialization of the *ead.json* was suggested by numerous experts as this would further facilitate the process for individuals that are unfamiliar with the JSON format (see concern C5) and support the correct choice of business layer assignments (see concern C6). A driver of efforts currently is that these configuration files have to be created and integrated for each application component. However, most of the information contained, especially business layer assignments, rarely will deviate for components of an application. Form-support therefore could also serve to reduce efforts by generating the *ead.json* once per application and replicate it for each of its components. As a result, the risk of manual errors and inconsistencies introduced is minimized. The form should be connected to a repository that holds current information about existing business layer assignments e.g. the EAM repository for the current domain landscape or a directory service for business units.

S5: Automatic detection of federated information sources by static code analysis
From the pipeline's environment, it is possible to automatically detect federated information sources including the source code repository, the CI/CD system and if applicable, source code inspection tool (i.e. SonarQube[22], Selenium[23]) used during the test stage and binary repositories (e.g. Docker Hub[24], Artifactory[25]) where the build image is used to be uploaded and stored. Still, a lot of other relevant information sources remain to be provided manually as part of the *ead.json*, such as monitoring dashboard, logging systems, etc. A product owner argued that some links to such federated information systems could be automatically detected from the source code repository by the means of code analysis. For instance, the monitoring solution Grafana and Prometheus used to be configured in configuration files which could be extracted and analyzed for such information.

S6: Incorporate EAD pipeline stage into a standardized overarching pipeline
Following the slogan of "Trust is a good thing, control is better" some experts proposed to integrate the solution even deeper into CI/CD systems. They argue that the library could be delivered per default when being included in the CI/CD servers images. Moreover, instead of having individual deployment pipelines per team, they suggest to establish a standardized, team-overarching deployment pipeline that includes default

[22]https://www.sonarqube.org/
[23]https://www.seleniumhq.org/
[24]https://hub.docker.com/
[25]https://jfrog.com/artifactory/

steps needed across the entire organization but also the EA documentation stage. With such an approach even technical enforcement of this process would be possible. (see also concerns C10 and C12)

S7: Usage of cloud native technologies instead of tracing
Two enterprise architects considered distributed tracing as not appropriate for EA discovery as it is too invasive and cost intense when being conducted continuously (see concern C9). Moreover, enabled application-based tracing would reduce the application's performance. For these reasons, tracing should only be used on-demand. However, continuous tracing across the entire IT landscape is an important precondition to the discovery of communication relationships. This depicts a conflict with a potentially considerable cost impact. The experts, therefore, argue to use a cloud native solutions as provided for example by service meshes. Another option, for instance, are so-called *Kubernetes Operators*[26], an extension to the platforms master node that could actively report EA relevant information and activities using a push principle to the EAM repository.

S8: Adoption due to intrinsic motivation of agile teams
Some experts argued that the determined roll-out cost could not be covered by the existing EAM budget. However, the roll-out makes the major fraction of costs. Due to the solution's easy and fast to adopt integration procedure (see Appreciation A4 and A5), experts argue that such a roll-out must not be funded as a project. They rather see an intrinsic motivation with agile teams to adopt the solution as they can save time consumed by meetings, calls and email conversations which are currently needed to align manual EA modeling. Agile teams would therefore quickly realize the benefit of the suggested solution and thus integrate it on their own.

6.7.7. Revised process based on experts' feedback

This section proposes a preliminary, revised process which incorporates the most important experts' feedback. The process should overcome the Concerns C5, C6, C11 and Limitation L1 as well as consider the Suggestions S1, S2 and S4. As demanded by enterprise architects, the process already starts from the EA planning phase. The approach is oriented on the solution demonstrated by Farwick et al. [7]. The following concepts are required additionally:

- **System-overarching GUID:** In order to map elements contained in different information systems, it is key that there is at least one unique, shared attribute

[26]https://kubernetes.io/docs/concepts/extend-kubernetes/operator/

available which allows the matching of elements. This often is not the case as information systems use their own identifiers. Matching based on element names or other textual attributes is usually too prone to errors. The introduction of a system-overarching GUID for applications is therefore required to overcome this challenge. The GUID should be generated and handled by the *EAD-Tool*. This should satisfy Suggestion S2.

- **Handling of applications across the lifecycle:** If new applications are planned, domain architects used to create a new EA element in the EAM repository and assign it a "PLANNED" state. Once this planned application becomes reality and is being deployed to the cloud platform, the suggested solution would discover it and create another EA element for this application in a "CURRENT" state. This would cause a duplicate in the EAM repository as the previously registered element in a "PLANNED" state still exists in parallel. As of now, this cannot be consolidated. Therefore, the revised process suggests to bridge this gap by the help of the introduced GUID. This should overcome Concern C11, Limitation L1 and Suggestion S1.
- **From-based ead.json initialization at EA planning phase:** A prerequisite to the revised process is, that the *ead.json* configuration files can be generated using an interactive web-based form. A significant change to the process is, that *ead.json* files should already be created at the EA planning phase, i.e. if a new application is planned to be developed. The form should be provided by the *EAD-tool* and is meant to be used by domain architects who have knowledge about business layer. The aforementioned GUID is generated together with the *ead.json* and directly stored into it. This is important, so that the GUID is also present at the time the artifact is deployed. This approach should overcome the Concerns C5, C6 and should satisfy Suggestion S4.

Assuming that these concepts have been introduced, a revised process that could overcome many of the claimed issues could look as follows. (cf. figure 6.43).

Activities during EA Planning Phase

1. **Plan new application**: Domain architects play a central role during the planning of new applications. They are directly involved when such decisions are made. At this time it is already clear for what business purpose a new application is developed for. This means that the core business layer relationships the resulting artifact will have are already known (Business domain and subdomain, supported business units and processes, etc.).

2. **Register planned application in EAD-tool:** Domain architects should register a planned application including their business assignments by the help of the web-based form provided by the *EAD-tool*.

3. **Initialize ead.json stump:** Based on the input provided, the *ead.json* configuration file is generated. At this stage, not all information required by the *ead.json* is known. Therefore it is only a stub at this point in time. The remainder will be completed by agile teams.

4. **Generate application GUID:** Along with the previous step, a system-overarching GUID is generated and stored into the *ead.json*.

5. **Store ead.json incl. GUID & create EA element:** The *EAD-tool* creates and stores a new *Application* EA element and assigns the status "PLANNED". The *ead.json* is also stored to the database and provided as a download to agile teams for later purposes.

Activities during EA Build & Run Phase

1. **Design application components**: After the project started, agile teams begin to work on the application's implementation. At this phase, it becomes clear what components the application should consist of.

2. **Download, propagate & complete ead.json per application component:** Each of the application's components has to be equipped with a *ead.json* file. For this purpose the agile team downloads the pre-generated *ead.json* file from the *EAD-Tool*. The configuration file then has to be replicated and to be completed for each application components (i.e. add component name, description and technical references).

3. **Add ead.json & instrument pipeline:** Next, the agile teams adds the *ead.json* files into the source code repository of the corresponding component and instruments the pipeline as described in section 4.4.3.

4. **Build/deploy application component(s)**: Finally, the teams starts to deploy the application components to the cloud platform.

5. **Register application component as EA element**: Along with the deployment processes triggered by the agile teams the *ead-library* will extract the *ead.json* file and report the new component to the *EAD-Tool*. Based on this, the *EAD-Tool* creates a new EA element *Application Component* in the repository and assigns a "CURRENT" state.

6. **Map component based on GUID**: Using the application GUID contained in each *ead.json* file, the *EAD-Tool* can match the newly created elements to the *Application* EA element they belong to and which domain architects registered at the EA planning phase.

7. **Update status to "CURRENT"**: As at this time the first components have been deployed to production, the state of the parent application can be changed from "PLANNED" to "CURRENT".

The lifecycle does not end after the last activity listed. Following to this, the operation phase moves on and, at some point in time, the application will be decommissioned and/or substituted. This phases are already covered as described in the solution's EA model maintenance process in section 4.3.2.

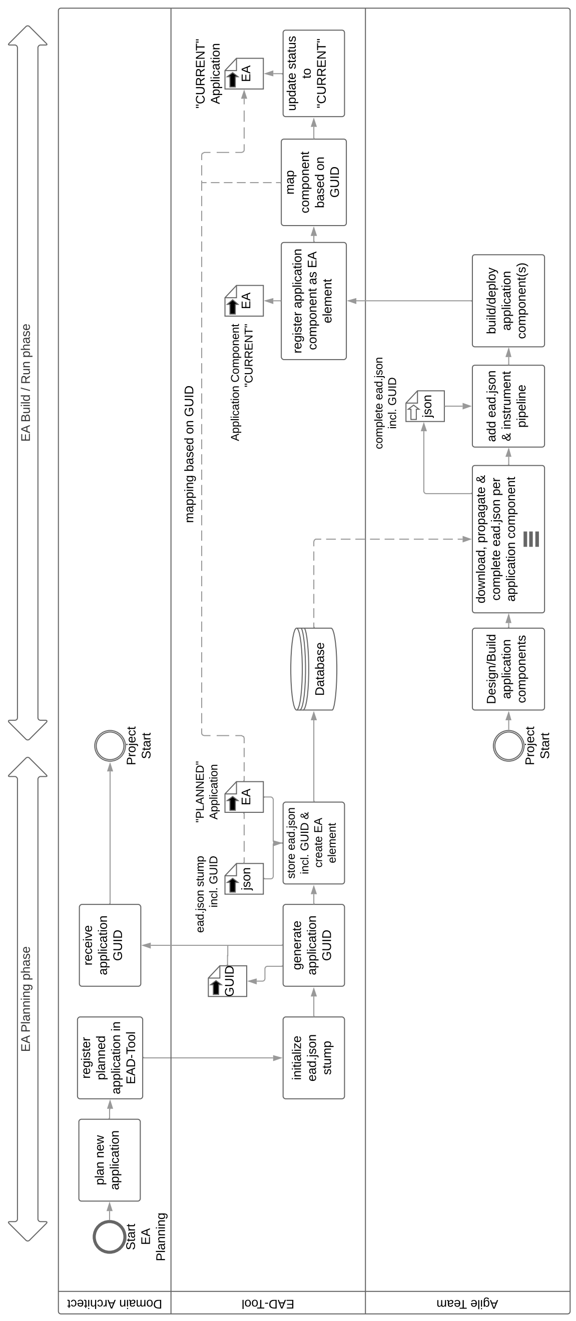

Domain Architect
EAD-Tool
Agile Team
EA Planning phase
EA Build / Run phase
Start EA Planning
plan new application
register planned application in EAD-Tool
receive application GUID
Project Start
initialize ead.json stump
generate application GUID
GUID
store ead.json incl. GUID & create EA element
ead.json stump incl. GUID
json
"PLANNED" Application
EA
Database
mapping based on GUID
Application Component "CURRENT"
register application component as EA element
map component based on GUID
update status to "CURRENT"
"CURRENT" Application
Project Start
Design/Build application components
download, propagate & complete ead.json per application component
complete ead.json incl. GUID
add ead.json & instrument pipeline
build/deploy application component(s)

Figure 6.43.: Revised process including the EA plan phase

7. Fulfillment of requirements and evaluation criteria

This sections elaborates to what degree the suggested automated EA model maintenance solution is capable to satisfy demanded requirements and evaluation criteria.

7.1. Fulfillment of requirements

Referring back to section 2 which summarizes general requirements to an automated EAMM solution and section 6.3 which analyzed the industry partner's specific EA information demand, this section gives a conclusion to what degree recorded requirements are fulfilled and where limitations remain.

Integration/Data Source Requirements: Section 4 explained how automated EAMM is achieved by the suggested solution. Due to the use of runtime data, the suggested solution is capable of reflecting the current IT landscape on the application component level at any given point in time. The detection of changes to information systems is enabled by regularly retrieving status updates about deployed apps from the cloud platform and by pipeline-triggered EA documentation. Thanks to this combination most important changes to the real-world EA can be reliably covered. At the case study environment, only a fraction of application inter-relationships and exposed APIs could be covered, as distributed tracing is not in place. A central API Gateway was used alternatively. It proved to be valuable for the discovery of exposed services and their clients, however, fails to provide full transparency as not all communication traffic passes the gateway. Usually, depending on the availability and the depth of instrumentation with distributed tracing technology, the solution should be capable to detect interfaces between information systems and changes to the underlying infrastructure. As this could not be demonstrated in this thesis, **Integration/Data Source Requirements** are considered to be satisfied to a large extent, except for inter-relationships on the EA application layer.

Architectural Requirements: As the solution promotes to incorporate additional information sources, Architectural Requirements regarding the use of a federated strategy formulated by [8] are satisfied. In this work, the potential EA model contribution of several information systems contained in a development tool-chain was assessed. A cloud platform, an API gateway and a VCS system were then actually incorporated for EA discovery. As of now, there is no generic solution concept of how federated information systems can efficiently be integrated. So far, the solution's source code would need to be extended to consume additional APIs and to transform the data into its internal data structure.

Data Quality Requirements: Thanks to the use of runtime data and deployment-triggered EA documentation, data actuality and validity is ensured at all times except for manual entries made in the *ead.json*. To solve this, the solution concept suggests a validation process for the *ead.json* (see section 4.3.3) to minimize the risk of errors. For the runtime data sources used, such functionality is not required as this kind of data directly reflects the as-is IT landscape and its behavior. Therefore, such data sources are highly credible. Instead, a challenge is to aggregate the raw data to an abstraction level that is useful to EAM. Within its own system boundaries and the set of data sources used, data consistency did not cause issues as elements can be mapped uniquely. As the data retrieved does not overlap, conflict resolution was not a problem as well. However, when reflecting the solution as part of a larger EAM ecosystem, including other independent data supplying information systems, the solution concept does not offer a general answer. Such environments are very specific to an organization and can hardly be generalized. All in all, it can be stated that *Data Quality Requirements* formulated by literature are fulfilled with regards to *actuality, validity and credibility*. More advanced functionality to support quality assurance teams or the propagation of changes is not covered by this work.

Functional Requirements named with regards to the definition and the calculation of KPIs based on runtime data were not in the scope of this work. Still, the suggested solution could pioneer the path into this direction and enable such functionality as it already interconnects important data suppliers. For instance, cloud platforms, API gateways and distributed tracing can provide runtime metrics and behavioral information that could be used to assess an application's operational stability. Moreover, pipeline integration could be used to asses an application's software quality and complexity. VCS systems provide detailed information about used software dependencies and technologies whereas Continuous Code Inspection systems store software test results, vulnerabilities and other indicators of source code quality. Both, VCS and Continuous Code inspection tools, are accessible from within the CI/CD pipeline. Within the course of this work, no assessment of the system under heavy load could be performed as only two agile teams supported the integration of the prototype. Therefore no judgment about the solution's capability to scale for large inputs can be provided.

Cost-Benefit ratio: An in-depth monetary assessment of the solution was conducted in section 6.6. The calculations resulted in an amortization period of four to five year solely by reducing manual EA modeling efforts. As other value propositions which cannot be directly mapped to financial impact have to be calculated additionally, a faster amortization period can be assumed. The use of distributed tracing data could additionally improve the financial performance as it allows to abolish manual modeling for application inter-relationships which are a main driver of efforts. Experts' judgments obtained during evaluation interviews showed that the solution is worth being rolled-out despite some further conditions named that should be considered.

7.2. Fulfillment of evaluation criteria

This section discusses the degree of fulfillment of evaluation criteria sketched in 5 based on the following inputs

1. Own practical experiences made during implementation and productive use across the case study
2. Qualitative feedback that originated from evaluation interviews
3. Quantitative observations

Validity refers to the degree the artifact works correctly to achieve its goals [28]. Vice versa this criteria asks about the potential the artifact might fail or work incorrectly. In the context of EA documentation, the challenge is to ensure high data quality and prevent the system from introducing inconsistencies, conflicts or duplicates into the data stock. The *ead-crawler*, the *ead.json* validation process and other quality assurance functionality described in chapter 4 are some of the instruments that serve to achieve this criterion. The system is designed to be proof of refactoring and to rely on GUID-based consistency checks where ever possible. Still, the case study's time frame did not allow for long term observation of the system's behavior. There might be situations that have not been considered yet. Another limitation is that only few artifacts were equipped with the solution and the productive EAM repository was not fully integrated. This means that there is no proof for the artifact's reliability yet.

Generality refers to the scope of the artifact's goal [28]. Transferred to the suggested solution this can be interpreted twofold. (1) Whether the solution can cover different kinds of cloud platforms or (2) whether the solution can also be applied to non-cloud environments. With regards to (1), it can be stated that the type of cloud platform is of little matter as long as the platform allows deployment using CI/CD technologies and there are exposed APIs for data retrieval. The solution would need to be extended to support a certain platform as there is no standard API across cloud platforms. Still, popular technologies like Kubernetes[1] or Docker[2] and also proprietary services as Azure Service Fabric[3] or AWS Elastic Container Service[4] allow for remote access through REST API or CLI calls. Regarding the second interpretation of generality, there are limitations. Operating environments that do not support a remote deployment by pipelines cannot be covered by the suggested solution. Assuming that tracing data is available for EA discovery, it depends on whether applications are instrumented from an end-to-end perspective. The case of the industry partner shows that this is not self-evident. As they are using a highly layered architecture with several independent cloud platforms separated by central API gateways or message queues this end-to-end perspective might be fragmented.

Fit into technical IS architecture: This criterion denotes the degree to which the artifact integrates into the technical IS architecture [28]. Actively affected (i.e. writing access or need for adaption) IS architecture elements are CI/CD servers and the documentation target system, the EAM repository. Passively affected, i.e. reading access only, are information sources used to gather data from (e.g. cloud platform, API Gateway, VCS). Section 4.4.3 and 4.3 described how the solution can be integrated into existing processes and systems without causing a lot of overhead. This was confirmed by interview responses presented in section 6.7. All other named systems are only affected by API calls for information gathering (read-only access). This means that except for the CD servers, the solution solely relies on exposed APIs. As no adaption is needed to any of these systems the solution is considered to have a good fit into the technical IS architecture.

Accuracy denotes the degree of agreement between outputs of the artifact and the expected outputs [28]. A central success factor to fulfill this requirement is a correctly implemented EA modeling behavior, i.e. the logic of how raw data is transformed into EA model elements. Before the implementation, a mapping has to be set up and aligned with the target EA meta model. It has to be well defined what gathered raw data has to be mapped to what target EA element or attribute. Once implemented, such mapping is static. Thus, there is little risk that actual outputs deviate from expected outputs. This

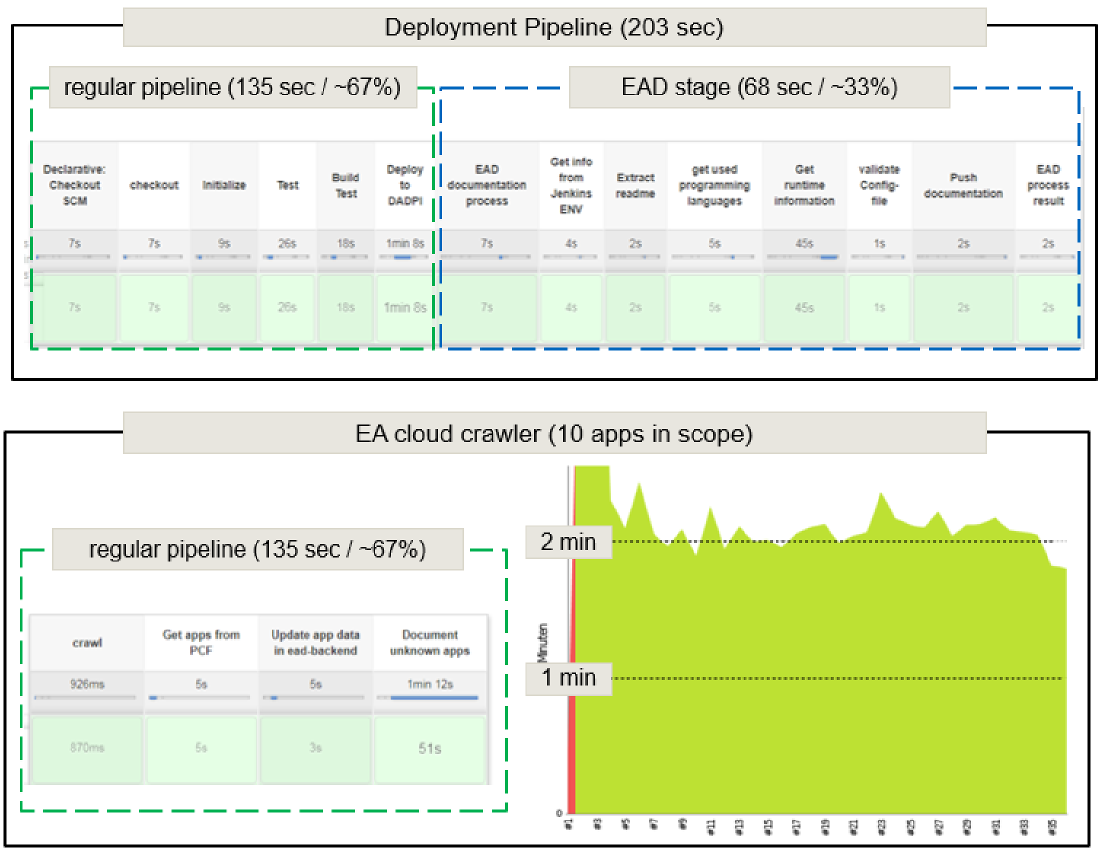
[1]https://kubernetes.io/docs/concepts/overview/kubernetes-api/
[2]https://docs.docker.com/registry/spec/api/
[3]https://docs.microsoft.com/de-de/rest/api/servicefabric/
[4]https://docs.aws.amazon.com/de_de/AmazonECS/latest/APIReference/Welcome.html

makes a general advantage of EA documentation automation over manual modeling.

Performance: [28] defines the term performance as "the degree to which the artifact accomplishes its functions within given constraints of time or space". As the solution consists of several independent components (*ead-library*, *ead-crawler* and *ead-backend*), performance is not easy to grasp. Due to the Jenkins Library provisioning model, most EA documentation logic is executed decentral as part of the deployment process. The *ead-crawler* is implemented as a pipeline job as well. Due to access rights limitations, it was only possible to test within the boundaries of a single Cloud Foundry space that contained ten deployed apps. Nevertheless, Figure 7.1 gives an indication of the solution's performance. The majority of time is consumed by logins and API calls at the cloud platform. Except for one product owner, interviewees judged the duration as acceptable. The *ead-crawler* was configured to run each working day at 01:00 AM. The average duration slightly fluctuates around 2.0 minutes for a scope of 10 deployed apps. Thereof four apps were instrumented with the *ead-library*. Additional documentation for the remaining, unregistered apps was necessary which consumed most of the time. Considering the total amount of more than 2,000 deployed apps the crawling procedure should be paralleled or implemented as an own microservice to achieve acceptable performance.

Adaptability is defined as "the ease with which the artifact can work in contexts other than those for which it was specifically designed" [28]. Parts of this criterion were already answered when elaborating on the solution's *Generality*. As the range of integrated federated information sources might change over time or vary according to organizations needs, the solution has to be easy to extend and maintain. Groovy[5], the used implementation programming language, is a scripting language that supports key concepts of object-oriented programming such as inheritance, interfaces, abstraction, etc.. Therefore, Groovy can be used to implement most important software design patterns that allow for easy extension by further information sources. As there are CI/CD solutions that do not support the Groovy language, the solution could be designed more universal by being implemented in Python[6], Ruby[7] or even in plain Shell scripts that have broader support than Groovy. From the backend's perspective, an universal data model, ideally adhering to an EA modeling standard such as ArchiMate, should be implemented. Still, for each information source used, integration logic is required to allow model transformation. The prototypical implementation so far does not fulfill this and therefore is subject to further development.

[5]https://groovy-lang.org/
[6]https://www.python.org/
[7]https://www.ruby-lang.org/


Deployment Pipeline (203 sec)
regular pipeline (135 sec / ~67%)
EAD stage (68 sec / ~33%)
EA cloud crawler (10 apps in scope)
regular pipeline (135 sec / ~67%)
2 min
1 min
Minuten
0
#1
#3
#5
#7
#9
#11
#13
#15
#17
#19
#21
#23
#25
#27
#29
#31
#33
#35

Declarative: Checkout SCM	checkout	Initialize	Test	Build Test	Deploy to DADPI	EAD documentation process	Get info from Jenkins ENV	Extract readme	get used programming languages	Get runtime information	validate Config-file	Push documentation	EAD process result
7s	7s	9s	26s	18s	1min 8s	7s	4s	2s	5s	45s	1s	2s	2s
7s	7s	9s	26s	18s	1min 8s	7s	4s	2s	5s	45s	1s	2s	2s

crawl	Get apps from PCF	Update app data in ead-backend	Document unknown apps
926ms	5s	5s	1min 12s
870ms	5s	3s	51s

Figure 7.1.: Duration EAD pipeline stage and EAD crawler

Alignment with business strategy and IT innovation: [28] defines this term as "the congruence of the artifact with the organization and its strategy" (business alignment) and "the degree to which the artifact uses innovative IT" (IT innovation alignment). The industry partner currently undertakes a large cloud migration endeavor to move and transform legacy information systems to cloud-ready applications. As a consequence, the number of cloud-based applications will continue to grow. New implementation projects have to be built on top of cloud technologies. This matches the scope of the suggested solution and therefore is in line with this business strategy. Apart from that, the industry partner prepares to introduce a global group level EAM repository whose use is mandatory to all country organizations. As used cloud platforms become increasingly standardized within the group, the value of the suggested solution increases if it would be adopted on an overarching level. Interviewed experts confirmed this reasoning as presented in section 6.7.

Absence of side effects: [28] defined this term as "the degree to which the artifact is free of undesirable impacts" on (1) the technical IS architecture, (2) the organization and (3) on individuals in the long run. During the evaluation interviews, experts were asked to judge and name risks that might emerge from the solution's adoption. These risks can be seen as undesirable side effects (cf. Figure 6.30). Moreover, they were asked whether the efforts caused to agile teams for the solution's integration are acceptable. The majority of the responses were positive (cf Figures 6.31, 6.33 and 6.34). Still, the risk lacking acceptance with agile teams is judged as *medium* which indicates that there further need for improvement. The agile teams involved reported that the initial adoption took four to five hours. This implies that the process requires further facilitation, for instance, form-based initialization of the *ead.json* configuration file or pre-configured pipelines. The risk of data quality issues with the *ead.json* configuration file is also considered to have *medium* severity. However, such issues can easily be mitigated by the use of form-based *ead.json* initialization and validation procedures as described in section 4.3.3. Challenges of handling conflicts and data consistency in the central EAM repository is a risk seen most severe by experts. This is not caused by the suggested solution itself but emerges from the industry partner's specific context where multiple sources import their data independently into the EAM repository. Insofar, the adoption of *EAD* could (in the specific context of the industry partner) lead to data quality issues as long as no further quality assurance functionality is implemented or data imports are fully free of overlapping. This is an issue that needs to be solved individually and is not of this work.

Technical Feasibility: This term is defined as "the ease with which a proposed artifact will be built and operated" [28]. As explained in section 4.4.2 the use of a library as a provisioning model is an important step to facilitate the maintenance of the EA documentation logic. Benefit are (1) the decoupling of maintenance and usage, (2) central management and maintenance, (3) reduced roll-out cost, (4) increased flexibility for future extensions without the need for another roll-out. At the industry partner, pipeline libraries are already in use for several purposes (e.g. share and improve reuse of code or provisioning of functional pipeline modules). One of these libraries is partly provided and managed by the EA department. This means, that the technical knowledge and skills on a provider (EA department) and consumer (agile teams) side are already established at the industry partner. It is subject to further investigation whether such knowledge is also present in other practitioners.

Functionality defines as "the capability of the artifact to provide functions which meet stated and implied needs when the artifact is used under specified conditions" [28]. This criterion is handled in detail in section 7.1.

Utility and Usefulness: The term *Usefulness* is defined as "the degree to which the artifact positively impacts the task performance of individuals" and *Utility* as "the value of achieving the artifact's goal, i.e. the difference between the worth of achieving this goal and the price paid for achieving it"[28]. The difference between these terms is that *Utility* considers the cost-benefit ratio on an organizational level whereas *Usefulness* focuses on the value proposition for individuals. During the evaluation interviews, it was confirmed that the suggested solution is worth being rolled out could pay back in a reasonable time (cf. Figures 6.23, 6.24 and 6.41). Key value propositions on an organizational level are (1) reduction of manual modeling efforts and errors, (2) increase of data quality in terms of completeness, actuality and reliability, (4) the engagement of agile team into EA documentation and (4) enabling of new use cases. Regarding certain affected roles, the main profiteers of the suggested solution are Domain Architects that currently bear most of EA modeling efforts as well as Agile Teams, which can save time for meetings, calls and email conversations to support Domain Architects with EA modeling. Agile teams also benefit insofar as the EA documentation integrates into their common process and tooling environment rather than imposing them to use an EAM repository.

Completeness is defined as "the degree to which the activity of the artifact contains all necessary elements and relationships between elements" [28]. This criterion refers to the solution's EA model coverage measured against a target EA meta model. This question is analyzed in-depth in section 6.4 and under real-case conditions in section 6.5. The evaluation interviews delivered experts' judgments about these criteria. The solution's capability to cover the *business layer* and *technology layer* reached the highest satisfaction level (cf. Figures 6.36 and 6.39). On the *application layer*, the solution is able to establish full transparency regarding deployed applications. Regarding application inter-relationships, however, the solution strongly depends on the availability of distributed tracing data to discover communication flows. In the course of this work, an API gateway was used as a potential alternative. However, it was found that the API gateway only covers a fraction of existing relationships. Therefore, the solution's capability to detect inter-relationships was judged to be *insufficient* regarding the current state of implementation (cf. Figure 6.37). Assuming that tracing data was available, most experts would have been *highly satisfied* (cf. Figure 6.38). In the industry partner's EAM repository, the EA element *Application* has more than a hundred attributes assigned whereof the majority has either a project-specific or qualitative character. As a consequence, the coverage of attributes measured against the industry partners meta model is rather low. Still, a majority of experts judged the solution's capability in this area as *rather satisfying* as it covers the fundamental attributes (cf. Figure 6.40).

Efficacy and Effectiveness: These criteria denote the "degree to which the artifact achieves its goal considered narrowly, without addressing situational concerns" (Efficacy) and under real-case circumstances (Effectiveness) [28]. Referring to the suggested solution, this asks whether the approach and is appropriate and suited for EAMM automation. Its characteristics are (1) the use of runtime data to discover EA elements, (2) the instrumentation of deployment pipelines to drive EA documentation, (3) the usage of configuration files to provide static information and lastly (4) the shift of responsibility of EA documentation into agile teams. Experts agreed or fully agreed that the approach is practicable and reasonable with regards to these core elements (cf. Figures 6.16, 6.17, 6.18 and 6.19) which confirms that the solution goes into the right direction. Moreover, the solution was attributed to have a good fit into agile development (cf. Figure 6.20) which is a necessity to engage agile teams into the EA process. As a result, the solution enables EA documentation close to knowledge carriers and leverages their expertise.

To conclude this section, table 7.1 summarizes and aggregates all of the above. The evaluation interview questions are mapped onto the list of evaluation criteria as referenced in table column *Input (Question \Source)*. The percentual distribution of answers and an average rating (Likert scale "1" (strong agreement) to "5" (strong disagreement)) is depicted in the table's center. Criteria that have no interview questions assigned were either answered by qualitative analysis or quantitative observation in the course of this master's thesis. Remaining criteria were qualitatively judged in the first part of this section.

Experts clearly see the concept's strengths in its *Efficacy*, *Alignment with business*, its

Evaluation Criteria	**Description**	**Level of Agreement on ordinal scale (1 - 5)**						**Input (Questions / Section)**
		1 (++)	2 (+)	3 (o)	4 (-)	5 (--)	**Avg. Rating**	
GOAL								
Efficacy	The artifact fulfills its purpose	61%	32%	5%	2%	0%	**1,51**	Q_1 - Q_4; Q_{21}
Effectiveness	The artifact achieves its goal in a real situation	qualitative analysis					---	cf. Chapter 6.4 and Chapter 6.5
Utility	The artifacts value, i.e. positive cost/benefit ratio	36%	29%	19%	9%	0%	**1,69**	Q_7; Q_{11a} - Q_{11b}; Q_{15}
Validity	The artefact works correctly	qualitative judgement					---	cf. Chapter 7.2
Generality	The broader the goal scope, the more general the artifact	qualitative judgement					---	cf. Chapter 7.2
ENVIRONMENT								
Technical Feasibility	The ease to built and operate the artifact	qualitative judgement					---	cf. Chapter 7.2
Operational Feasibility	The degree of support / acceptance by stakeholders	42%	50%	0%	8%	0%	**1,75**	Q_{16}
Economic Feasibility	The artifacts value, i.e. positive cost/benefit difference	27%	45%	9%	18%	0%	**2,18**	Q_9 - Q_{10}
Usefulness	The degree to which stakeholders profit from the artifact	qualitative judgement					---	cf. Chapter 7.2
Ease of Use	The degree to which the artifact usage is free of effort	53%	42%	5%	0%	0%	**1,51**	Q_{17}; Q_{22} - Q_{24}
Alignment with Business	The congruence with the organization and its strategy	82%	18%	0%	0%	0%	**1,18**	Q_{12}
Technology Fit	The artifact integrates well into the IS architecture	10%	70%	20%	0%	0%	**2,10**	Q_5 - Q_6; Q_{13}
Absence of Side Effects	The artifact is free of undesirable impacts	3%	21%	27%	43%	6%	**2,73**	Q_{14}; Q_{18} - Q_{19}; Q_{25}
ACTIVITY								
Functionality	Provided functionality meets stated and implied needs	qualitative analysis					---	cf. Chapter 7.1
Accuracy	Accordance of acual and expected outputs of the artifact	qualitative judgement					---	cf. Chapter 7.2
Performance	Accomplishment of functions within given constraints	quantitative observation					---	cf. Chapter 7.2
STRUCTURE								
Completeness	Containment of all necessary elements and relationships	49%	40%	9%	0%	2%	**1,70**	Q_{26} - Q_{29}
EVOLUTION								
Adaptability	The artifact can work in different contexts	qualitative judgement					---	cf. Chapter 7.2

Table 7.1.: Overall fulfillment of evaluation criteria incl. tracing data

Ease of Use and *Operational Feasibility*. The solution's *Utility* and *Completeness* is also seen as a strength given that distributed tracing data is available. Without tracing data, as it was the case at the industry partner, the average ratings would decrease to 2.54 for *Completeness* and 2.4 for *Utility*. A good *Technology fit* is at hand, however, there could be further improvements. The costs to implement and roll-out the solution are seen as reasonable. Still, due to the limited EAM budget available, a full sponsoring by the EA department is doubtful. An average rating for *Economic feasibility* of 2.03 which is still a good result. A weakness of the solution is seen in the criteria *Absence of Side Effect* that subsumed potential risks including acceptance, data quality issues as well as performance and security concerns. Risks with regards to handling data conflicts and data consistency are not caused by the solution itself but emerge from the industry partner's specific EAM environment where other automatisms are already in place. As a result, a key success factor to adopting organization is that a seamless interplay between several automation solutions is ensured.

8. Summary

This master's thesis is based on a holistic, multi-layer Enterprise Architecture discovery framework, called *MICROLYZE*, developed as part of a larger research endeavor by the university's chair. This framework is meant to achieve automated EA model documentation and maintenance with regards to the special challenges that emerge from the increasing adoption of cloud computing, agile methodologies and microservice-based architectures. Key enabling technologies used by the solution are cloud platforms, distributed tracing and continuous deployment. This work focused on the evaluation of CI/CD driven EA discovery as one of the main foundations of *MICROLYZE*.

The goal of this thesis was to assess the cost and benefits of the aforementioned solution concept in a real-case enterprise environment. To fulfill this, the work covered two major parts: (1) A detailed description and evolution of how CI/CD pipelines can be instrumented to serve the goal of EA model maintenance including required integration processes and a prototypical implementation and (2) a comprehensive evaluation conducted in the course of a case study and a series of expert interviews. This included the solution's capabilities, value propositions, cost and savings.

The qualitative analysis of the solution within an enterprise context and the adoption to two real-case projects brought forth valuable insights about the capabilities and value propositions. Besides, limitations and challenges that remain to be solved were identified. This allowed to answer the research questions posed at the beginning of this work and which are shortly summarized:

Research Question 1 ("How can the suggested solution be integrated into agile development and what challenges do occur?"): As the solution has a direct impact on agile teams in the field, a key success factor is to keep the solution easy to adopt and integrate. The implemented prototype and its adoption to real-case projects showed that this challenge can be solved by providing the required EA documentation logic as a library. This enables an easy integration procedure for agile teams while in parallel, the functionality contained is centrally maintainable for the EA department. Form-based initialization of the *ead.json* configuration files can further facilitate the process and reduce the risk of errors. As long as *MICROLYZE* cannot replace classic EAM repositories, a challenge that remains to be solved, is the collaboration with preexisting EAM ecosystems consisting of processes, tools and other data sources which are specific to adopting organizations.

Research Question 2 ("What EA model elements should be documented and to what degree can this be automated using the solution approach?"): This work has revealed that the solution is capable to cover most important aspects of an EA meta model that spans across all architectural layers. Information sources used in this work are a cloud platform, a API gateway, source code repositories and configuration files. For the interviewed Enterprise Architects, the combination of business-related information contained in the *ead.json* configuration files with runtime data retrieved from operating environments plays a crucial role to make the data valuable and usable for EA purposes. Distributed tracing is indispensable as a data source to discover application inter-relationships based on their communication behavior.

Research Question 3 ("What are the solution's integration costs and value propositions for Enterprise Architecture Management?"): The in-depth monetary analysis and experts' feedback revealed, that the investment required to develop, roll-out and operate the suggested solution is acceptable and can pay off in a reasonable time due to the reduction of manual EA modeling efforts and other value propositions. Key cost drivers are the number of application components and pipelines to be integrated. Given a consolidated CI/CD infrastructure, costs can be significantly lowered. Distributed tracing plays an important role in the solution's value proposition as it could make manual efforts for time-consuming relationship modeling obsolete and thus, further increase the cost savings potential.

8.1. Limitations

MICROLYZE could not be evaluated in its entirety as distributed tracing turned out not to be a reliable data source at the case study environment. Besides this constraint, the following limitations are captured:

1. Due to the aforementioned, the monetary analysis does not include figures about the costs and savings that emerge from the use of distributed tracing.
2. As part of the case study, only a few teams did integrate the prototype into their environment. Future work should include additional field studies distributed across multiple organizations to draw conclusions with general validity.

3. The case study time frame did not allow to make long-term observations about the solution's behavior. As a consequence, no judgment about the long term impact on EA model quality and completeness can be inferred.

8.2. Suggestions for future work

As inferred from the expert interviews conducted, future work and development of *MICROLYZE* should include the following.

Preexisting EAM ecosystems: Meanwhile, EAM has evolved to a widely-adopted discipline and is being practiced for several years to even decades. This implies that organizations might already use established EAM repositories and might have partly solved the challenge of automated EA documentation. To be successful, *MICROLYZE* has to face such situations as enterprises might no longer be on a greenfield. Future work should develop ideas on how *MICROLYZE* can complement and collaborate with preexisting environments.

Integration of the EA plan phase: As of now, *MICROLYZE* concentrates on the discovery of the current state of Enterprise Architecture. However, EAM used to maintain multiple states of EA including a target state and, optionally, one or more intermediate states. Experts therefore demand that the solution should support the evolution of EA elements from a "planned" state into the "current" state. In section 6.7.7, a preliminary process was presented that might form the basis for future development.

EAM KPIs and automation of architecture assessment: *MICROLYZE* interconnects various runtime information sources as well as tools along the software development tool-chain. This forms a solid foundation for the calculation of EAM KPIs from runtime data. It can also serve to facilitate or even automate architectural assessments that used to be manual work. Continuous reporting on software quality, complexity (e.g. based on VCS and Continuous Code Inspection systems) as well as compliance with organizational standards or adherence to best practices (e.g. 12-factor-app[1]) could be the subject of such KPIs. Experts' feedback obtained in this work indicates that these are promising use cases. Future research should assess the needs of practitioners and evaluate how *MICROLYZE* could support this.

Cloud native EA discovery: Services meshes could be a valuable and lightweight solution to discover applications and their inter-relationships as compared to distributed

[1]https://12factor.net/

tracing. Examples of open-source service meshes are Istio[2] or LINKERD[3] but also proprietary solution offered by public cloud providers (e.g. AWS AppMesh[4] or Google Cloud Service Mesh[5]) exist. Istio, for example, exposes JSON based service graphs via REST APIs that could easily be integrated. Future work should closely watch this emerging technology and assess its potential for automated EA model maintenance.

[2]https://istio.io/
[3]https://linkerd.io/2/overview/
[4]https://aws.amazon.com/de/app-mesh/
[5]https://cloud.google.com/service-mesh/

A. Appendix

A.1. Appendix to EA model coverage

A.2. Appendix to Cost & Savings analysis

A.2.1. Estimated implementation onetime and running costs for *EAD*

A.2.2. AS-IS EAM repository quantities

A.3. Evaluation Interview

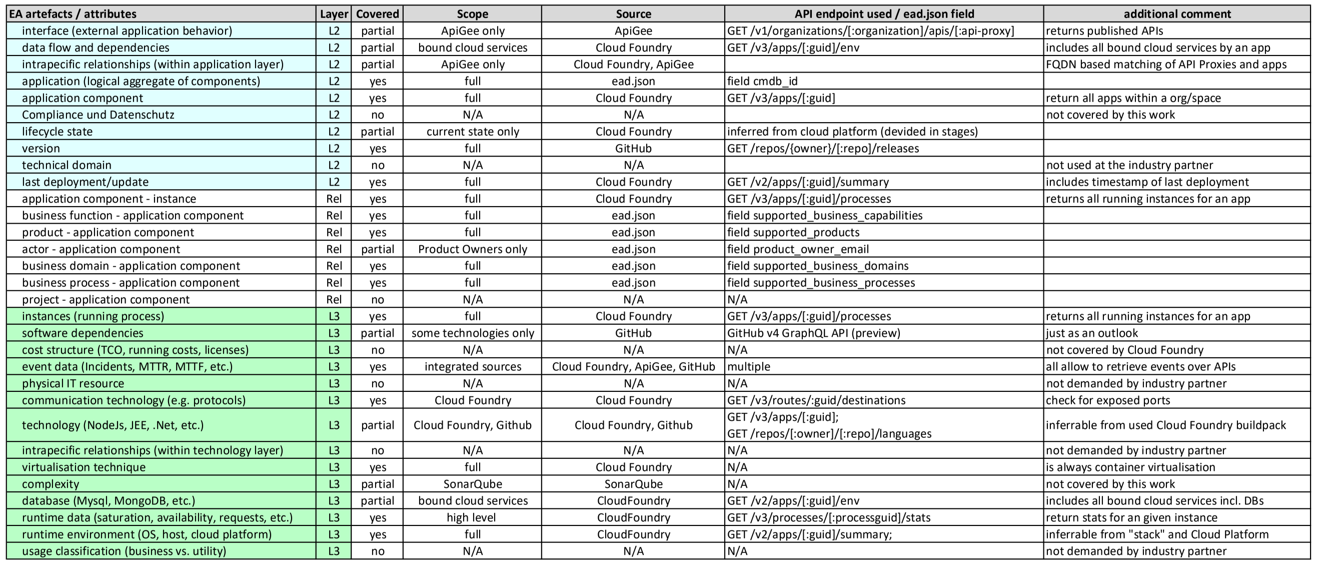
A.3.1. Part I - General Information

A.3.2. Part II - General feedback about solution approach

A.3.3. Part III - Pipeline integration

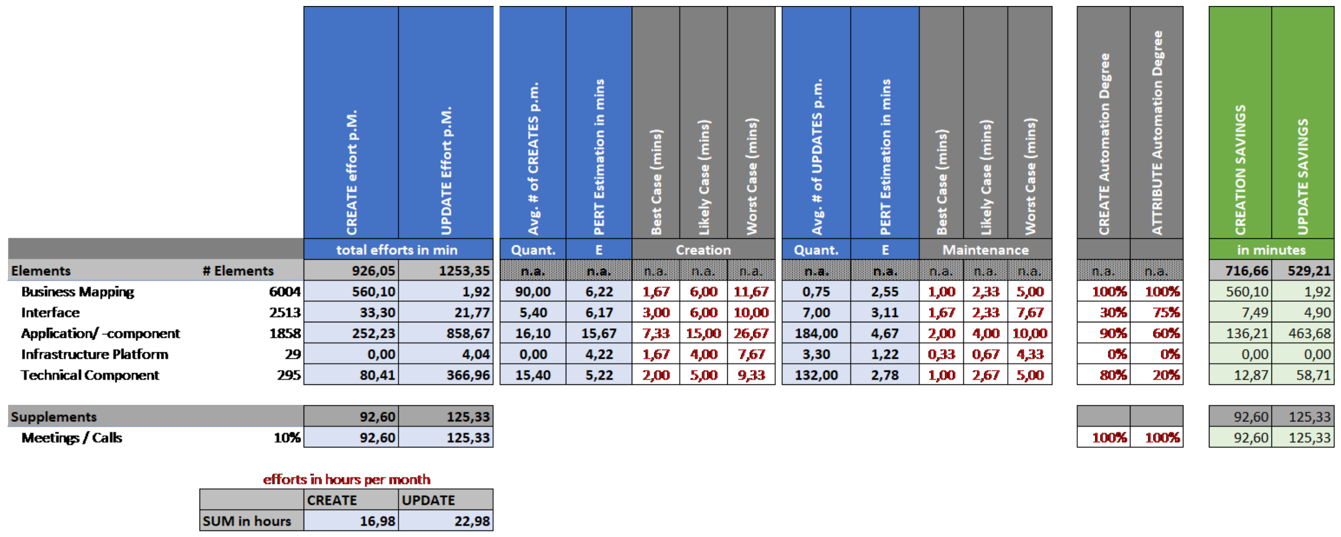
A.3.4. Part IV - ead.json configuration file

A.3.5. Part V - EA Documentation coverage



EA artefacts / attributes	Layer	Covered	Scope	Source	API endpoint used / ead.json field	additional comment
interface (external application behavior)	L2	partial	ApiGee only	ApiGee	GET /v1/organizations/[:organization]/apis/[:api-proxy]	returns published APIs
data flow and dependencies	L2	partial	bound cloud services	Cloud Foundry	GET /v3/apps/[:guid]/env	includes all bound cloud services by an app
intrapecific relationships (within application layer)	L2	partial	ApiGee only	Cloud Foundry, ApiGee		FQDN based matching of API Proxies and apps
application (logical aggregate of components)	L2	yes	full	ead.json	field cmdb_id	
application component	L2	yes	full	Cloud Foundry	GET /v3/apps/[:guid]	return all apps within a org/space
Compliance und Datenschutz	L2	no	N/A	N/A		not covered by this work
lifecycle state	L2	partial	current state only	Cloud Foundry	inferred from cloud platform (devided in stages)	
version	L2	yes	full	GitHub	GET /repos/{owner}/[:repo]/releases	
technical domain	L2	no	N/A	N/A		not used at the industry partner
last deployment/update	L2	yes	full	Cloud Foundry	GET /v2/apps/[:guid]/summary	includes timestamp of last deployment
application component - instance	Rel	yes	full	Cloud Foundry	GET /v3/apps/[:guid]/processes	returns all running instances for an app
business function - application component	Rel	yes	full	ead.json	field supported_business_capabilities	
product - application component	Rel	yes	full	ead.json	field supported_products	
actor - application component	Rel	partial	Product Owners only	ead.json	field product_owner_email	
business domain - application component	Rel	yes	full	ead.json	field supported_business_domains	
business process - application component	Rel	yes	full	ead.json	field supported_business_processes	
project - application component	Rel	no	N/A	N/A	N/A	
instances (running process)	L3	yes	full	Cloud Foundry	GET /v3/apps/[:guid]/processes	returns all running instances for an app
software dependencies	L3	partial	some technologies only	GitHub	GitHub v4 GraphQL API (preview)	just as an outlook
cost structure (TCO, running costs, licenses)	L3	no	N/A	N/A	N/A	not covered by Cloud Foundry
event data (Incidents, MTTR, MTTF, etc.)	L3	yes	integrated sources	Cloud Foundry, ApiGee, GitHub	multiple	all allow to retrieve events over APIs
physical IT resource	L3	no	N/A	N/A	N/A	not demanded by industry partner
communication technology (e.g. protocols)	L3	yes	Cloud Foundry	Cloud Foundry	GET /v3/routes/:guid/destinations	check for exposed ports
technology (NodeJs, JEE, .Net, etc.)	L3	partial	Cloud Foundry, Github	Cloud Foundry, Github	GET /v3/apps/[:guid]; GET /repos/[:owner]/[:repo]/languages	inferrable from used Cloud Foundry buildpack
intrapecific relationships (within technology layer)	L3	no	N/A	N/A	N/A	not demanded by industry partner
virtualisation technique	L3	yes	full	Cloud Foundry	N/A	is always container virtualisation
complexity	L3	partial	SonarQube	SonarQube	N/A	not covered by this work
database (Mysql, MongoDB, etc.)	L3	partial	bound cloud services	CloudFoundry	GET /v2/apps/[:guid]/env	includes all bound cloud services incl. DBs
runtime data (saturation, availability, requests, etc.)	L3	yes	high level	CloudFoundry	GET /v3/processes/[:processguid]/stats	return stats for an given instance
runtime environment (OS, host, cloud platform)	L3	yes	full	CloudFoundry	GET /v2/apps/[:guid]/summary;	inferrable from "stack" and Cloud Platform
usage classification (business vs. utility)	L3	no	N/A	N/A	N/A	not demanded by industry partner

Figure A.1.: Questionnaire coverage by selected information sources and relevant API endpoints

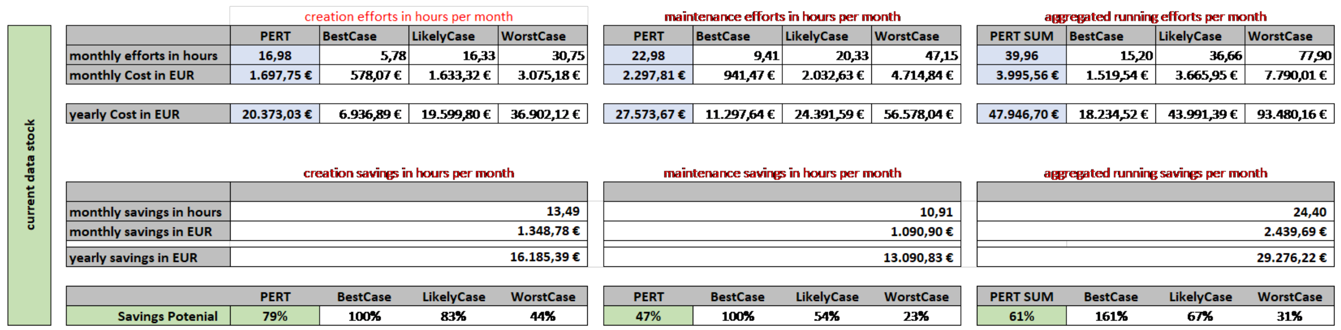


		CREATE effort p.M.	UPDATE Effort p.M.	Avg. # of CREATES p.m.	PERT Estimation in mins	Best Case (mins)	Likely Case (mins)	Worst Case (mins)	Avg. # of UPDATES p.m.	PERT Estimation in mins	Best Case (mins)	Likely Case (mins)	Worst Case (mins)	CREATE Automation Degree	ATTRIBUTE Automation Degree	CREATION SAVINGS	UPDATE SAVINGS
		total efforts in min		Quant.	E	Creation			Quant.	E	Maintenance					in minutes	
Elements	# Elements	926,05	1253,35	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	716,66	529,21
Business Mapping	6004	560,10	1,92	90,00	6,22	1,67	6,00	11,67	0,75	2,55	1,00	2,33	5,00	100%	100%	560,10	1,92
Interface	2513	33,30	21,77	5,40	6,17	3,00	6,00	10,00	7,00	3,11	1,67	2,33	7,67	30%	75%	7,49	4,90
Application/ -component	1858	252,23	858,67	16,10	15,67	7,33	15,00	26,67	184,00	4,67	2,00	4,00	10,00	90%	60%	136,21	463,68
Infrastructure Platform	29	0,00	4,04	0,00	4,22	1,67	4,00	7,67	3,30	1,22	0,33	0,67	4,33	0%	0%	0,00	0,00
Technical Component	295	80,41	366,96	15,40	5,22	2,00	5,00	9,33	132,00	2,78	1,00	2,67	5,00	80%	20%	12,87	58,71
Supplements		92,60	125,33													92,60	125,33
Meetings / Calls	10%	92,60	125,33											100%	100%	92,60	125,33

efforts in hours per month

	CREATE	UPDATE
SUM in hours	16,98	22,98

Figure A.2.: Effort estimations and savings potential



current data stock

creation efforts in hours per month

	PERT	BestCase	LikelyCase	WorstCase
monthly efforts in hours	16,98	5,78	16,33	30,75
monthly Cost in EUR	1.697,75 €	578,07 €	1.633,32 €	3.075,18 €
yearly Cost in EUR	20.373,03 €	6.936,89 €	19.599,80 €	36.902,12 €

maintenance efforts in hours per month

PERT	BestCase	LikelyCase	WorstCase
22,98	9,41	20,33	47,15
2.297,81 €	941,47 €	2.032,63 €	4.714,84 €
27.573,67 €	11.297,64 €	24.391,59 €	56.578,04 €

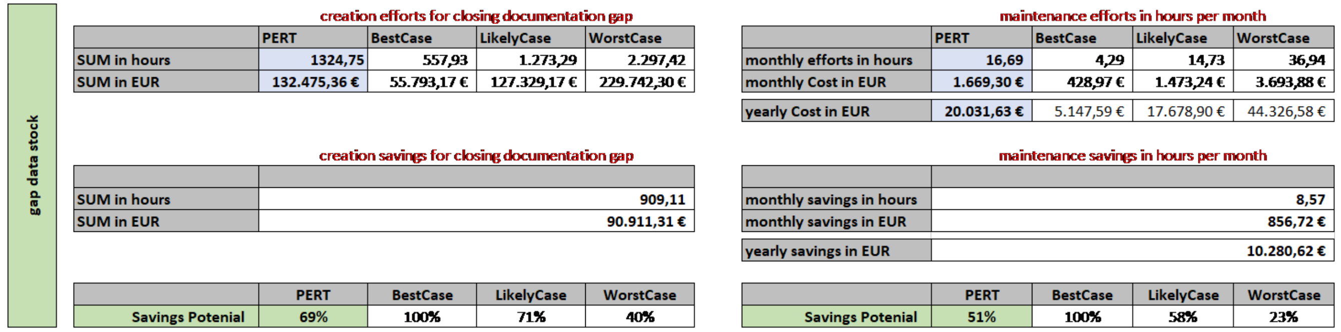
aggregated running efforts per month

PERT SUM	BestCase	LikelyCase	WorstCase
39,96	15,20	36,66	77,90
3.995,56 €	1.519,54 €	3.665,95 €	7.790,01 €
47.946,70 €	18.234,52 €	43.991,39 €	93.480,16 €

	creation savings in hours per month	maintenance savings in hours per month	aggregated running savings per month
monthly savings in hours	13,49	10,91	24,40
monthly savings in EUR	1.348,78 €	1.090,90 €	2.439,69 €
yearly savings in EUR	16.185,39 €	13.090,83 €	29.276,22 €

	PERT	BestCase	LikelyCase	WorstCase	PERT	BestCase	LikelyCase	WorstCase	PERT SUM	BestCase	LikelyCase	WorstCase
Savings Potenial	79%	100%	83%	44%	47%	100%	54%	23%	61%	161%	67%	31%

Figure A.3.: Monthly running cost and savings potential of the current data stock


gap data stock

creation efforts for closing documentation gap

	PERT	BestCase	LikelyCase	WorstCase
SUM in hours	1324,75	557,93	1.273,29	2.297,42
SUM in EUR	132.475,36 €	55.793,17 €	127.329,17 €	229.742,30 €

creation savings for closing documentation gap

SUM in hours	909,11
SUM in EUR	90.911,31 €

	PERT	BestCase	LikelyCase	WorstCase
Savings Potenial	69%	100%	71%	40%

maintenance efforts in hours per month

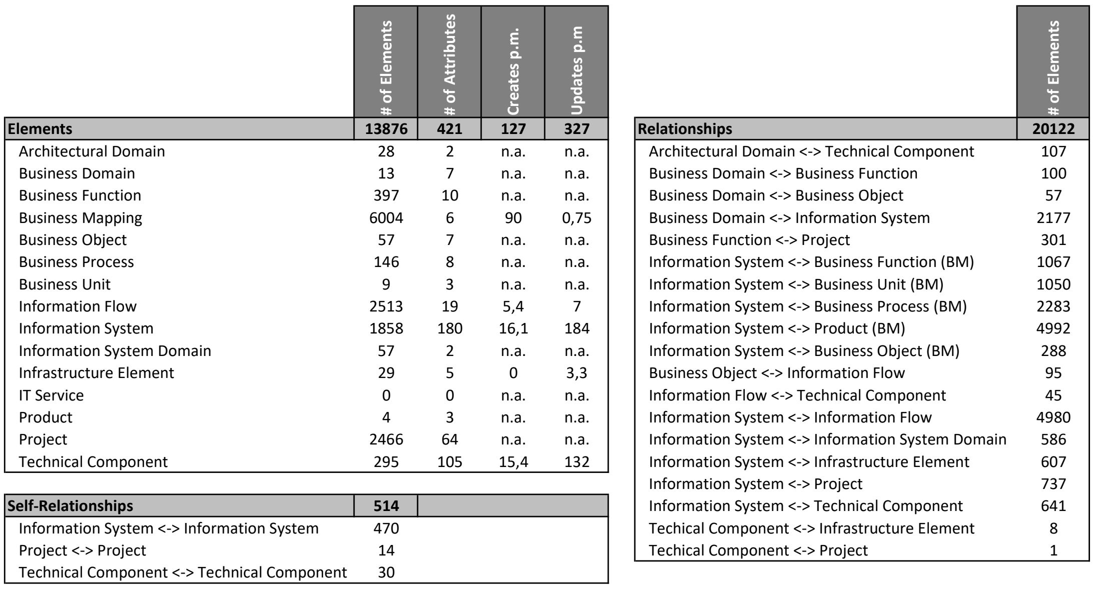
	PERT	BestCase	LikelyCase	WorstCase
monthly efforts in hours	16,69	4,29	14,73	36,94
monthly Cost in EUR	1.669,30 €	428,97 €	1.473,24 €	3.693,88 €
yearly Cost in EUR	20.031,63 €	5.147,59 €	17.678,90 €	44.326,58 €

maintenance savings in hours per month

monthly savings in hours	8,57
monthly savings in EUR	856,72 €
yearly savings in EUR	10.280,62 €

	PERT	BestCase	LikelyCase	WorstCase
Savings Potenial	51%	100%	58%	23%

Figure A.4.: Onetime and running cost and savings potential for documentation gap to be closed

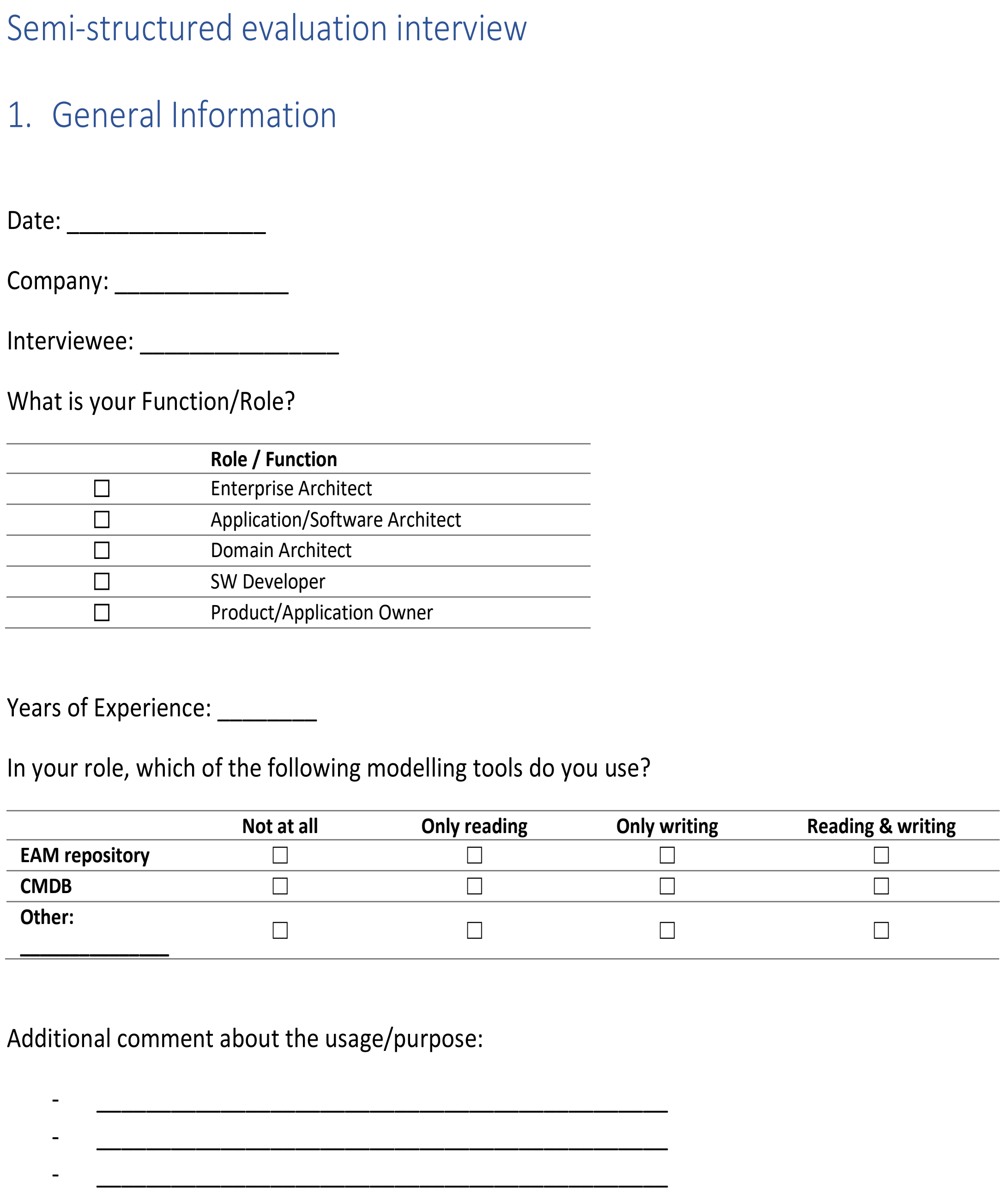


Elements	# of Elements	# of Attributes	Creates p.m.	Updates p.m
Elements	**13876**	**421**	**127**	**327**
Architectural Domain	28	2	n.a.	n.a.
Business Domain	13	7	n.a.	n.a.
Business Function	397	10	n.a.	n.a.
Business Mapping	6004	6	90	0,75
Business Object	57	7	n.a.	n.a.
Business Process	146	8	n.a.	n.a.
Business Unit	9	3	n.a.	n.a.
Information Flow	2513	19	5,4	7
Information System	1858	180	16,1	184
Information System Domain	57	2	n.a.	n.a.
Infrastructure Element	29	5	0	3,3
IT Service	0	0	n.a.	n.a.
Product	4	3	n.a.	n.a.
Project	2466	64	n.a.	n.a.
Technical Component	295	105	15,4	132

Self-Relationships	**514**
Information System <-> Information System	470
Project <-> Project	14
Technical Component <-> Technical Component	30

Relationships	# of Elements
Relationships	**20122**
Architectural Domain <-> Technical Component	107
Business Domain <-> Business Function	100
Business Domain <-> Business Object	57
Business Domain <-> Information System	2177
Business Function <-> Project	301
Information System <-> Business Function (BM)	1067
Information System <-> Business Unit (BM)	1050
Information System <-> Business Process (BM)	2283
Information System <-> Product (BM)	4992
Information System <-> Business Object (BM)	288
Business Object <-> Information Flow	95
Information Flow <-> Technical Component	45
Information System <-> Information Flow	4980
Information System <-> Information System Domain	586
Information System <-> Infrastructure Element	607
Information System <-> Project	737
Information System <-> Technical Component	641
Techical Component <-> Infrastructure Element	8
Techical Component <-> Project	1

Figure A.5.: EAM repository quantities as of August 2019



Semi-structured evaluation interview

1. General Information

Date: ________________

Company: ______________

Interviewee: ________________

What is your Function/Role?

	Role / Function
☐	Enterprise Architect
☐	Application/Software Architect
☐	Domain Architect
☐	SW Developer
☐	Product/Application Owner

Years of Experience: ________

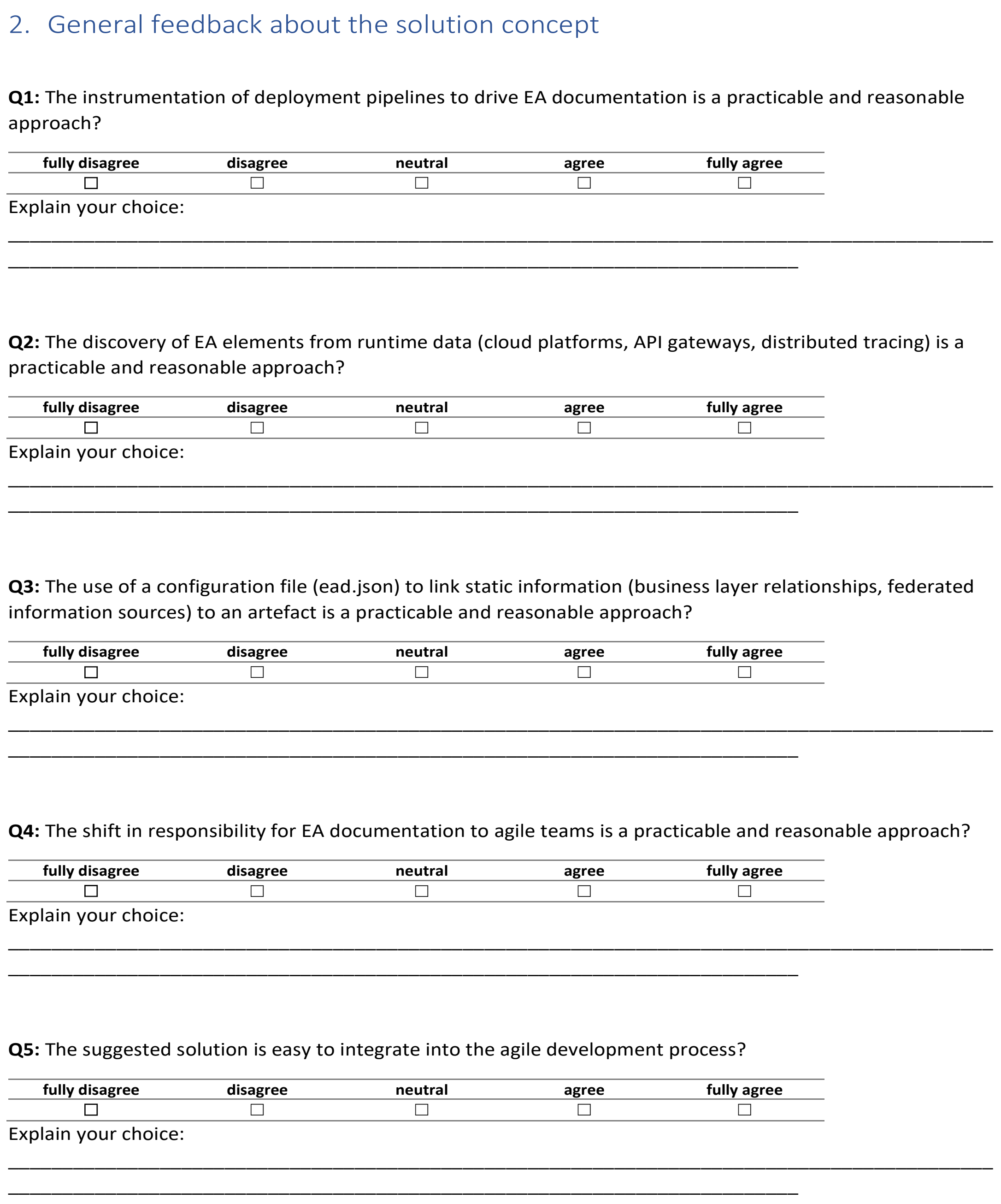
In your role, which of the following modelling tools do you use?

	Not at all	Only reading	Only writing	Reading & writing
EAM repository	☐	☐	☐	☐
CMDB	☐	☐	☐	☐
Other: ____________	☐	☐	☐	☐

Additional comment about the usage/purpose:

- ____
- ____
- ____

Figure A.6.: Part I - General Information



2. General feedback about the solution concept

Q1: The instrumentation of deployment pipelines to drive EA documentation is a practicable and reasonable approach?

fully disagree	disagree	neutral	agree	fully agree
☐	☐	☐	☐	☐

Explain your choice:

____

____

Q2: The discovery of EA elements from runtime data (cloud platforms, API gateways, distributed tracing) is a practicable and reasonable approach?

fully disagree	disagree	neutral	agree	fully agree
☐	☐	☐	☐	☐

Explain your choice:

____

____

Q3: The use of a configuration file (ead.json) to link static information (business layer relationships, federated information sources) to an artefact is a practicable and reasonable approach?

fully disagree	disagree	neutral	agree	fully agree
☐	☐	☐	☐	☐

Explain your choice:

____

____

Q4: The shift in responsibility for EA documentation to agile teams is a practicable and reasonable approach?

fully disagree	disagree	neutral	agree	fully agree
☐	☐	☐	☐	☐

Explain your choice:

____

____

Q5: The suggested solution is easy to integrate into the agile development process?

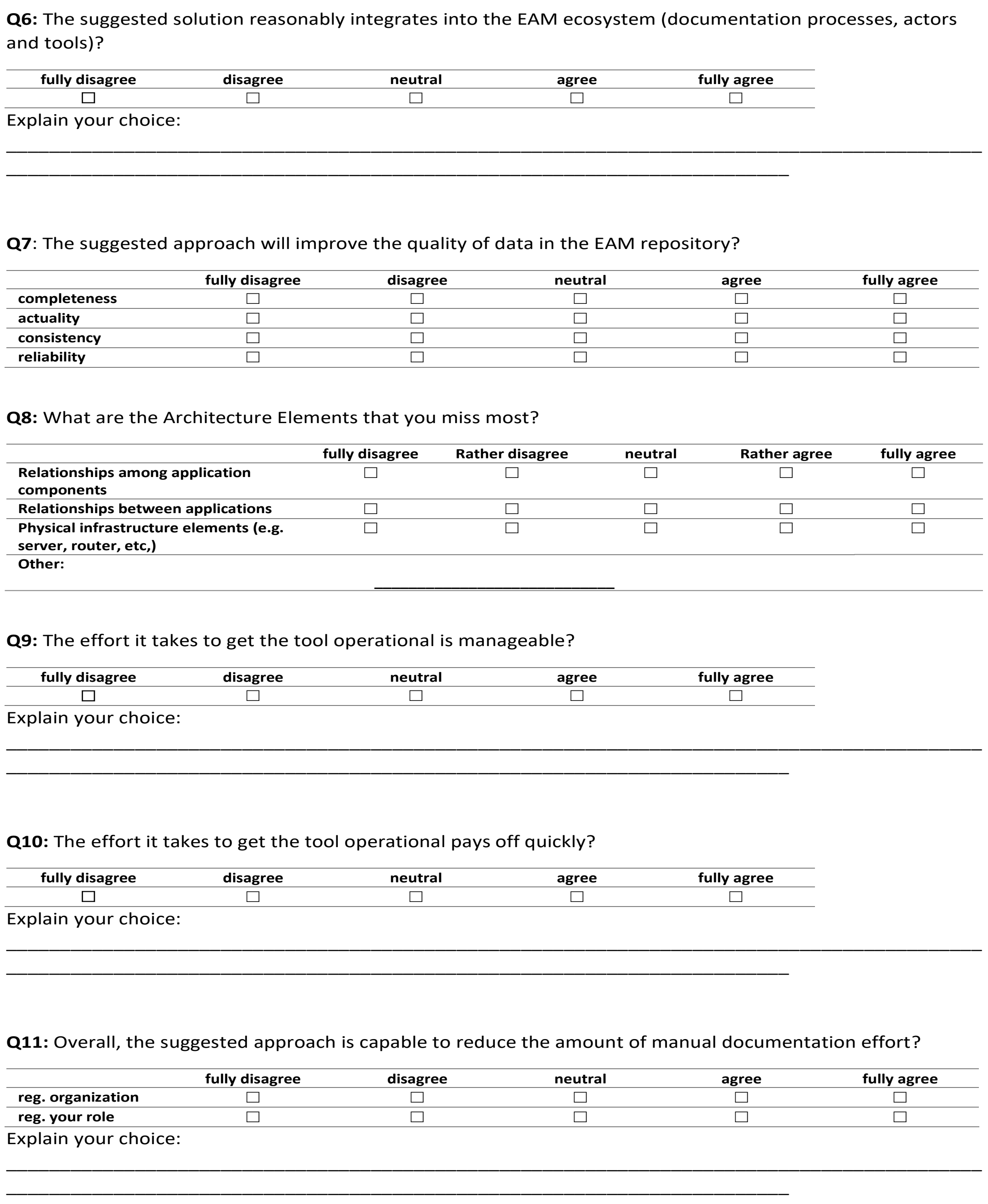
fully disagree	disagree	neutral	agree	fully agree
☐	☐	☐	☐	☐

Explain your choice:

____

____

Figure A.7.: Part IIa - General feedback about solution approach



Q6: The suggested solution reasonably integrates into the EAM ecosystem (documentation processes, actors and tools)?

fully disagree	disagree	neutral	agree	fully agree
☐	☐	☐	☐	☐

Explain your choice:

____

____

Q7: The suggested approach will improve the quality of data in the EAM repository?

	fully disagree	disagree	neutral	agree	fully agree
completeness	☐	☐	☐	☐	☐
actuality	☐	☐	☐	☐	☐
consistency	☐	☐	☐	☐	☐
reliability	☐	☐	☐	☐	☐

Q8: What are the Architecture Elements that you miss most?

	fully disagree	Rather disagree	neutral	Rather agree	fully agree
Relationships among application components	☐	☐	☐	☐	☐
Relationships between applications	☐	☐	☐	☐	☐
Physical infrastructure elements (e.g. server, router, etc,)	☐	☐	☐	☐	☐
Other:	____________________				

Q9: The effort it takes to get the tool operational is manageable?

fully disagree	disagree	neutral	agree	fully agree
☐	☐	☐	☐	☐

Explain your choice:

____

____

Q10: The effort it takes to get the tool operational pays off quickly?

fully disagree	disagree	neutral	agree	fully agree
☐	☐	☐	☐	☐

Explain your choice:

____

____

Q11: Overall, the suggested approach is capable to reduce the amount of manual documentation effort?

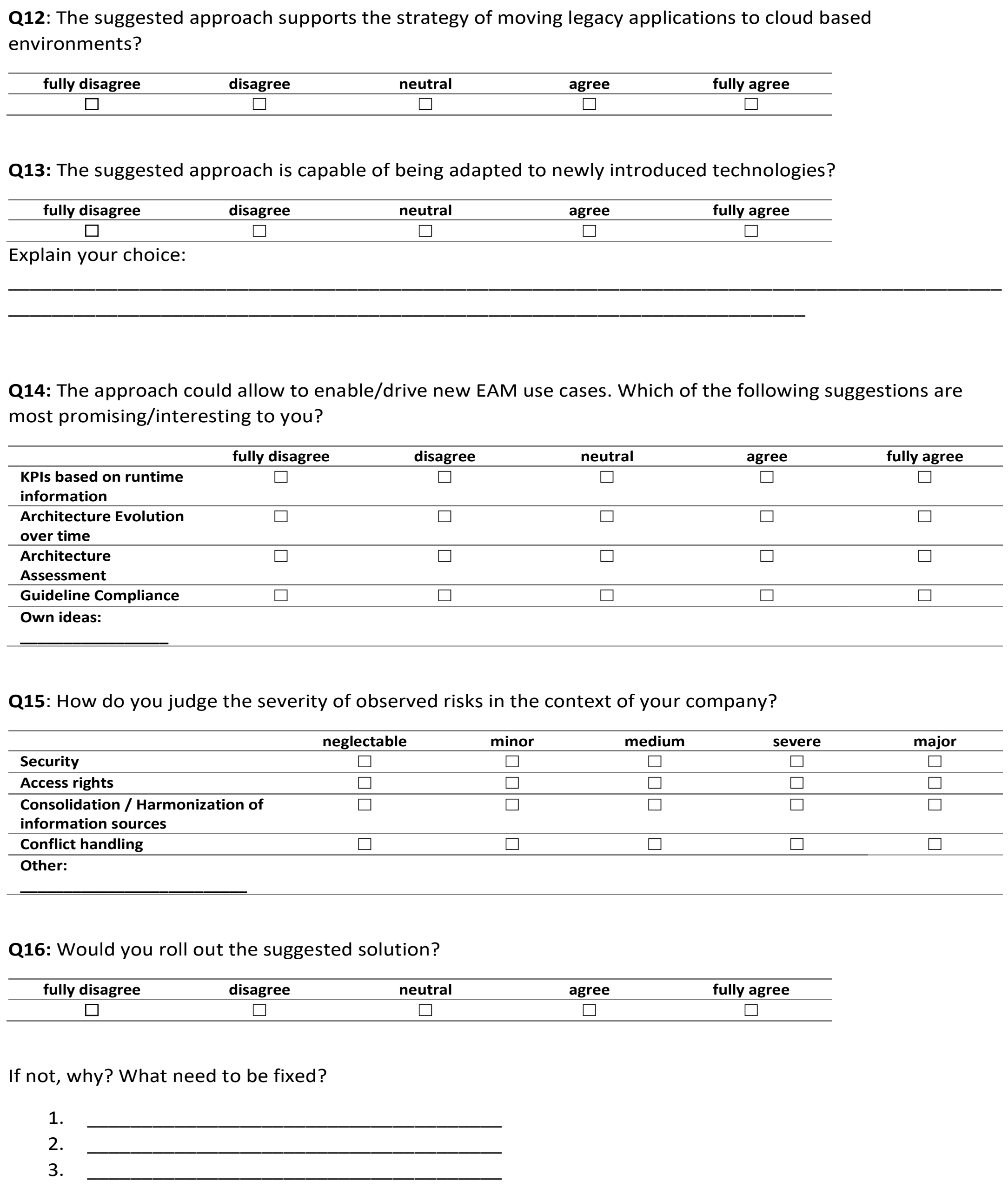
	fully disagree	disagree	neutral	agree	fully agree
reg. organization	☐	☐	☐	☐	☐
reg. your role	☐	☐	☐	☐	☐

Explain your choice:

____

____

Figure A.8.: Part IIb - General feedback about solution approach



Q12: The suggested approach supports the strategy of moving legacy applications to cloud based environments?

fully disagree	disagree	neutral	agree	fully agree
☐	☐	☐	☐	☐

Q13: The suggested approach is capable of being adapted to newly introduced technologies?

fully disagree	disagree	neutral	agree	fully agree
☐	☐	☐	☐	☐

Explain your choice:

____

____

Q14: The approach could allow to enable/drive new EAM use cases. Which of the following suggestions are most promising/interesting to you?

	fully disagree	disagree	neutral	agree	fully agree
KPIs based on runtime information	☐	☐	☐	☐	☐
Architecture Evolution over time	☐	☐	☐	☐	☐
Architecture Assessment	☐	☐	☐	☐	☐
Guideline Compliance	☐	☐	☐	☐	☐
Own ideas: ____________					

Q15: How do you judge the severity of observed risks in the context of your company?

	neglectable	minor	medium	severe	major
Security	☐	☐	☐	☐	☐
Access rights	☐	☐	☐	☐	☐
Consolidation / Harmonization of information sources	☐	☐	☐	☐	☐
Conflict handling	☐	☐	☐	☐	☐
Other: ____________________					

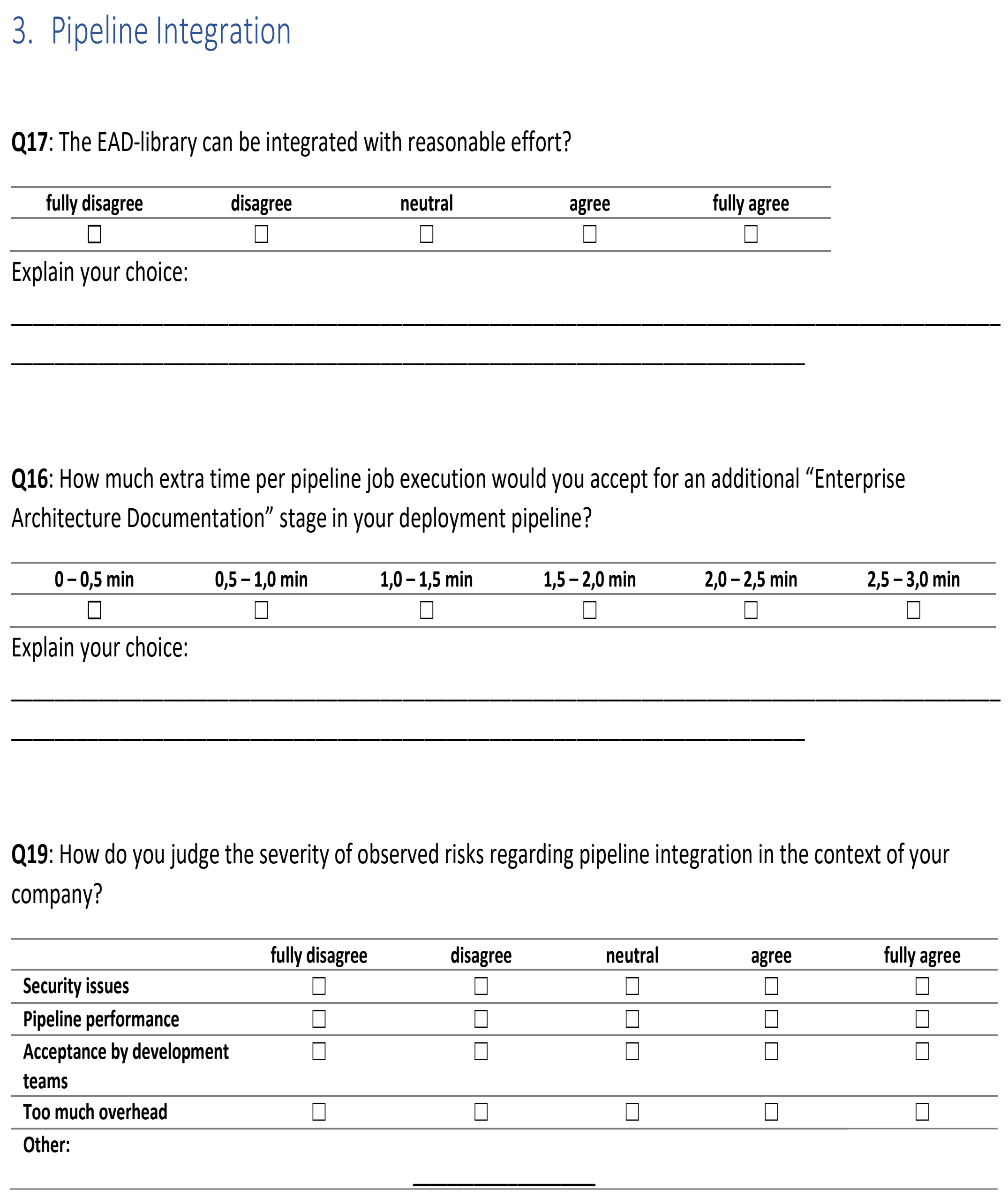
Q16: Would you roll out the suggested solution?

fully disagree	disagree	neutral	agree	fully agree
☐	☐	☐	☐	☐

If not, why? What need to be fixed?

1. ________________________________
2. ________________________________
3. ________________________________

Figure A.9.: Part IIc - General feedback about solution approach



3. Pipeline Integration

Q17: The EAD-library can be integrated with reasonable effort?

fully disagree	disagree	neutral	agree	fully agree
☐	☐	☐	☐	☐

Explain your choice:

____

____

Q16: How much extra time per pipeline job execution would you accept for an additional “Enterprise Architecture Documentation” stage in your deployment pipeline?

0 – 0,5 min	0,5 – 1,0 min	1,0 – 1,5 min	1,5 – 2,0 min	2,0 – 2,5 min	2,5 – 3,0 min
☐	☐	☐	☐	☐	☐

Explain your choice:

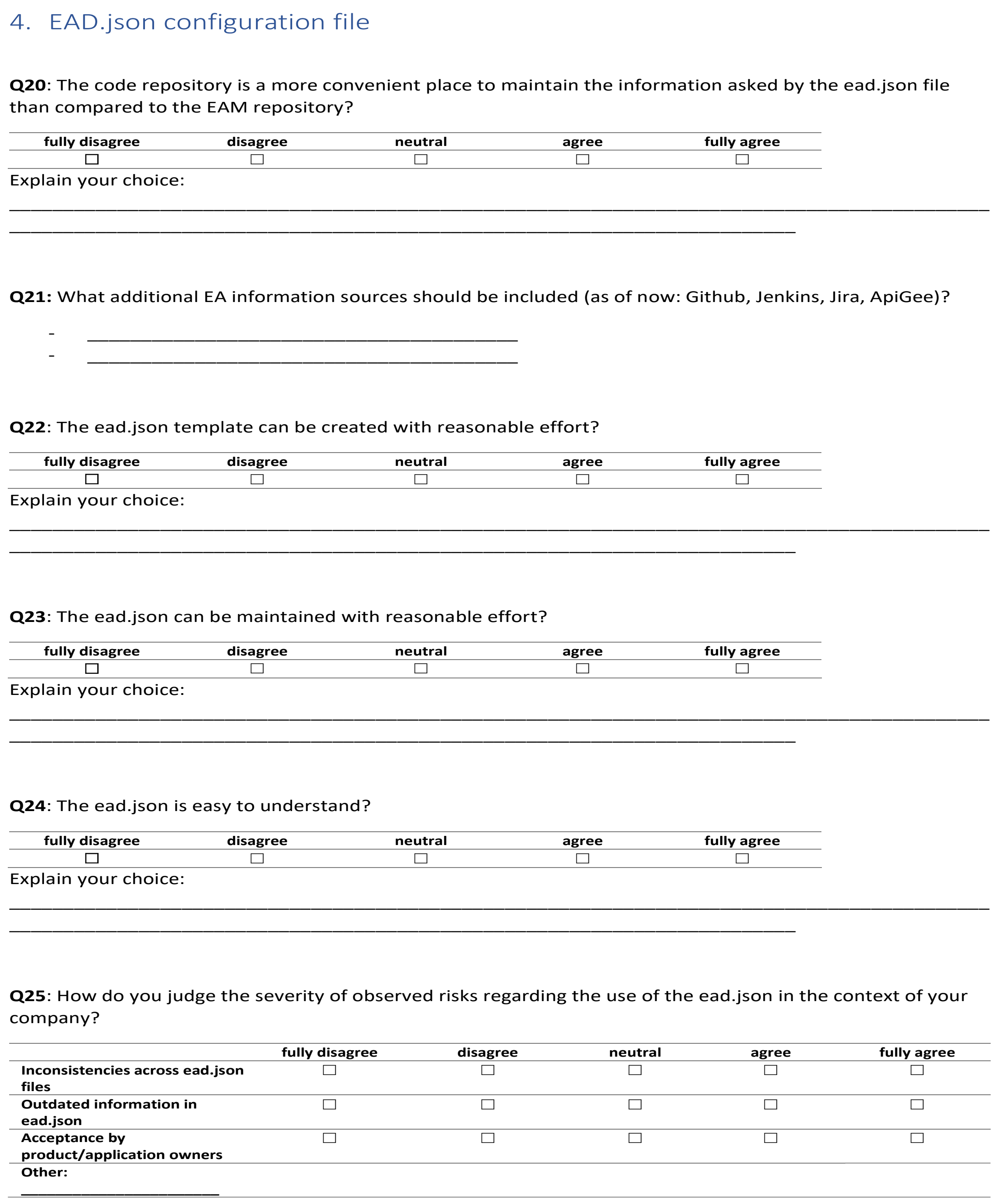
____

____

Q19: How do you judge the severity of observed risks regarding pipeline integration in the context of your company?

	fully disagree	disagree	neutral	agree	fully agree
Security issues	☐	☐	☐	☐	☐
Pipeline performance	☐	☐	☐	☐	☐
Acceptance by development teams	☐	☐	☐	☐	☐
Too much overhead	☐	☐	☐	☐	☐
Other:		______________			

Figure A.10.: Part III - Pipeline Integration



4. EAD.json configuration file

Q20: The code repository is a more convenient place to maintain the information asked by the ead.json file than compared to the EAM repository?

fully disagree	disagree	neutral	agree	fully agree
☐	☐	☐	☐	☐

Explain your choice:
____
____

Q21: What additional EA information sources should be included (as of now: Github, Jenkins, Jira, ApiGee)?

- ___________________________________
- ___________________________________

Q22: The ead.json template can be created with reasonable effort?

fully disagree	disagree	neutral	agree	fully agree
☐	☐	☐	☐	☐

Explain your choice:
____
____

Q23: The ead.json can be maintained with reasonable effort?

fully disagree	disagree	neutral	agree	fully agree
☐	☐	☐	☐	☐

Explain your choice:
____
____

Q24: The ead.json is easy to understand?

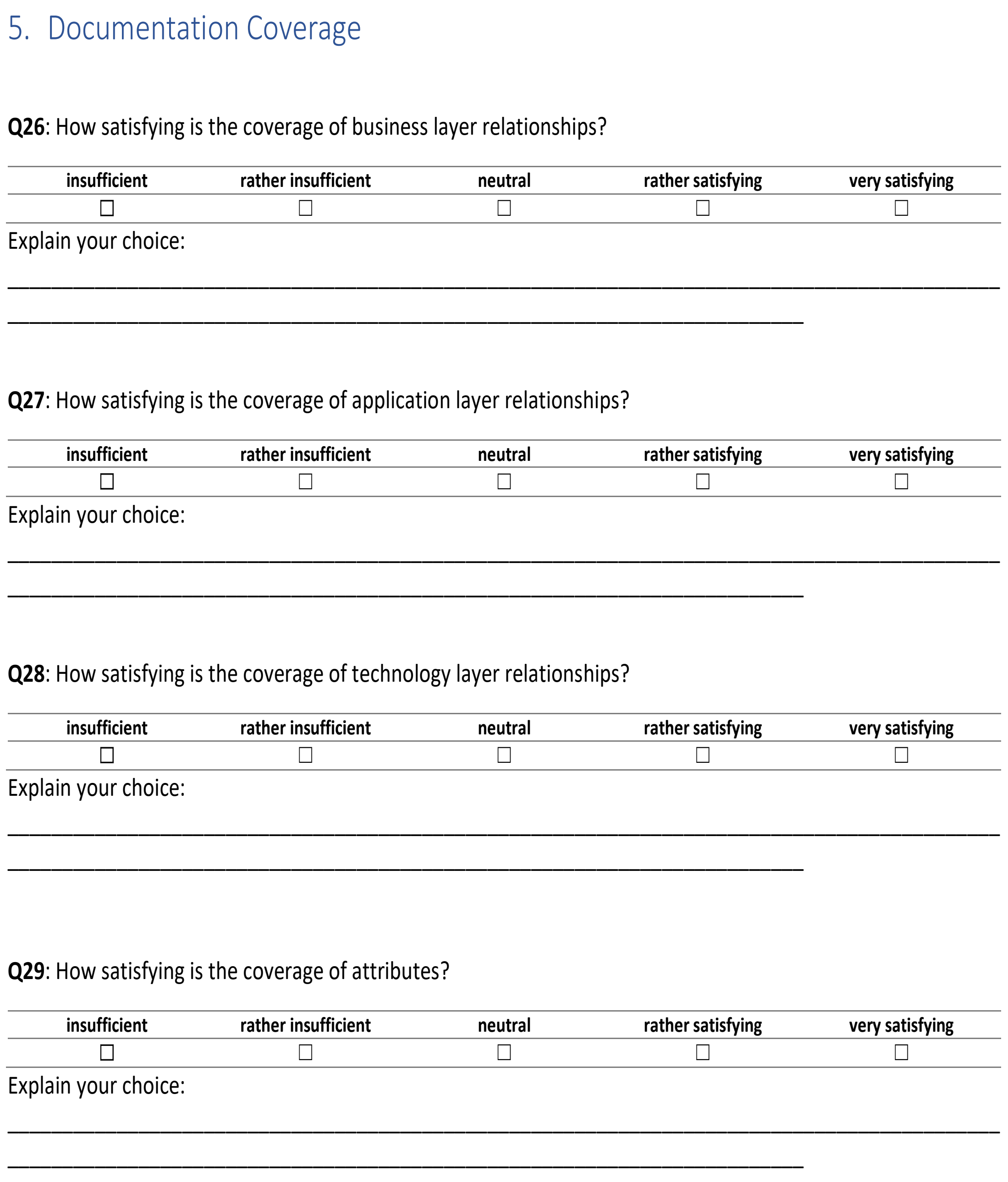
fully disagree	disagree	neutral	agree	fully agree
☐	☐	☐	☐	☐

Explain your choice:
____
____

Q25: How do you judge the severity of observed risks regarding the use of the ead.json in the context of your company?

	fully disagree	disagree	neutral	agree	fully agree
Inconsistencies across ead.json files	☐	☐	☐	☐	☐
Outdated information in ead.json	☐	☐	☐	☐	☐
Acceptance by product/application owners	☐	☐	☐	☐	☐
Other: ________________					

Figure A.11.: Part IV - ead.json configuration file



5. Documentation Coverage

Q26: How satisfying is the coverage of business layer relationships?

insufficient	rather insufficient	neutral	rather satisfying	very satisfying
☐	☐	☐	☐	☐

Explain your choice:

____

____

Q27: How satisfying is the coverage of application layer relationships?

insufficient	rather insufficient	neutral	rather satisfying	very satisfying
☐	☐	☐	☐	☐

Explain your choice:

____

____

Q28: How satisfying is the coverage of technology layer relationships?

insufficient	rather insufficient	neutral	rather satisfying	very satisfying
☐	☐	☐	☐	☐

Explain your choice:

____

____

Q29: How satisfying is the coverage of attributes?

insufficient	rather insufficient	neutral	rather satisfying	very satisfying
☐	☐	☐	☐	☐

Explain your choice:

____

____

Figure A.12.: Part V - EA Documentation coverage

Bibliography

[1] *A guide to the Project Management Body of Knowledge (PMBOK guide), fifth edition*, 5th ed. Newtown Square, Pa.: Project Management Institute, 2013, ISBN: 978-1-935589-67-9.

[2] *ArchiMate 3.0.1 specification: Open group standard*, Fifth edition, ser. The Open Group Series. Zaltbommel: Van Haren Publishing, 2017, ISBN: 9789401802369.

[3] J. Bogner and A. Zimmermann, "Towards integrating microservices with adaptable enterprise architecture," in *2016 IEEE 20th International Enterprise Distributed Object Computing Workshop (EDOCW)*, IEEE, 2016, pp. 1–6, ISBN: 978-1-4673-9933-3. DOI: `10.1109/EDOCW.2016.7584392`.

[4] M. Buschle, M. Ekstedt, S. Grunow, M. Hauder, F. Matthes, and S. Roth, "Automating enterprise architecture documentation using an enterprise service bus," in *AMCIS*, 2012.

[5] B. Cameron. (2018). Common-perspectives-on-enterprise-architecture-final-1-copy, [Online]. Available: `https://feapo.org/wp-content/uploads/2018/10/Common-Perspectives-on-Enterprise-Architecture-Final-1-copy.pdf` (visited on 09/05/2019).

[6] J. S. David, D. Schuff, and R. St. Louis, "Managing your total it cost of ownership," *Communications of the ACM*, vol. 45, no. 1, pp. 101–106, 2002, ISSN: 00010782. DOI: `10.1145/502269.502273`.

[7] M. Farwick, B. Agreiter, R. Breu, M. Häring, K. Voges, and I. Hanschke, "Towards living landscape models: Automated integration of infrastructure cloud in enterprise architecture management," in *2010 IEEE 3rd International Conference on Cloud Computing: A Requirements Analysis based on a Literature Review and an Exploratory Survey*, IEEE, 2010, pp. 35–42, ISBN: 978-1-4244-8207-8. DOI: `10.1109/CLOUD.2010.20`.

[8] M. Farwick, B. Agreiter, R. Breu, S. Ryll, and I. Hanschke, "Requirements for automated enterprise architecture model maintenance - a requirements analysis based on a literature review and an exploratory survey," 2011, pp. 325–337.

[9] M. Farwick, B. Agreiter, R. Breu, S. Ryll, K. Voges, and I. Hanschke, "Automation processes for enterprise architecture management," in *2011 IEEE 15th International Enterprise Distributed Object Computing Conference Workshops*, IEEE, 2011, pp. 340–349, ISBN: 978-1-4577-0869-5. DOI: 10.1109/EDOCW.2011.19.

[10] M. Farwick, R. Breu, M. Hauder, S. Roth, and F. Matthes, "Enterprise architecture documentation: Empirical analysis of information sources for automation," in *2013 46th Hawaii International Conference on System Sciences*, IEEE, 2013, pp. 3868–3877, ISBN: 978-1-4673-5933-7. DOI: 10.1109/HICSS.2013.200.

[11] M. Farwick, C. M. Schweda, R. Breu, and I. Hanschke, "A situational method for semi-automated enterprise architecture documentation," *Software & Systems Modeling*, vol. 15, no. 2, pp. 397–426, 2016, ISSN: 1619-1366. DOI: 10.1007/s10270-014-0407-3.

[12] R. Fischer, S. Aier, and R. Winter, "A federated approach to enterprise architecture model maintenance: 14-22 pages / enterprise modelling and information systems architectures, vol 2, no 2 (2007)," 2015. DOI: 10.18417/EMISA.2.2.2.

[13] G. Granchelli, M. Cardarelli, P. Di Francesco, I. Malavolta, L. Iovino, and A. Di Salle, "Microart: A software architecture recovery tool for maintaining microservice-based systems," in *ICSA 2017*, IEEE, 2017, pp. 298–302, ISBN: 978-1-5090-4793-2.

[14] S. Hacks, A. Steffens, P. Hansen, and N. Rajashekar, "A continuous delivery pipeline for ea model evolution," in *Enterprise, Business-Process and Information Systems Modeling*, ser. Lecture Notes in Business Information Processing, I. Reinhartz-Berger, J. Zdravkovic, J. Gulden, and R. Schmidt, Eds., vol. 352, Cham: Springer International Publishing, 2019, pp. 141–155, ISBN: 978-3-030-20617-8. DOI: 10.1007/978-3-030-20618-5_10.

[15] M. Hauder, F. Matthes, and S. Roth, "Challenges for automated enterprise architecture documentation," in *Trends in enterprise architecture research and practice-driven research on enterprise transformation : 7th Workshop, TEAR 2012, and 5th Working Conference, PRET 2012 ; held at the Open Group Conference 2012, Barcelona, Spain, October 23-24, 2012 ; proceedings*, vol. 131, Springer, 2012, pp. 21–39, ISBN: 978-3-642-34162-5. DOI: 10.1007/978-3-642-34163-2_2.

[16] H. Holm, M. Buschle, R. Lagerström, and M. Ekstedt, "Automatic data collection for enterprise architecture models," *Software & Systems Modeling*, vol. 13, no. 2, pp. 825–841, 2014. DOI: 10.1007/s10270-012-0252-1.

[17] S. Horovitz, Y. Arian, M. Vaisbrot, and N. Peretz, "Non-intrusive cloud application transaction pattern discovery," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, IEEE, 2019, pp. 311–320, ISBN: 978-1-7281-2705-7. DOI: 10.1109/CLOUD.2019.00059.

[18] P. Johnson, M. Ekstedt, and R. Lagerstrom, "Automatic probabilistic enterprise it architecture modeling: A dynamic bayesian networks approach," in *2016 IEEE 20th International Enterprise Distributed Object Computing Workshop (EDOCW)*, IEEE, 2016, pp. 1–8, ISBN: 978-1-4673-9933-3. DOI: `10.1109/EDOCW.2016.7584351`.

[19] P. A. Khosroshahi, M. Hauder, and F. Matthes, "Analyzing the evolution and usage of enterprise architecture management patterns," in *AMCIS*, 2016.

[20] M. Kleehaus, M. Hauder, F. Matthes, and Ö. Uludag, "Enterprise architecture discovery via runtime instrumentation for automating enterprise architecture model maintenance," 2019.

[21] M. Kleehaus, Ö. Uludağ, P. Schäfer, and F. Matthes, "Microlyze: A framework for recovering the software architecture in microservice-based environments," in *Information Systems in the Big Data Era*, ser. Lecture Notes in Business Information Processing, J. Mendling and H. Mouratidis, Eds., vol. 317, Springer International Publishing, 2018, pp. 148–162, ISBN: 978-3-319-92900-2.

[22] H. Krcmar, *Informationsmanagement*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, ISBN: 978-3-662-45862-4.

[23] J. Landthaler, Ö. Uludağ, G. Bondel, A. Elnaggar, S. Nair, and F. Matthes, "A machine learning based approach to application landscape documentation," in *The Practice of Enterprise Modeling*, ser. Lecture Notes in Business Information Processing, R. A. Buchmann, D. Karagiannis, and M. Kirikova, Eds., vol. 335, Cham: Springer International Publishing, 2018, pp. 71–85, ISBN: 978-3-030-02301-0. DOI: `10.1007/978-3-030-02302-7_5`.

[24] P. Measey, Ed., *Agile Foundation: Principles, Practices and Frameworks*. London: BCS, 2015, ISBN: 9781780172569.

[25] F. Montesi and J. Weber, *Circuit breakers, discovery, and api gateways in microservices*, Sep. 19, 2016.

[26] S. Newman, *Building microservices*. Sebastopol: O'Reilly Media, 2015, ISBN: 9781491950357.

[27] M. Panwar, "Application performance management emerging trends," in *2013 International Conference on Cloud & Ubiquitous Computing & Emerging Technologies*, IEEE, 2013, pp. 178–182, ISBN: 978-1-4799-2235-2. DOI: `10.1109/CUBE.2013.40`.

[28] N. Prat, I. Comyn-Wattiau, and J. Akoka, "A taxonomy of evaluation methods for information systems artifacts," *Journal of Management Information Systems*, vol. 32, no. 3, pp. 229–267, 2015, ISSN: 0742-1222. DOI: `10.1080/07421222.2015.1099390`.

[29] I. Red Hat. (2019). What's a service mesh? [Online]. Available: `https://www.redhat.com/en/topics/microservices/what-is-a-service-mesh` (visited on 09/11/2019).

[30] J. W. Ross, P. Weill, and D. Robertson, *Enterprise Architecture As Strategy: Creating a Foundation for Business Execution*. Boston: Harvard Business Review Press, 2014, ISBN: 1-59139-839-8.

[31] S. Rossel, *Continuous Integration*. Birmingham: Packt Publishing, 2017, ISBN: 978-1-78728-418-0.

[32] S. Roth, "Federated enterprise architecture model management: Conceptual foundations, collaborative model integration, and software support," Dissertation, Technische Universität München, München, 2014.

[33] S. Roth, M. Hauder, M. Farwick, R. Breu, and F. Matthes, "Enterprise architecture documentation: Current practices and future directions," in *Wirtschaftsinformatik*, 2013.

[34] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[35] K. Schwaber and J. Sutherland. (2015). The scrum guide, [Online]. Available: `https://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-DE.pdf` (visited on 09/14/2019).

[36] G. O. Smith F. (2018). What is a service mesh? [Online]. Available: `https://www.nginx.com/blog/what-is-a-service-mesh/` (visited on 09/11/2019).

[37] C. Sonnenberg and J. vom Brocke, "Evaluations in the science of the artificial – reconsidering the build-evaluate pattern in design science research," in *Design Science Research in Information Systems. Advances in Theory and Practice*, ser. Lecture Notes in Computer Science, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, K. Peffers, M. Rothenberger, and B. Kuechler, Eds., vol. 7286, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 381–397, ISBN: 978-3-642-29862-2. DOI: `10.1007/978-3-642-29863-9_28`.

[38] *The TOGAF standard*, Version 9.2. Zaltbommel: Van Haren Publishing, 2018, ISBN: 9789401802833.

[39] T. Trojer, M. Farwick, M. Häusler, and R. Breu, "Living modeling of it architectures: Challenges and solutions," in *Software, Services, and Systems*, ser. Lecture Notes in Computer Science, R. de Nicola and R. Hennicker, Eds., vol. 8950, Cham: Springer International Publishing, 2015, pp. 458–474, ISBN: 978-3-319-15544-9. DOI: `10.1007/978-3-319-15545-6_26`.

[40] S. Vadapalli, *DevOps: Continuous Delivery, Integration, and Deployment with DevOps*. [Place of publication not identified]: Packt Publishing, 2018, ISBN: 978-1-78913-299-1.

[41] M. Valja, R. Lagerstrom, M. Ekstedt, and M. Korman, "A requirements based approach for automating enterprise it architecture modeling using multiple data sources," in *2015 IEEE 19th International Enterprise Distributed Object Computing Workshop*, IEEE, 2015, pp. 79–87, ISBN: 978-1-4673-9331-7. DOI: 10.1109/EDOCW.2015.33.

[42] J. Venable, J. Pries-Heje, and R. Baskerville, "Feds: A framework for evaluation in design science research," *European Journal of Information Systems*, vol. 25, no. 1, pp. 77–89, 2016, ISSN: 0960-085X. DOI: 10.1057/ejis.2014.36.

[43] J. Verona, *Practical DevOps, Second Edition: Implement DevOps in your organization by effectively building, deploying, testing, and monitoring code, 2nd Edition*, 2nd ed. Birmingham: Packt Publishing, 2018, ISBN: 9781788398619.

[44] VersionOne. (2019). Collabnet versionone releases 13th annual state of agile report, [Online]. Available: https://www.collab.net/news/press/collabnet-versionone-releases-13th-annual-state-agile-report (visited on 09/24/2019).

[45] N. Villasana Corpancho, "Automated documentation of business domain assignments and cloud application information from an application development pipeline," Master Thesis, 2019.

[46] J. T. Zhao, S. Y. Jing, and L. Z. Jiang, "Management of api gateway based on microservice architecture," *Journal of Physics: Conference Series*, vol. 1087, p. 032 032, 2018, ISSN: 1742-6588. DOI: 10.1088/1742-6596/1087/3/032032.