Institute of Software Engineering

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelor’s Thesis

Architectural Requirements for Image Recognition: An Industrial Case Study

Patrick Reich

Course of Study:	Software Engineering
Examiner:	Prof. Dr. Stefan Wagner
Supervisor:	Markus Haug
Commenced:	May 2, 2023
Completed:	October 31, 2023

Abstract

Communication through construction drawings is a vital aspect of the product development process. Because different companies own the intellectual property rights to different construction drawings, it is necessary to verify the ownership of construction drawings before sending them to other companies.

This thesis addresses the software architecture design process for a system that automates construction drawing ownership verification via artificial-intelligence-based document classification. A focus group identifies a number of architectural requirements for the software, which are used together with the C4 model to develop an architecture proposal. The architecture proposal is evaluated in a series of expert interviews that also identify additional requirements.

The thesis' contributions include a catalog of architectural requirements, together with a prioritization and division into mandatory and non-mandatory requirements by the focus group participants, and an architecture proposal that provides guidelines on how to fulfill most of the requirements. The architecture proposal and requirements are well applicable to related problem statements, such as other classification problems, possibly in contexts other than the product development process. However, because the importance of the architectural requirements is very case-dependent, the prioritization and division into mandatory and non-mandatory requirements is more limited to the use case at hand.

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence. 14, 15, 17, 24, 25, 26, 29, 34, 35, 36, 37, 38, 40, 41, 43, 44, 46, 47, 48, 51, 52, 53, 54, 55, 56, 57, 85

API Application Programming Interface. 34, 37, 38, 40, 41, 42, 43, 45, 46, 48, 50, 51, 54

ATAM Architecture Tradeoff Analysis Method. 30

E2EE End-To-End Encryption. 46, 52, 60

GDPR General Data Protection Regulation. 35

GUI Graphical User Interface. 20, 24, 34, 37, 38, 40, 43, 46, 51, 53, 54

HTTP HyperText Transfer Protocol. 21

IPR Intellectual Property Rights. 7, 13, 14, 15

ISO International Organization for Standardization. 50

MVC Model View Controller. 7, 20, 21, 24, 34, 38, 40, 51, 52, 53

OEM Original Equipment Manufacturer. 13, 14, 15, 26, 29, 34, 40, 56

OS Operating System. 34, 51

PDM Product Data Management. 14

REST REpresentational State Transfer. 34

RQ Research Question. 25

SAAM Software Architecture Analysis Method. 6, 30, 31, 46, 47, 85

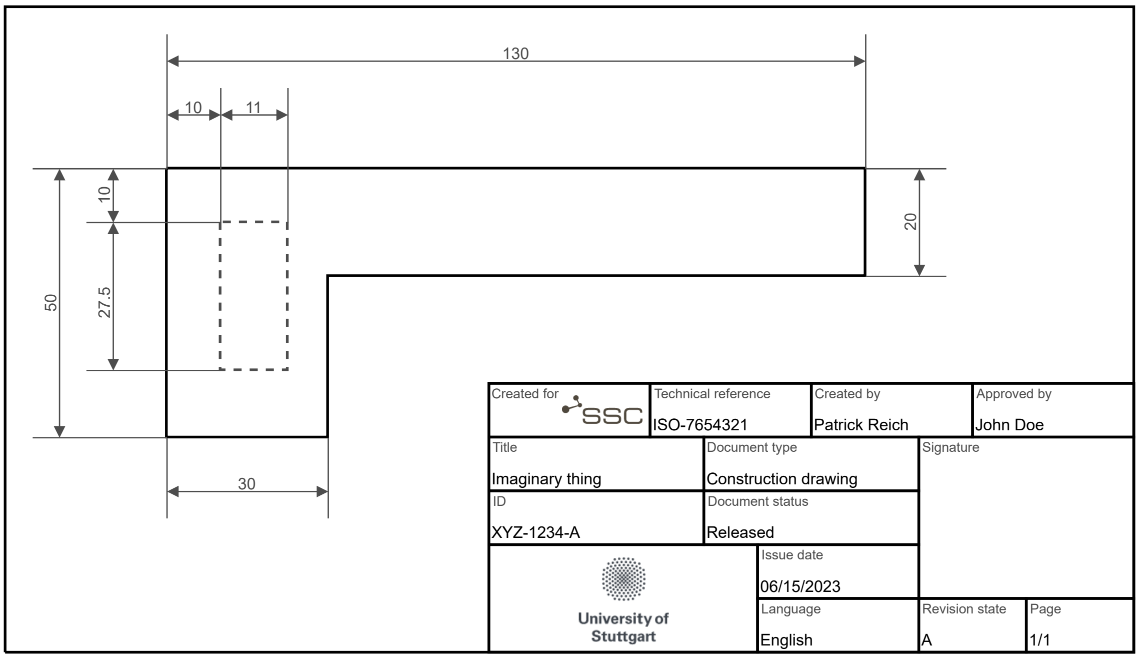
1 Introduction

This chapter motivates the need for the research conducted. This is done by first giving an overview of the case study problem and then defining the concrete contribution of this paper. Finally, the structure of this paper is outlined.

1.1 Motivation

Image recognition is increasingly used as a supporting tool for classification tasks in industrial processes. For instance, in the product development process. In this process, one Original Equipment Manufacturer (OEM), such as a car manufacturer, works together with many supplier companies. A large part of the communication between an OEM and its suppliers takes place in the form of construction drawings. An example of a construction drawing is depicted in Figure 1.1.

Before sending a construction drawing to a supplier, the OEM has to make sure, that it is allowed to send it. This is either the case, if the OEM itself is the creator of the construction drawing, and therefore owns the Intellectual Property Rights (IPR) of the drawing, or it owns a license specifically permitting it to share the construction drawing with others. This is vizualized in Figure 1.2. Supplier



Created for: SSC	Technical reference: ISO-7654321	Created by: Patrick Reich	Approved by: John Doe
Title: Imaginary thing	Document type: Construction drawing	Signature:	
ID: XYZ-1234-A	Document status: Released		
	Issue date: 06/15/2023		
	Language: English	Revision state: A	Page: 1/1

Figure 1.1: Example of a construction drawing with header.

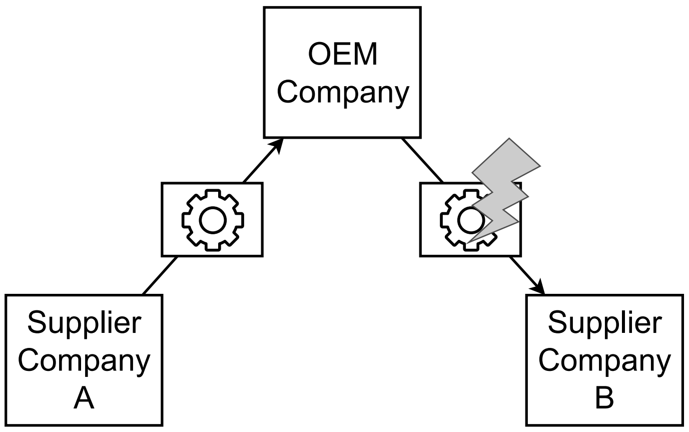


Figure 1.2: Data transfer process of a construction drawing resulting in the violation of IPR.

Company A created the construction drawing, and therefore owns the IPR of the drawing. Thus, it can simply send the construction drawing to the OEM Company. The OEM Company, however, does neither own the IPR, nor a license permitting it to share the construction drawing, and therefore is not allowed to send it to Supplier Company B.

Sending a construction drawing without proper authorization may result in costly penalties. Therefore, it is essential to verify the IPR situation of each construction drawing before sending. This constitutes a classification task. Currently, this is done manually, which is a cost-intensive and error-prone process. A software system could reduce manual efforts by employing automatic classification of construction drawings before transmission. One possibility to perform automatic document classification is via image recognition, which belongs to the domain of Artificial Intelligence (AI).

1.2 Context

An OEM commissioned SSC-Services[1], the company that this thesis is created with, to develop such a software system. The research of this thesis is conducted within the scope of the software development project at SSC-Services, in the form of an industrial case study.

The OEM uses a Product Data Management (PDM) system to store and and keep track of all of its construction drawings. To send the drawings, the OEM uses an application for inter-organizational secure data exchange.

From the viewpoint of the OEM, it is only relevant that a supplier company owns the IPR to the construction drawing they are trying to send, regardless of which supplier company that is. Therefore, the goal of the system is a binary document classification of construction drawings, to the predefined classes of internal or external construction drawings.

[1] https://www.ssc-services.de/ (followed on October 25, 2023)

The OEM works with single-page construction drawings like the one in Figure 1.1, as well as with multi-page construction drawings. These consist of multiple pages, with each page containing a different construction drawing. The IPR of different pages might be owned by different companies. Such multi-page construction drawings therefore can't be classified as a whole, but each page must be classified on its own. Also, as always in the product development process, the highest level of data security must be applied to protect trade secrets.

In the project, the AI algorithm, which does the actual classification of the drawings, already exists and is therefore only a marginal part of this thesis. It implements a similarity search approach, performing comparisons of vector representations of header tables of construction drawings like the one on the bottom right of Figure 1.1. As the position of the header table may vary with different construction drawings layouts, and there may be multiple header tables, a machine learning model is trained to identify the header tables beforehand.

1.3 Contribution

The objective of this thesis is to develop an architecture for such a software system as described in the section above. To this end, it provides two key contributions:

1. A catalog of architecturally significant requirements
2. An architecture proposal based on and fulfilling the requirements

These contributions are derived from an empirical study in the context of the industrial case described above.

1.4 Structure of this Thesis

Chapter 2 covers the theoretical background, that forms the basis of the work of this thesis. In Chapter 3 related work is presented. The main ideas and solutions of these related studies are also briefly presented. Chapter 4 introduces the structure of the case study. The main focus is on the methodology and data analysis. However, the specific circumstances of the case study are also addressed. In Chapter 5, the contributions of the research conducted in the scope of the thesis are presented. These are divided into a catalog of architectural requirements and a specific architecture proposal. The results are discussed in Chapter 6. The focus is on the applicability of the results, their limitations and a number of threats to the validity of the contributions. Lastly, Chapter 7 represents a summary of the thesis and highlights possible future studies.

2 Background

This chapter presents the concepts and theories that form the basis of the results presented in Chapter 5. First, document classification and binary document classification problems are reviewed. The remaining concepts are several machine learning-related topics and an introduction to the concept of patterns in software architecture together with two prominent representatives. The goal of the machine learning-related topics is not to provide a deep understanding of each concept, but rather to offer just enough information about them so that architectural decisions related to specific AI features make sense.

2.1 Document Classification

Alzubi et al. [ANK18] define a classification problem as “a problem in which the output can be only one of a fixed number of output classes known apriori”. If the number of output classes is two, it is called a binary classification problem [SSR+21]. The input to a general classification problem can be any object in the form of a vector, which is then mapped to one of the n classes [Mar11; Raj20; SA13]. In the case of document classification, of course, the input is a type of document, either textual, visual, or any other type.

Since there is no precise definition of “document”, according to Chen and Blostein [CB07], a document classification problem usually defines a so-called document space. They define a document space as a subset of all currently existing and all future documents that specifies which documents are valid input for the instance that solves the document classification problem. For example, the document space could consist of all text documents or, as in the case of this study, all construction drawings.

According to Martens and Provost [MP14], document classification problems are usually solved using some form of AI, such as a machine learning approach. They state that this is essential because most documents lack a consistent structure to enable mathematical or traditional data mining techniques.

2.2 Machine Learning

Problems that can be solved using traditional software engineering approaches are typically unrelated to the context that they occured in and require highly detailed specifications in order to be solved [JLL+22; Par86]. However, there are problems that cannot be specified in such detail, or where context plays a critical role. These are the kinds of problems that need to be solved with an AI approach, e.g. machine learning.

One of these complex problems is detecting an object in an image. Rebala et al. [RRC19] state that it is impossible to provide software with a specification of the object free from context, i.e., to create an exact description of the object that matches every image. However, with a machine learning approach, this problem can be solved. All the machine learning algorithm needs is a dataset consisting of a large number of images and a label for each image indicating whether the object is in the image or not. The dataset primarily represents how the system should behave. The labeled dataset is used by the machine learning algorithm as training data to create a so-called model, which is "a set of rules, [...] that [...] can correctly predict the labels of [images] not in the dataset" [RRC19].

2.2.1 Supervised and Unsupervised Learning

Two primary directions of machine learning algorithms are supervised and unsupervised learning. The key difference between the two is the presence of labels in the training dataset [AAM+20; Raj20; SA13].

Supervised learning creates a mapping function that maps input data to a desired output, as indicated by the label of each input. This function can later be used to map inputs that were not part of the training dataset to the set of desired outputs [Mar11; Raj20].

In contrast, unsupervised learning relies on an unlabeled training dataset. The algorithm detects latent patterns in the training data and applies them to categorize the input. Once trained, an unsupervised learning algorithm can map inputs outside the training data to the categories it identified during the training phase [AAM+20; Mar11].

According to Kohonen et al. [KOS+96] as well as Sathya and Abraham [SA13], using an unsupervised learning approach instead of specifying targets can be preferable in some cases, because the algorithm often detects patterns that were not previously considered.

2.2.2 Active Learning

Settles [Set10] defines active learning as a machine learning algorithm, that "may propose queries, usually in the form of unlabeled data instances to be labeled by an oracle (e.g., a human annotator)". Since it relies on labeled training data, active learning is a supervised learning approach.

It is believed that the accuracy of a model can be increased by strategically selecting training data, e.g., by condensing the data in a region of high uncertainty [SJ06]. Active learning approaches put this strategical selection of training data into practice by actively querying only informative data for labeling [BBL06; SJ06].

Because the majority of queries yield little new information for an active learning system, it is the main task of active learning to construct queries that maximize the information gain of the system [BBL06; DWD+16; RKS11; SJ06].

Annotating training data is usually done manually, and is therefore very expensive. Active learning approaches require significantly less training data compared to models based on ordinary supervised learning, and consequently help to reduce costs [BBL06; Set10].

2.2.3 Feedback-based Learning

Similar to active learning, feedback-based learning relies on humans to manually label data that the system requests. However, feedback-based learning is used to improve the accuracy of a pre-trained model, not to train a model from scratch [MM21].

Feedback requests are typically made only when the system has failed to perform a task with sufficient reliability, as noted by Ponnusamy et al. [PGY+22]. They further state that this approach is typically used in systems with many users, where other continuous improvement approaches are not scalable enough. Since a feedback request is usually sent to the user who triggered it, Ponnusamy et al. [PGY+22] identify the central problem of feedback-based learning as distinguishing between qualitative, correct feedback and feedback from irritated users who just answered anything.

2.2.4 Similarity Search

Similarity search is an approach to processing data using a knowledge base consisting of large amounts of data, very similar to machine learning. According to Blott and Weber [BW97], Echihabi [Ech20], and Echihabi et al. [EZP21], similarity search is useful when classical database search is not sufficient for the complexity of the database records. They define classical database search as a boolean approach: each database entry is either an exact match to the query or not. They state that this approach is ineffective when both the entries and queries become too complex. They further claim that these kinds of problems can be solved with similarity search, which finds objects in a set that are most similar to a query, according to some definition of similarity.

Similarity search can be simplified to nearest neighbor search, where objects are represented by high-dimensional vectors. Using a specified distance measure, such as the Euclidean or Manhattan distance, this method calculates similarity between vectors [BW97; Ech20; EZP21].

Vectors representing objects are generated through a process called feature extraction. This is an application specific process [BW97; GE06]. Each dimension of a vector represents the degree of expression of a feature in the object it represents [GE06]. Features in general should represent objects in such a way that “nearness in the vector space corresponds to some natural notion of similarity between the objects themselves” [BW97].

Vectors representing complex objects, such as construction drawings, often have thousands of dimensions. Echihabi et al. [EZP21] as well as Blott and Weber [BW97] claim that the resource usage required for comparing a query to a large set of vectors is therefore very high. They further claim that there are several approaches to similarity search that greatly reduce the cost of such comparisons, but at the sacrifice of a guarantee that the results are correct.

Similarity search can be used to address document classification problems by estimating the class of an input vector based on the similarity between the input vector and a set of labeled training vectors [CGG+09].

2.3 Software Architecture Pattern

Buschmann et al. [BMR+13] introduce patterns with the following example: When a human works on a problem, they usually don't invent a completely new solution but instead remember that they once solved a related problem and try to apply that solution to the current problem. This leads to the abstraction of the solution of the past problem into a so-called pattern.

Patterns typically contain best practices for solving specific problems. In software architecture, patterns show how design principles can be applied to specific problems. These are usually problems that go beyond the complexity of a single class or instance. Using proven patterns can help avoid common pitfalls in architectural design [KJ13].

According to Buschmann et al. [BMR+13], an architectural pattern consists of three parts: the context, the problem, and the solution. The context describes the general situation in which a problem occurs. The problem specifies what the solution is intended to fix, and the solution, of course, provides guidance on how to solve the problem. It is often a very abstracted architecture that can be applied to the problem like a mold and filled with concrete content.

2.3.1 Model View Controller Pattern

The Model View Controller (MVC) architectural pattern was first introduced for user interface development, implemented with the programming language Smalltalk [SCD06]. Nowadays it is a basic pattern for interactive software applications that contain a Graphical User Interface (GUI) [MQ11] and addresses the problem of how to distribute logic within the application. With MVC a system is divided into three modules: model, view and controller [MQ11]. This division is illustrated in Figure 2.1.

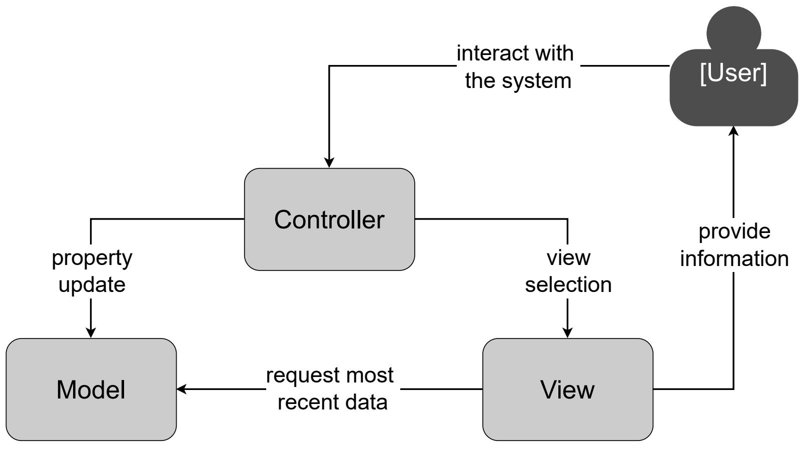


Figure 2.1: Overview of the MVC pattern, adapted in outline from [MQ11]

The model consists of the components that directly address the underlying problem. These are usually as long-lived as the problem itself, and should preferably have nothing to do with the connection to external entities [Dea09]. Depending on the implementation of MVC, the model can be responsible to actively update the information shown by the view components [MQ11].

The view components provide the user with information. Views should be dynamic by design and each view should always have a controller component associated to it [KCK14]. Depending on the implementation of MVC, if the model does not update the view automatically, the view must itself ensure that it always displays the most recent data of the model [Dey11].

The controller accepts user interactions and translates them into requests to the model component [KCK14]. In web applications, user interactions usually are HyperText Transfer Protocol (HTTP) requests, such as GET or POST. Depending on the interaction, the controller may also select a different view, e.g. switch from a request page to the page with the results [Dey11].

2.3.2 Modified MVC

Dey [Dey11] describes a modified MVC architecture where model and view don't communicate with each other directly. Figure 2.2 illustrates the modified message flow. The controller is placed in beetween model and view components. It no longer accepts interactions directly from the user, but receives them through the view. It also receives messages from the model before they are sent to the view. The controller can be considered a broker that processes messages from the outside as well as from the inside [MQ11]. A major benefit of the message flow from the model via the controller to the view is that the navigation and business logic can be well encapsulated in the controller [Dey11].

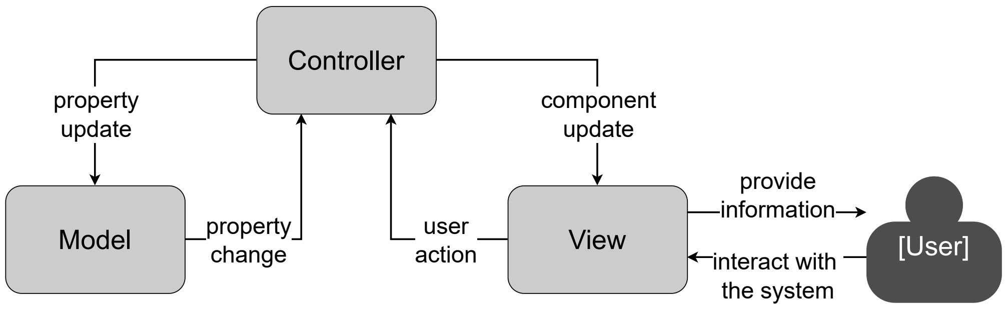


Figure 2.2: Overview of the modified MVC pattern, adopted in outline from [Dey11]

Implementing this modified pattern, Gupta et al. [GG10] argue that it makes the frontend much more lightweight, since the business logic is not as tightly coupled with it as in the original pattern.

3 Related Work

There are many papers on training machine learning models and building processing pipelines for document classification on different document spaces. However, not many papers address the architecture of the software around the machine learning model or pipeline. Since this research has two goals, the collection of a set of architectural requirements and an architecture proposal that shows how to satisfy those requirements, there are also two main topics where related work can intersect. This chapter presents a selection of the papers most closely related to the work of this thesis.

Kazman et al. [KIC05] conducted a "teaching case study" in which they used a method proposed in the same paper, to generate and negotiate requirements and architecture strategies for a distributed, large-scale application, that would be able to handle huge amounts of data and serve many users simultaneously. The details of the system are considerably different from the description of the case of this thesis. Prior to this publication, another case study was conducted by Kazman et al. [KAK01] investigating architecture decisions based on stakeholder requirements for a similar system. An intermediate result of both case studies is a catalog of quality attributes concerning the application. According to the stakeholders questioned, the three most important quality attributes from these catalogs are maintainability, operability and reliability. Further quality attributes presented are performance and scalability. In both case studies, the stakeholders were presented with different architectural strategies to meet the quality attributes, which were also ranked. The different strategies are not further elaborated, as both studies are concerned with the applicability of their methods rather than the actual architecture of their systems.

Rimal et al. [RJKG11] start with a catalog of architectural requirements for cloud computing systems in general, and discuss and assign them to different stakeholder groups. The initial catalog of architectural requirements originates from industry as well as academia, using a method that is not further described and is therefore especially not a well-documented empirical method. The results are three sets of requirements for providers, enterprises, and users, each covering a specific set of issues. The user requirements focus on the user experience and its characteristics; the provider requirements focus on runtime issues; and the enterprise requirements include data security and business-related issues.

In their paper, Ding and Ben Salem [DB18] propose an architecture for automatic document classification in edge computing environments. According to the authors, the proposed three-layer architecture is able to provide real-time document classification at edge servers, close to the endpoint devices where the data is uploaded. Model training and updates are performed centrally via the cloud, based on aggregated data streams provided by the edge servers via feature extraction and feature reduction. The authors argue that their distributed design minimizes the risk of leaking sensitive data by processing data close to where it is uploaded to the system, since security was one of their key requirements. However, there is no in-depth discussion of the architecture design methodology and requirements origins.

Appiani et al. [ACC+01] present a cloud-based system for classifying and then archiving user-uploaded documents in an ordered fashion. At its core, the system uses a decision-tree-based image classification approach capable of processing image data. The architecture of the system implements an object-oriented approach called client/server/database by the authors. The server is the main part of the architecture and is structured in a modular design that allows high scalability and basic functionality of the system even when not all modules are running.

This thesis differs from the two papers addressed in the two paragraphs above insofar as both focus on the classification algorithm as strongly as on the architecture. The former architecture relies heavily on the distributability of its classification algorithm, and the latter may also be less generalizable than the work presented in this thesis due to the specification of the classification algorithm.

Several other papers address the architectural design of AI-related cloud applications in various use cases. When comparable, their results are often similar to the ones of this thesis, but none of the studies use empirical methods to evaluate their architecture design. Lai et al. [LTWL10] focused on the creation of a social network where “friendship” relations are based on a face recognition model. They developed a centralized architecture proposal, whereby the face recognition AI algorithm is isolated from the rest of the system and can be accessed via a SOAP web service. The authors do not specify which architecture pattern they used, although it appears to be a variation of the MVC architectural pattern. Jung and Joe [JJ23] propose an architecture for a cloud-based mobile application that provides users with situation-aware travel plans through an AI-based recommendation algorithm. According to the authors, the architecture comprises a modular integrated framework based on a plug-in architecture method, making it highly scalable. Hendradi et al. [HKM19] propose an architecture for cloud-based e-learning systems that make use of AI. Their work is founded on a literature review of papers on architectures for other cloud-based e-learning systems. The architecture features different user roles and mobile support for the GUI, among other things.

In summary, to the knowledge of the author, no previous study has addressed the full process of gathering architectural requirements, the development of an architecture proposal, and then the evaluation of the architecture proposal using empirical methods. The previous studies utilizing empirical methods for architectural requirements generation primarily concentrate on the methodology itself, while failing to report on solutions for fulfilling the requirements. On the other hand, the studies that focus on providing an architecture proposal don’t report on how they obtained their requirements or use empirical methods to evaluate their results. However, the results of this thesis are reasonably consistent with the results of both types of studies.

4 Case Study

This chapter focuses on the design and execution of the case study. It covers the specific research questions, how the research is conducted, i.e., what research methods are used, and also how the data that resulted from the research methods will be analyzed.

4.1 Research Questions

Since the task of this thesis can be divided into two parts, the collection of architectural requirements and the fulfillment of these, there will be also two Research Questions (RQs).

RQ1: Which architectural requirements exist for an AI-based image recognition software to classify construction drawings before transmission?

RQ2: How are these architectural requirements to be fulfilled?

The first question aims to collect architectural requirements as defined by stakeholders of the product. The second question aims to develop a well-founded architectural proposal that fulfills the previously defined requirements.

4.2 Methodology

The methodology can be divided into the four main steps depicted in Figure 4.1. First, a focus group is conducted with the goal of generating architectural requirements for the software. Then, based on these requirements, an architecture proposal is developed, which is evaluated in the third step via expert interviews. Further architecture requirements could also be identified in this stage.

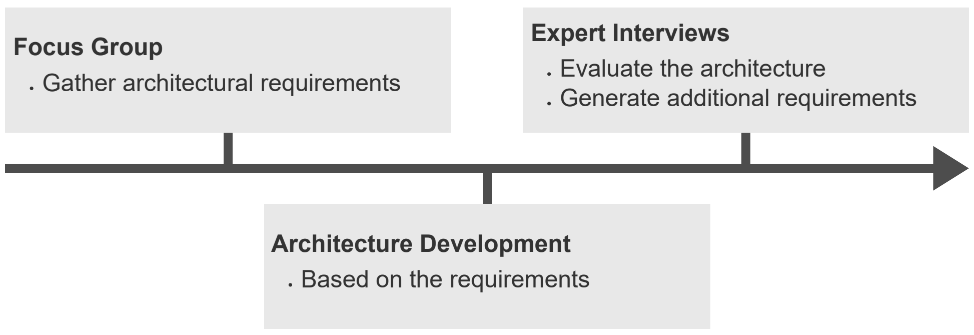

Focus Group
. Gather architectural requirements
Expert Interviews
. Evaluate the architecture
. Generate additional requirements
Architecture Development
. Based on the requirements

Figure 4.1: Methodology overview

4.2.1 Focus Group

This method aims to answer the first research question, as defined in Section 4.1. According to Kontio et al. [KBL08] the focus group's "interactive nature [...] encourage[s] [...] participants to react to points during discussion, reflecting and building on each other's experiences". In the case of requirements generation, it can be assumed that "building on each other's experience" and possibly also building on each other's ideas leads to the requirements being largely compatible with each other. Collecting the requirements of each participant individually, e.g. by conducting a series of interviews, would result in multiple sets of requirements, possibly conflicting on several aspects. Since the merging of the different requirements of the different experts, i.e., making them compatible with each other, should also be done with empirical methods, the focus group method was chosen because it combines the steps of generating requirements and merging them.

Role	Number of participants
Architecture Expert	1
AI Expert	2
OEM employees	3
Product Owner	1
Operator	1

Table 4.1: Participants of the focus group

The focus group is performed with eight participants who have the positions described in Table 4.1. Since the OEM is an important stakeholder in the development of the software, three of its employees participate in the focus group. The other participants are selected to provide a wide range of experience and as many different perspectives as possible. The participants' roles are defined by their daily work, not necessarily by their studies or other certifications. Thus, the architecture and AI experts are confronted with architecture and AI, respectively, in their daily software engineering business. Furthermore, the product owner holds this role not only for the software project of the system that is the subject of this case study, but also for others. The operator is the person who will later operate the system, a task he performs for several other systems as well. The employees of the OEM are grouped together, since all other roles are filled by employees of the company developing the software. The OEM employees all work in different areas of software engineering.

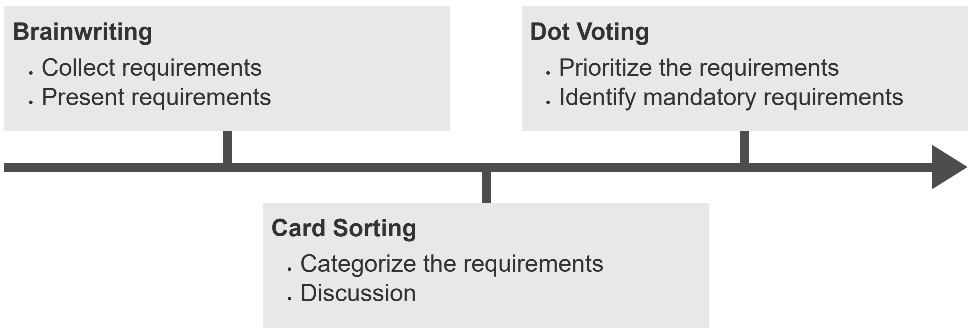

Brainwriting
. Collect requirements
. Present requirements
Dot Voting
. Prioritize the requirements
. Identify mandatory requirements
Card Sorting
. Categorize the requirements
. Discussion

Figure 4.2: Overview of the focus group method

The focus group is conducted using a mix of different methods, as depicted in Figure 4.2. It takes place on a virtual whiteboard called Miro[1], since the participants work in different locations, far away from each other. Furthermore, a digital whiteboard provides greater oversight for the number of participants compared to a physical one. The guide used by the moderator can be found in Appendix A. The data analysis of the focus group is discussed together with the expert interview data analysis in Section 4.3, as both follow a similar procedure.

The focus group consists of three main steps. The first step aims to collect numerous architectural requirements for the software and makes use of a method called Brainwriting. “Brainwriting is the silent, written generation of ideas by a group of people.” [Van84]. The method is considered to be appropriate to generate ideas in an exploratory manner. This approach is used as it allows all participants to find their own ideas without being biased by having heard the ideas of others beforehand. It is hoped that such an approach will yield the most diverse results. The participants are given 10 minutes to silently think of architectural requirements and note them on Post-Its. Afterwards, everyone presents their results in the plenum.

The second step is to cluster the collected requirements. For this purpose, a method called Card Sorting [Zim16] is used. The participants are proposed with the question “How would you categorize the architectural requirements?”. According to Wood et al. [WW08], it is supposed to be better to provide participants with explicit instructions than just leaving them to the task in fear of adding bias by saying anything. After all requirements are distributed, each category should be assigned a descriptive name. This method is used as it encourages discussion. Some requirements might not be clearly specified and become clearer as a result, or there might be conflicting requirements that now need to be made compatible.

The third step focuses on prioritizing the requirements. To achieve this, a method called Dot Voting is used. For this technique, each participant receives the same number of sticky points. These are then distributed among the individual requirements [Dal19]. The dots have a different color for each participant, so that the distribution of votes of each participant can also be analyzed afterwards. The distribution of points is performed with the key question “How important do you think each of the requirements is?”. Prioritizing the requirements is essential to the later development of an architecture proposal, as it helps identify which requirements are more important than others, and provides a sequence in which the requirements should be addressed.

As a final step, the participants are asked to identify requirements that they believe must be taken into account in the resulting system. For this purpose, the requirements are sorted in descending order according to the votes received in the previous step. Then each participant should draw a line separating the mandatory from the non-mandatory requirements. The mandatory requirements can serve as a starting point for the later development of the architecture proposal.

4.2.2 Architecture Proposal

To create the architecture proposal, Brown’s C4 model [Bro15] is utilized. The C4 model proposes a reference model consisting of four different elements, which are defined below.

Person represents one of the human users of the software system.

[1] https://miro.com/whiteboard/ (followed on July 5, 2023)

Software System is the highest level of abstraction describing something that provides value to its users (human or non-human). This can refer to the software system being modeled as well as external software systems that have some relation to the modeled system.

Container is a separately deployable part of a software system. Containers often run in their own process space, but this does not always have to be the case.

Component is a "grouping of related functionality encapsulated behind a well-defined interface" [Bro15]. Multiple components form a container and therefore cannot be deployed individually.

To model interactions between different elements, the C4 model proposes uni-directional relationships. The meaning of these relationships is left open for individual interpretation. In this thesis, interactions between different elements are visualized via information flow by default. To keep the diagrams as simple as possible, some interactions are represented via access relations, which minimize unnecessary overhead through obvious information flow by combining multiple information flow relations into one access relation. These unidirectional access relations are directed in the direction of the initial request.

The C4 model proposes four different diagram types, called views, to model the system at different levels of abstraction. The four views are defined below.

System context view provides a high-level overview of the system, its users, and the other software it interacts with.

Container view shows an expanded version of the system from the system context view. The most detailed elements of this diagram are the containers that make up the system.

Component view is a decomposed view of one of the containers from the container view. The most detailed elements of this diagram are the components that make up the container.

Code diagram view defines the code implementation of a component, usually in the form of a UML class diagram or similar reference model.

For simplicity, only the system context, container and component view are developed. According to Brown [Bro15], the code diagram view is only recommended for highly complex components. It involves too many implementation specific details, which would in this case compromise the generalizability of the architecture. The code diagram view is thus omitted.

"Design is the activity of transforming requirements specifications into a technically feasible solution" [Alb03]. Therefore, the primary focus of designing the proposed architecture is to translate the architectural requirements from the focus group into diagrams as defined previously. After the expert interviews, the architecture is refined by applying the expert's requirements to the diagrams as well.

4.2.3 Expert Interviews

The purpose of this method is to evaluate the architecture proposal that was created previously. It also serves as a second instance for generating further architecture requirements. A total of three expert interviews will be conducted, each with one of the interviewees listed in Table 4.2. The expert interview data analysis is discussed together with the data analysis of the focus group in Section 4.3, as both follow a similar procedure.

The roles of the participants listed in Table 4.2 are derived from their everyday work, i.e. especially not from their studies or other certificates. Therefore, the architecture and AI experts are software engineers with a focus on software architecture and AI development, respectively. The product owner as defined in the SCRUM process by Schwaber et al. [Sch97] is also part of the software engineering process. Therefore, the participant's experience as a product owner also represents their experience in software engineering in general.

The participants were selected for their diverse areas of expertise. The product owner was interviewed due to the importance of the OEM-clients' opinion in this project, with the product owner representing their collective view.

ID	Role	Years of experience			Months in the project
		Software Engineering	Software Architecture	AI	
AE	Architecture-expert	7	5	0	0
AI	AI-expert	5	2	3	3
PO	Product Owner	15	0	2	48

Table 4.2: Participants of the expert interviews

There are multiple types of interviews, including structured and semi-structured interviews. Seaman [Sea99] as well as Runeson and Höst [RH09] define structured interviews as consisting of a predefined set of questions that are asked exactly as written. These questions may be open-ended, but the interviewee only answers one question before the next is asked. In contrast, they claim that semi-structured interviews follow more of a conversational flow. They are also based on a set of questions, but do not have to follow them strictly.

Sometimes also called unstructured, semi-structured interviews may be not even based on a set of pre-formulated questions, but on the general interests of the researcher [RH09].

Following the example of Demirsoy et al. [DP18] who used expert interviews for evaluation purposes as well, the expert interviews for this case study adopt the semi-structured approach. For this purpose, an interview guide containing a set of topics to be covered during the interview was prepared. The guide can be found in Appendix B. As an interview guide should be used by the interviewer to steer the interview [Sea99], the guide for this series of expert interviews contains, in addition to the topics that should be discussed, the entire interview procedure, which is discussed later in this section.

There is also an interview preamble and a document with materials, which the interviewees receive before the interview. The preamble (Appendix C) summarizes general guidelines for the interviews and outlines the procedure. It is based on the example of Bogner et al.[2] and was written in German, since the expert interviews are conducted in German as well.

Since the limited time available in an interview can quickly become a problem [MN07], the interviewees receive the material on which the interviews are based in advance. The material can be found in Appendix D, and consists of the architectural diagrams and associated explanations of the architectural proposal. The explanations are in English, because this is how they were developed, and translating them into German could lead to ambiguities, which are avoided this way.

The focus group generated many different quality attributes on top of the functional requirements. The expert interviews thus need to evaluate these as well. For this purpose, evaluation methods such as Software Architecture Analysis Method (SAAM) [KBAW94] or Architecture Tradeoff Analysis Method (ATAM) [KKC00] could be used. Since ATAM "requires detailed technical knowledge" [IHO02], SAAM was chosen because it can be performed by any of the interviewees. As mentioned above, lack of time is a common pitfall in expert interviews [MN07]. Therefore, only three of the quality attributes from the focus group are evaluated. Since they were ranked in terms of importance during the focus group, the three most important are selected for evaluation.

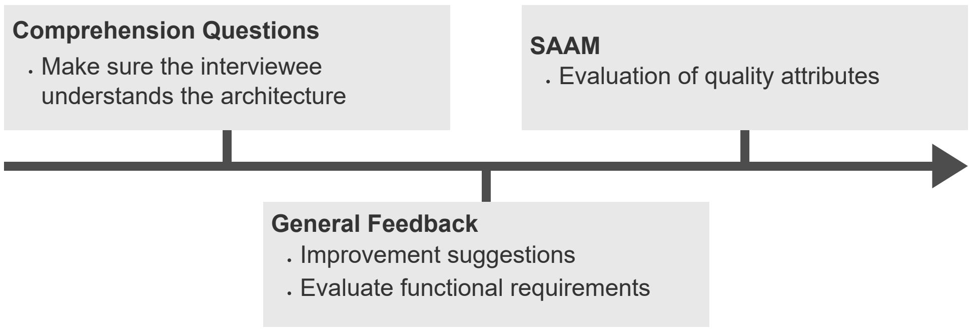

Comprehension Questions
. Make sure the interviewee understands the architecture
SAAM
. Evaluation of quality attributes
General Feedback
. Improvement suggestions
. Evaluate functional requirements

Figure 4.3: Overview of the expert interview procedure

The expert interviews are conducted following the steps in Figure 4.3. Before beginning with the method, the participants are asked about their experience and time in the project. After that, the interviewee is allowed to ask comprehension questions about the architecture proposal to avoid misunderstandings later on. Then the first part of the evaluation begins, in which the interviewer asks for general improvement suggestions. In a semi-structured interview approach, the fulfillment of the functional requirements from the focus group and the general applicability of the architecture are discussed. Afterwards, SAAM is used to evaluate some of the numerous quality attributes collected during the focus group. Finally, the interviewee is informed of the next steps.

SAAM proposes a five-step approach for evaluating quality attributes [KBAW94]. The expert interviews use only steps four and five because the architecture is already in a state where it can be evaluated and the set of quality attributes is predefined. The procedure in the interviews is therefore

[2] `https://github.com/xJREB/research-microservices-interviews/blob/master/interview-preamble.md` (followed on September 12, 2023)

as follows: The interviewee is asked to “choose a set of concrete tasks which test the desired quality attributes”[KBAW94] and “evaluate the degree to which [the] architecture provides support for each task”[KBAW94].

To provide a common ground for the qualitative content analysis, one of the tasks for each quality attribute for SAAM is provided by the interviewer. The interviewee is asked to think of 1–2 additional scenarios for each quality attribute. The given scenarios are the same for each interviewee to ensure comparability in case different interviewees rate the same attribute differently.

Nasar [Nas23] suggests that virtual interviews, while as comfortable as physical ones, are less suitable for gaining a complete understanding of a software system. Since this is necessary to properly evaluate such a system, the expert interviews are all conducted in a physical environment.

4.3 Data Analysis

The data analysis of both the focus group and expert interviews is largely alike and is therefore discussed together. The focus group and each interview are audio recorded. Before the qualitative content analysis begins, the audio recording is fully transcribed and the transcript is given to the participants for their confirmation. It is recommended that qualitative content analysis is conducted on a full transcript as this provides “the most desirable form of data” [Har89; May94]. The focus group analysis uses the miro board, which is used during the focus group for visualization purposes, as additional material.

Following the example of Magenheim et al. [MNR+10], the large text material of the transcripts is reduced in a way that no important information is lost. The main steps include summarizing, explicating, and structuring, as proposed by Mayring [May94]. Explicating is particularly useful for the focus group qualitative content analysis because it allows for easily combining information from the transcript and the miro board.

The focus group’s architectural requirements are combined in a method called cluster analysis [WW08], resulting in a hierarchical tree diagram. For this purpose, similar requirements are bundled [May94], supercategories are constructed [May94] and qualitative attributes are selected [May94] from colloquial paraphrases. For bundled architectural requirements, their votes from the dot voting phase are also combined. The threshold for mandatory requirements established by the participants during the last phase of the focus group remains in a position, so that no former mandatory requirement becomes non-mandatory and no former non-mandatory requirements become mandatory due to bundling.

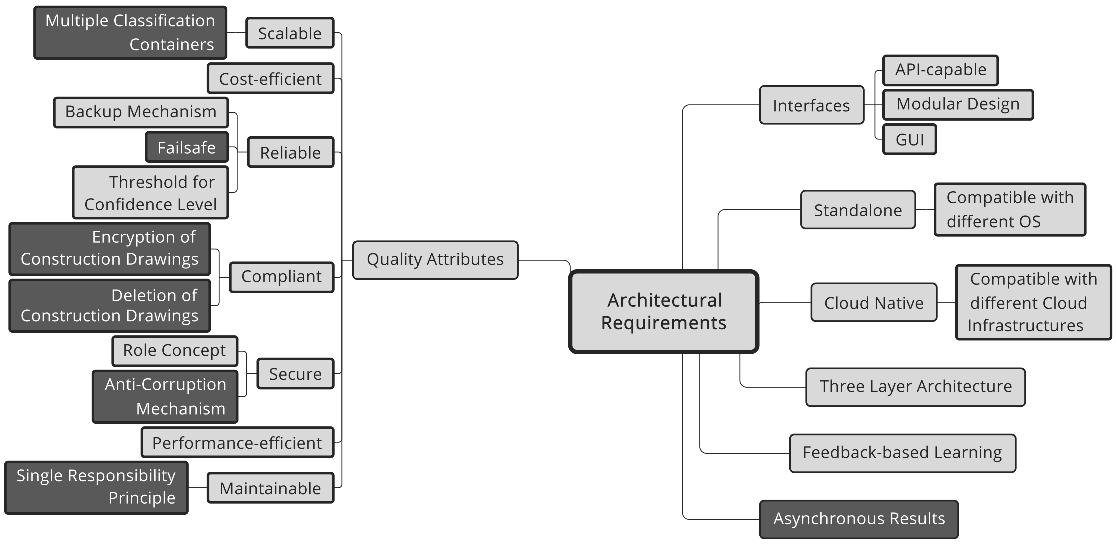
The interview transcripts are analyzed for architectural requirements as well. Requirements identified by multiple interviewees are bundled [May94] and added to the tree diagram. Furthermore, improvement suggestions are identified, and an improvement plan is constructed [May94]. General concerns of the experts are also extracted and to be considered in the improvement plan, before it is implemented.

5 Results

This chapter presents the two contributions of this research. These are, firstly, a set of architectural requirements for a software system as described in Section 1.1, and secondly, a concrete architectural proposal developed based on said requirements. Section 5.3 highlights further issues and solution approaches identified by the experts as well as the evaluation of certain quality attributes during the expert interviews.

5.1 Architectural Requirements

This thesis has generated architectural requirements in two different phases. Most requirements were identified through the dedicated focus group, with the expert interviews expanding the list. Figure 5.1 gives an overview of all architectural requirements for the software. Requirements from the expert interviews are highlighted with a dark background. The structure of the requirements tree was derived from the results of the card sorting phase of the focus group. Originally, the participants collected 59 architectural requirements during the brainwriting phase of the focus group. Since they were asked to write down their requirements individually, most of the requirements appeared multiple times or were similar to others. Therefore, when creating the requirements tree, duplicate requirements were removed, complementary requirements were combined, and in some cases new supercategories were added to provide a better structure. The requirements from the expert interviews were added to the tree based on the context in which they were mentioned.



Architectural Requirements
Quality Attributes
Scalable
Multiple Classification Containers
Cost-efficient
Reliable
Backup Mechanism
Failsafe
Threshold for Confidence Level
Compliant
Encryption of Construction Drawings
Deletion of Construction Drawings
Secure
Role Concept
Anti-Corruption Mechanism
Performance-efficient
Maintainable
Single Responsibility Principle
Interfaces
API-capable
Modular Design
GUI
Standalone
Compatible with different OS
Cloud Native
Compatible with different Cloud Infrastructures
Three Layer Architecture
Feedback-based Learning
Asynchronous Results

Figure 5.1: Tree of architectural requirements

There are a number of functional requirements on the right-hand side in Figure 5.1. Detailed explanations of these requirements can be found in Table 5.1. During the focus group, many quality attributes were mentioned but rarely specified. Definitions of these quality attributes can be found in Table 5.2. Especially during the expert interviews, more specific requirements for some of these quality attributes became evident. Detailed explanations are given in Table 5.3.

ID	Requirement	Description
FR1	API-capable	The application should provide one Application Programming Interface (API) that can be used to connect different systems. A standard REpresentational State Transfer (REST) interface was mentioned during the focus group.
FR2	Modular Design	"Allows use in an interactive mode as well as in an automated chain of different modules. Enables the software to be deployed in the context of an OEM client infrastructure" as defined by the product owner during the focus group.
FR3	GUI	The software should have a GUI that enables the user to perform all necessary actions. Furthermore, some kind of error detection should be possible via the GUI.
FR4	Compatible with different OS	There should be a standalone solution that runs entirely on the users PC and should therefore be compatible with different Operating Systems (OS).
FR5	Compatible with different Cloud Infrastructures	There should be a cloud native solution, that should be compatible with different cloud infrastructures. During the focus group, Azure[1] and AWS[2] were specifically mentioned.
FR6	Three Layer Architecture	MVC architecture as defined by [Dea09].
FR7	Feedback-based Learning	The user should be able to declare a decision of the AI model as wrong or right, which allows the model to evolve.
FR8	Asynchronous Results	For human users only, results should be saved if the user logs out before the result is ready. The user should be able to retrieve the result the next time they log in.

Table 5.1: Clarification of the architectural requirements

The results of the Dot Voting phase of the focus group were used to create a prioritization of requirements. Here, the votes of the requirements that were combined for Figure 5.1 were also summarized. The result of this process is shown in Figure 5.2. Here, the requirements are listed on the vertical axis. Requirements that did not receive any votes are not listed for simplicity. The horizontal axis indicates the number of votes received during the dot voting.

The classification into mandatory and non-mandatory requirements, which was performed as the last step of the focus group, was already relatively unanimous in the focus group. Two participants also classified modular design as a mandatory requirement. However, since the other six participants

[1]https://azure.microsoft.com (followed on July 11, 2023)
[2]https://aws.amazon.com (followed on July 11, 2023)
[3]https://gdpr-info.eu/ (followed on July 11, 2023)

ID	Requirement	Description
QA1	Scalable	"The ability of a system to accommodate an increased workload" [WG06]
QA2	Cost-efficient	High cost-efficiency - efficiency as defined in [ISO11]. During the focus group, it was specifically mentioned that the software should be "commercial license- and fee-free".
QA3	Reliable	High reliability as defined in [ISO11].
QA4	Compliant	Compliant to the General Data Protection Regulation (GDPR)[3]. During the focus group, the architecture expert stated, that "it should be possible to avoid storing user data at all in applications of this kind". Furthermore, construction drawings given as input to the system should not be shared with third parties.
QA5	Secure	The application should not be openly accessible. To achieve this, single sign-on was specifically requested during the focus group.
QA6	Performance-efficient	High performance efficiency as defined in [ISO11].
QA7	Maintainable	High maintainability as defined in [ISO11]. During the focus group, a good documentation, the modifiability of the AI model, the use of a common programming language and existing technologies were specifically mentioned.

Table 5.2: Clarification of the quality attributes among the architectural requirements

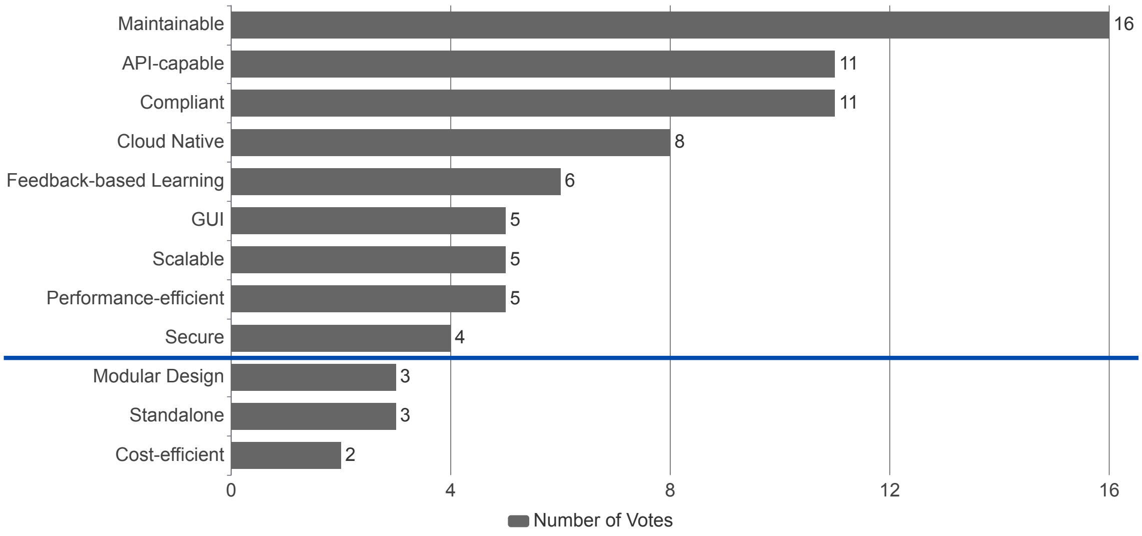


Figure 5.2: Prioritization of focus group results

were in agreement, their classification is shown as a blue line in Figure 5.2. All mandatory requirements are listed above the separator, and all non-mandatory requirements are listed beneath it. All requirements that did not receive any votes would be listed at the bottom, so they are considered non-mandatory as well.

ID	Requirement	Description
QR1	Multiple Classification Containers	The architecture should support a variable number of instances of the AI Classification Algorithm.
QR2	Backup Mechanism	Defined during the focus group as: "If a fatal error occurs, some power user can temporarily disable the automatism, so that users are forced to perform the classification manually".
QR3	Failsafe	The system should still be able to provide a result, even if any part of the system crashes during the process.
QR4	Threshold for Confidence Level	Defined during the focus group as a threshold that must be exceeded by the confidence level in order to be able to declare with sufficient certainty that a drawing doesn't belong to a supplier. If the confidence level is not high enough, the user should be informed.
QR5	Encryption of Construction Drawings	Construction drawings must be encrypted while inside the system, except inside the AI algorithm, where they have to be in plain to run the classification.
QR6	Deletion of Construction Drawings	After the classification process is finished and the user received the result, all sensitive data, i.e. the construction drawing, must be removed from the system.
QR7	Role Concept	Users should be divided into different groups with different permissions in the software. Ordinary users and Administrators were specifically mentioned during the focus group.
QR8	Anti-Corruption Mechanism	Since the architecture should allow the user to increase the classification accuracy, there must be a mechanism to prevent users with malicious intent from decreasing it.
QR9	Single Responsibility Principle	Each component of the architecture should serve only a single purpose. [Hut09]

Table 5.3: Clarification of the architectural requirements refining the quality attributes

5.2 Architecture Proposal

Based on the architectural requirements presented in the previous section, an architecture proposal was created, which is explained in detail in this section. Most of the decisions made during the development are justified directly by one or more architectural requirements, some are a result of the feedback from the expert interviews. The diagrams as well as the explanations in this section relay on data types explained in the following.

Construction Drawing A construction drawing is uploaded by a user or an External Software, who then awaits its classification.

Multi-page Construction Drawing A multi-page construction drawing consists of at least two pages, each containing a different construction drawing, possibly of different creator companies. Therefore, each page must be classified individually.

Feedback Feedback is given by the user, either in response to a feedback request from the system, or voluntarily, e.g., if a classification is incorrect. Feedback consists of a construction drawing and a simple boolean, stating that the construction drawing is either a supplier drawing or not.

Feedback Requests Feedback requests are created by the Classification Interpreter (Figure 5.7) if the confidence level of a classification is below a predefined threshold. They are sent to the user instead of the classification, requesting them to provide feedback for the construction drawing that the classification belonged to.

Classifications A classification is generated by the AI Classification Algorithm (Figure 5.4), and belongs to a specific construction drawing. A classification consists of a reference to the construction drawing and a simple boolean, stating that the construction drawing is either a supplier drawing or not.

Result Requests A user can send a result request, asking for the results of previous classifications that they have commissioned. They will either receive a message from the system that there are no results with their user ID in the database, or they will receive their results in the form of classifications or feedback requests.

Result A result is represented either by a feedback request or a classification.

User ID A user ID is a unique identifier of either a user using the GUI or an external software connected via the API.

5.2.1 System Context View

Figure 5.3 shows the system context view. Here, "System" is the software system that is to be developed. It allows the user to upload large numbers of construction drawings and get a classification for each drawing. "External Software" is a placeholder for different software systems that will be connected to the system in an automated process. Since it is an important architectural requirement, that the system provides one API that can be used to connect different systems (FR1), the diagram is limited to one placeholder external system. The Identity Provider provides a single sign-on approach to the system, which was mentioned during the focus group in the context of the requirement for the system to be secure (QA5).

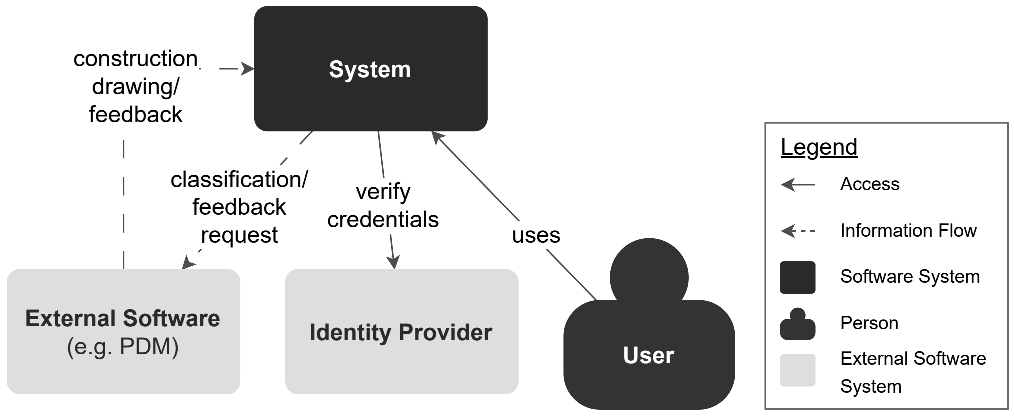


Figure 5.3: C4 system context view of the developed architecture

It is an architectural requirement that the system provides a role concept (QR7), e.g. by distinguishing between normal users and admins who have extended rights. However, as the system does not contain sufficient uses for extended rights to justify the increased complexity that would result from having different user groups, this feature has been omitted. More detailed descriptions of the external elements can be found below.

System The System accepts construction drawings from users or external software. It generates a classification for each construction drawing and sends it back to the user or external software.

User To increase security (QA5) of the system, users have to authenticate themselves before being able to upload any construction drawings. Each user has a unique user ID, which is internally used to identify the owner of a construction drawing or classification. Once authenticated, a user can upload a construction drawing. They then receive either a classification or a feedback request, prompting them to classify the construction drawing by themselves. Upon receiving a classification, the process would usually be finished. However, the user themselves can decide to give feedback, e.g. if the classification was wrong. It works the same as when the application prompts the user to give feedback. If a user logs out before a requested classification is finished, they can send a result request the next time they log in to get the result of their previous classification.

External Software External Software is similar to a normal user, with the one difference, that it communicates directly with the API instead of taking the detour over the GUI. As well as a user, External Software needs to be authenticated to increase security (QA5), which is done by attaching a valid token to each request. Even in an automated process, the external software should be able to handle a feedback request, as this can be the output of the system at any time, instead of a classification. Because External Software is usually connected to the system in an automated process, the classifications it orders are not stored in the database if the External Software is unreachable. In this case, the result is discarded to increase compliance (QA4).

Identity Provider The identity provider shields the system from access by unauthenticated users. Doing this, the identity provider receives authentication requests from the system, which it must process - usually by granting the user or external software access or returning a failure message.

5.2.2 Container View

In Figure 5.4 the container view of the system is depicted. Here, the system is divided into four main units: the GUI which was explicitly requested during the focus group (FR3); the Result Database; the API Application which is shown in more detail in Figure 5.5; and the AI Algorithm, which is itself divided into three parts: the Feedback and Classification Algorithms and the Vector Database. This division of the AI Algorithm follows a scheme of Sharma and Davuluri [SD19]. They suggest dividing an AI Algorithm into a controller and a model part, when using the MVC architectural pattern.

In their paper, Washizaki et al. [WUKG19] suggest, that the AI Algorithm should be isolated from business logic. That, together with the requirement for a modular design (FR2), is the reason for the AI Feedback Algorithm and AI Classification Algorithm being isolated from the API Application, where the business logic is handled. Between the API Application and the two AI Algorithms, Queues are located to decouple the containers from each other. This makes the system scalable

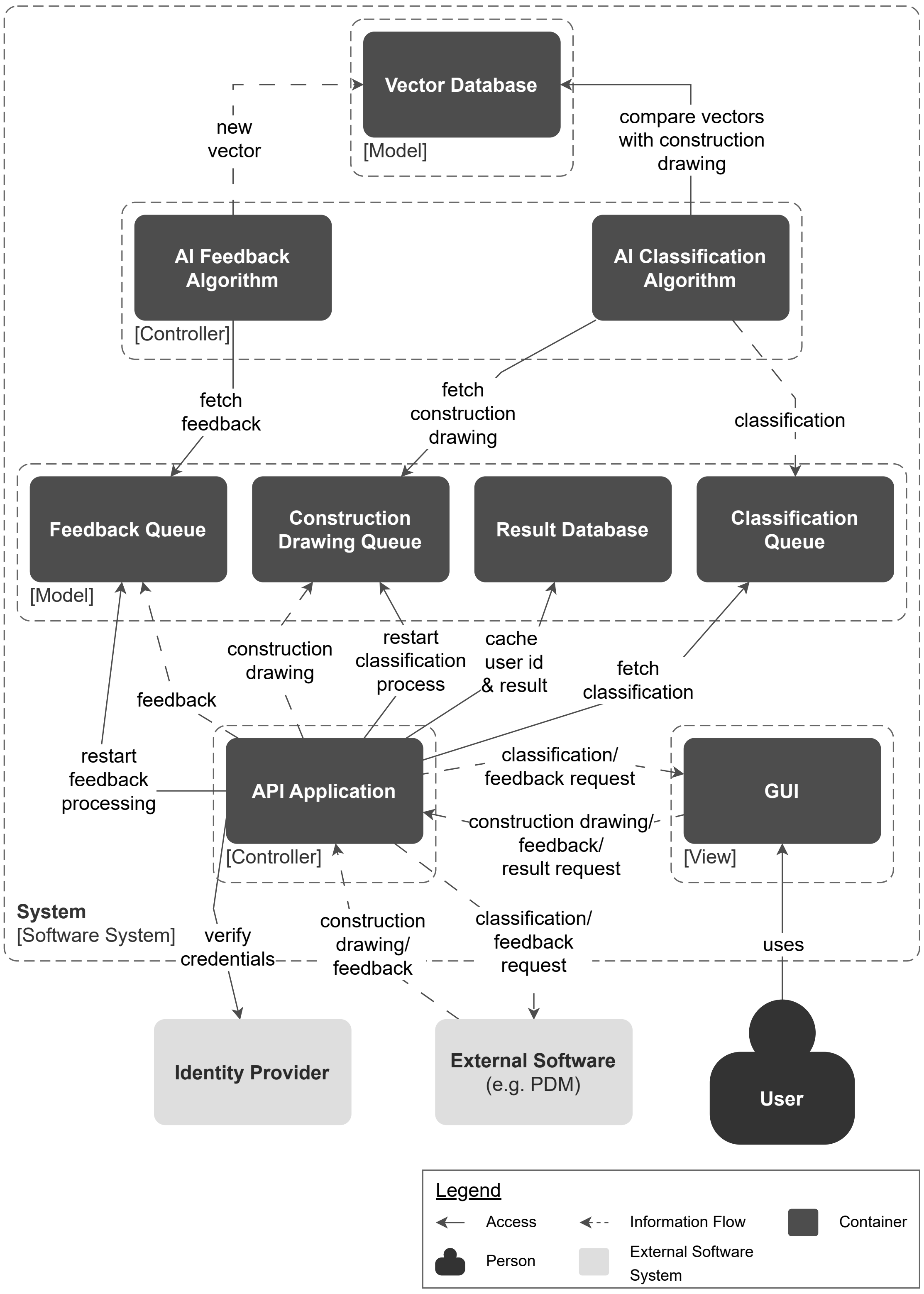


Figure 5.4: C4 container view of the developed architecture

(QA1) by preventing the overloading of one container by “pushing” data until an overflow occurs, because the queues balance workload peaks. Provided that the queues can handle parallel access, this allows the system to run multiple AI containers simultaneously (QR1), since each container can pick jobs from the queue and insert results into the other queue individually. As reliability (QA3) is an important architectural requirement, it is necessary that the queues are their own container so that if either the API Application or the AI Algorithm container crashes, the other can still access the queues. The location of the queues was explicitly requested by the architecture expert during his expert interview.

The system was designed using the MVC pattern, as can be seen by the three groups “Model”, “View” and “Controller” in the diagram. This architectural pattern was chosen because of two reasons: First, it was specifically requested by one of the OEM-employees during the focus group, and thus is one of the architecture requirements (FR6), and secondly, as elaborated in Section 2.3, it provides a good option to build a web-based application that can be centrally hosted in the cloud, which is another requirement for the system (FR5). The modified version of the MVC pattern introduced in Section 2.3.2 was used, because with this the data from the model passes through the controller before being presented to the user. This data flow is advantageous to the normal MVC pattern in this case, as the controllers must determine whether to present the classification to the user or generate a feedback request (QR4), instead of simply displaying the model’s classification to the user.

The Result Database was requested by the architecture expert and the product owner during their expert interviews to enable asynchronous results (FR8). The product owner emphasized that it depends on the cost and complexity of running such a database whether users will be forced to stay logged in until they get their result or the database approach will be implemented. However, since the asynchronous approach provides a much better user experience, the existence of the database is preferred. Detailed explanations of each container of the architecture proposal can be found below.

API Application The API Application manages the business logic of the Software. As the API is part of it, External Software directly communicates with the API Application. It distributes construction drawings, classifications, feedback requests and feedback to the intended receivers, evaluates confidence levels of classifications, and creates feedback requests if said confidence level is too low.

GUI For an unauthenticated user, the GUI consists only of the login screen of the identity provider. Authenticated users can upload construction drawings and get the classification displayed afterward. Furthermore, there is the possibility to give feedback to the construction drawings uploaded by them, which can be done by manually classifying the construction drawing. There is also a mechanism to request results of previously uploaded construction drawings, which is necessary to enable the concept of asynchronous results (FR8).

AI Algorithm The AI Algorithm works with a vector comparison approach. For this purpose, header-tables of construction drawings are stored in a database in the form of a vector. To classify a construction drawing, the algorithm derives a vector from the header-table of the construction drawing and compares it with each vector of the database using similarity search. The classification and its confidence level are derived from the vector most similar to the one extracted from the construction drawing. The AI Algorithm is divided into three main parts: the Vector Database, which stores all known vectors and thus is part of the model; the Classification Algorithm, which classifies construction drawings by performing the comparison of vectors; and the Feedback Algorithm, which enables feedback-based learning (FR7) on the AI Algorithm side by being able to process feedback which will then be taken into account for subsequent classifications. The processing is done by converting the construction drawing header-table into a vector and inserting it together with the classification into the Vector Database. Since both the classification of construction drawings and the processing of feedback are based on header-table-detection and vector computation, they are very resource intensive, and only a limited number of tasks can run in parallel.

Feedback Queue The feedback queue receives feedback from the API application. Therefore, it contains all feedback that needs to be processed by the AI feedback algorithm. Whenever the feedback algorithm starts processing feedback in the queue, it marks the entry as "in process". The feedback is not removed from the queue until processing is complete. The "in process" label is removed after a certain amount of time to ensure that it will be processed even if the AI container crashed during the process and thus make the system failsafe (QR3).

Construction Drawing Queue The construction drawing queue is similar to the feedback queue, except that it contains construction drawings instead of feedback.

Classification Queue The classification queue is filled by the AI Classification Algorithm. Each entry contains a classification, the confidence level, and a reference to the construction drawing to which the classification belongs. The API Application processes entries in this queue in the same way as entries in the feedback and construction drawing queues.

Result Database The result database is used to cache user IDs associated with construction drawings that are currently queued or being processed. Once a drawing is classified, the user ID is used to deliver the classification to the correct user. Simultaneously, to enable asynchronous results (FR8), the result is also written to the database to be requested by the user the next time they log in to the system. As compliance (QA4) and the deletion of construction drawings (QR6) are important requirements, entries in the database expire after a certain period of time. Otherwise critical data might remain in the system indefinitely, if the user never requests their previous classification results.

5.2.3 Component View

A detailed depiction of the API Application can be found in Figure 5.5. The inner dependencies were designed to have a controller for each major data type, although feedback is not pre-processed in any way and therefore does not need its own controller.

Since the AI Algorithm container may crash while performing a classification or processing feedback, there may be construction drawings or feedback in the corresponding queue that are labeled "in process" but are not being processed anymore. To solve this problem and make the system failsafe (QR3), the Restart Controller is introduced. It forces the processing of construction drawings and feedback to be restarted when a timeout is reached. This leads to another problem that must be handled by the Classification Controller: There is a chance that a classification will be completed after it has been restarted. Then there will be two results, which is acceptable if these results are equal. If they are not equal, to ensure reliability of the result (QA3), the Classification

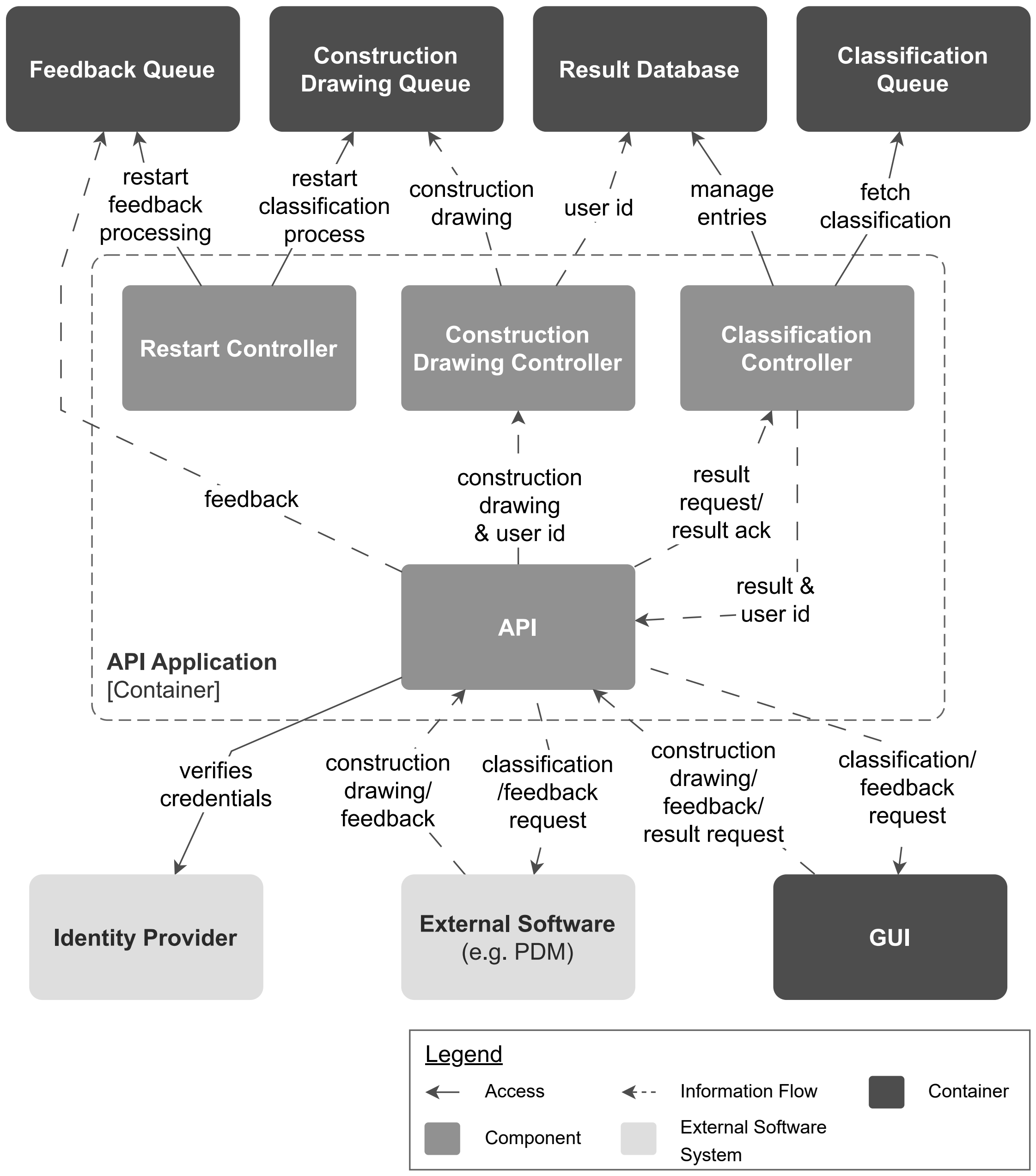

Feedback Queue
Construction Drawing Queue
Result Database
Classification Queue
restart feedback processing
restart classification process
construction drawing
user id
manage entries
fetch classification
Restart Controller
Construction Drawing Controller
Classification Controller
feedback
construction drawing & user id
result request/ result ack
result & user id
API
API Application
[Container]
verifies credentials
construction drawing/ feedback
classification /feedback request
construction drawing/ feedback/ result request
classification/ feedback request
Identity Provider
External Software (e.g. PDM)
GUI
Legend
Access
Information Flow
Container
Component
External Software System

Figure 5.5: C4 component view of the developed architecture

Controller must generate a feedback request, even if both classifications have a sufficiently high confidence level, but in two different directions. A detailed explanation of each component can be found below.

API On one hand, the API is responsible to forward construction drawings and result requests received from a user or external software to the respective controller. When forwarding a construction drawing, it also sends the user ID of the user who uploaded it. Feedback is directly appended to the Feedback Queue, since this doesn't need to be further processed beforehand. On the other hand, it receives classifications or feedback requests from the Construction Drawing Controller, together with an associated user ID, and is responsible to forward the output to the correct user. After successfully delivering the output, the API sends a request acknowledgment to the Result Controller. Before being able to upload any construction drawings, users need to authenticate themselves, which is also handled by the API, by redirecting any authentication requests to the external Identity Provider.

Construction Drawing Controller The Construction Drawing Controller receives construction drawings and associated user IDs from the API. The construction drawings are added to the Construction Drawing Queue, while the user IDs are cached in the Result Database. If a multi-page construction drawing is received, it is split up into different documents, each containing exactly one construction drawing. These are then processed further like normal construction drawings.

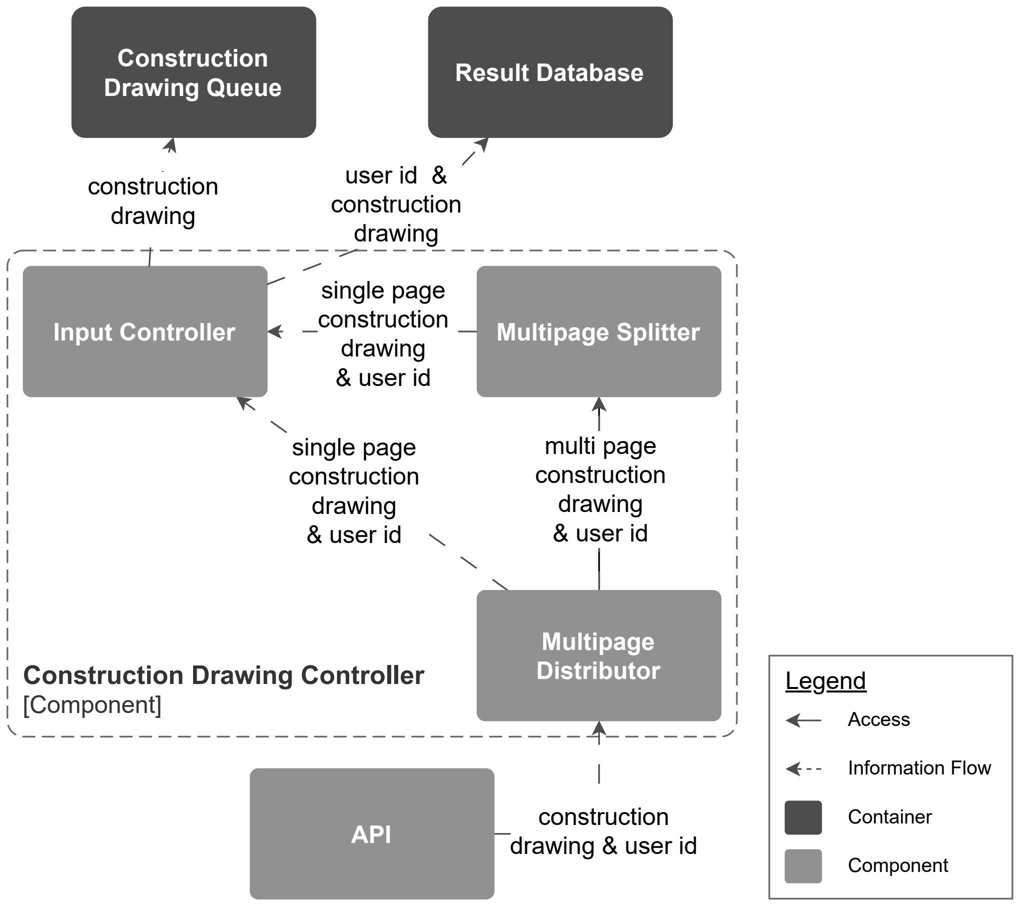
Restart Controller The Job Restart Controller periodically checks how long each entry in the Construction Drawing and Feedback Queues has been marked "in process" and removes the label if it exceeds a specific time limit.

Classification Controller The Classification Controller processes classifications it fetches from the Classification Queue. It determines whether the confidence level is above a specific threshold and if not, creates a feedback request. In this case, the classification is discarded to increase compliance (QA4). Then it fetches the user ID associated with the construction drawing the classification belongs to and sends both to the API. Meanwhile, it also checks if the user ID belongs to a user using the GUI, and if so, adds the result to the entry in the database to enable asynchronous results (FR8) by fetching it from the database at a later time. Otherwise, the entry is removed. When the Classification Controller receives a result request from the API, it fetches all result-entries associated to the user from the result database and returns those to the API. When a result acknowledgment is received from the API, the corresponding entry of the database is removed.

5.2.4 Component View of the Construction Drawing Controller

The Construction Drawing Controller, shown in detail in Figure 5.6, consists of three further components. The Input Controller manages the input given to the AI Classification Algorithm via the Construction Drawing Queue and other data related to the input, which is inserted into the Result Database. The Multipage Distributor and Multipage Splitter make sure that the Input Controller only receives single-page construction drawings, by splitting up multi-page drawings. Multipage Distributor and Splitter are separate components to fulfill the concept of single responsibility (QR9), one of the architectural requirements. This separation was explicitly requested by the architecture expert during his interview.

To increase compliance (QA4), multi-page construction drawings are deleted after they have been split up. Single results of former multi-page drawings can be delivered to the user individually, since the classifications of the different pages do not relate in any way. Therefore, to reduce complexity, the single-page construction drawings resulting from the split up, contain no reference to the former multi-page drawing or the other related single-page drawings. Detailed explanations of the three components can be found below.



Construction Drawing Queue
Result Database
construction drawing
user id & construction drawing
Input Controller
single page construction drawing & user id
Multipage Splitter
single page construction drawing & user id
multi page construction drawing & user id
Multipage Distributor
Construction Drawing Controller
[Component]
API
construction drawing & user id
Legend
Access
Information Flow
Container
Component

Figure 5.6: C4 component view of the Construction Drawing Controller

Multipage Distributor The Multipage Distributor has the single responsibility of checking the incoming construction drawings to determine if they are multi-page drawings. It forwards multi-page drawings to the Multipage Splitter and single-page drawings directly to the Input Controller.

Multipage Splitter The Multipage Splitter splits up multi-page construction drawings into different documents, each containing exactly one single-page construction drawing. These are then sent to the Input Controller together with the user ID former belonging to the multi-page drawing.

Input Controller The Input Controller controls the input given to the AI Classification Algorithm via the Construction Drawing Queue. It appends incoming construction drawings to the Construction Drawing Queue, while inserting the associated user IDs into the Result Database, each together with a reference to the construction drawing to which they belong. The AI Classification Algorithm does not require user IDs, and minimizing the locations where user data is stored, even if it is only cryptic IDs, increases the systems compliance (QA4).

5.2.5 Component View of the Classification Controller

The Classification Controller, depicted in Figure 5.7, evaluates classifications and delivers the result to the user. The Classification Interpreter primarily executes the task of evaluation. In order to allow for asynchronous results (FR8), the user can request a previously initiated classification's result. The Request Handler manages these requests.

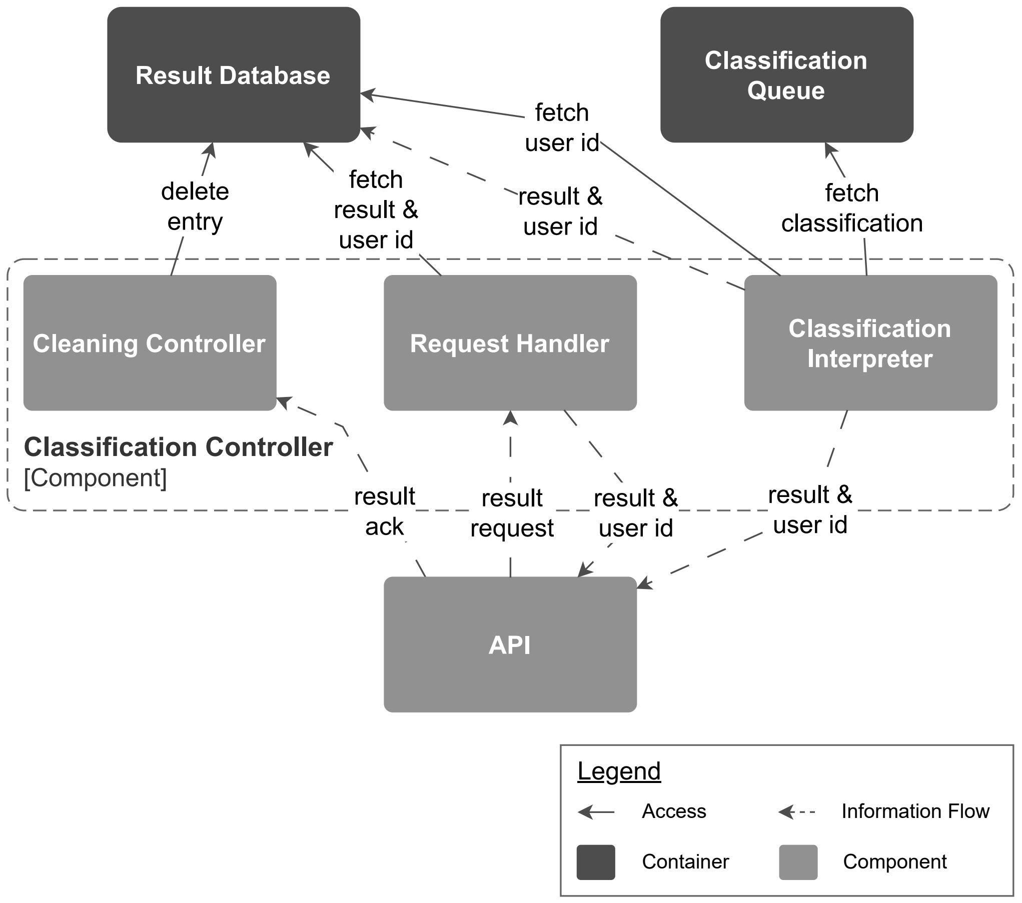


Figure 5.7: C4 component view of the Classification Controller

It is an architectural requirement, that construction drawings must be deleted after processing (QR6), and it is reasonable to delete any other data related to users or construction drawings when it is no longer required. To enable asynchronous results (FR8), it is necessary to store certain data temporarily in the database, but it must be deleted eventually. The Cleaning Controller addresses this issue by ensuring that no sensitive data remains in the system after the user has received their result. The Classification Controller was created as a result of the product owner's and architecture expert's requirement for a Result Database during their interviews. Previously, the Construction Drawing Controller was responsible for evaluating classification confidence levels. Detailed descriptions of the components of the Classification Controller can be found below.

Cleaning Controller The Cleaning Controller receives result acknowledgements from the API. It then removes the corresponding entry from the database. It also periodically deletes any entries from the database that have exceeded their expiration date.

Request Handler The Request Handler receives result requests from the API, checks if there are any entries in the database that match the user ID in the result request and returns these entries to the API. If there are no matching entries, it returns an error message instead.

Classification Interpreter The Classification Interpreter fetches classifications from the Classification Queue. Fulfilling the architectural requirement of a threshold for the confidence level (QR4), it determines whether the confidence level is above a specific threshold and if not, creates a feedback request. In this case, the classification is discarded. Otherwise the classification (without its confidence level) is treated as result. Regardless of the result, the Classification Interpreter fetches the user ID associated with the construction drawing the result belongs to and sends both to the API. Meanwhile, it checks if the user ID belongs to a user using the GUI, and if so, adds the result to the entry in the database, where it can later be fetched by the Request Handler to provide an asynchronous result (FR8). Otherwise, the entry is removed immediately from the database since its remaining there serves no further purpose. This additionally complies with the requirement to delete construction drawings after they have been processed (QR6), and helps to achieve cost (QA2) and performance efficiency (QA6) by keeping the database as small as possible.

5.3 Expert Analysis

This section reports the results of the expert interviews in the following manner. Section 5.3.1 highlights issues identified by the experts during their interviews, that still persist in the final version of the architecture proposal, presented earlier in this chapter. This section also discusses solution approaches proposed by the experts to eliminate the issues. Section 5.3.2 reports the evaluation of the quality attributes as performed by the experts during the SAAM phase of the expert interviews.

5.3.1 Issues and Solution Approaches

During his interview, the AI expert noted that the requirement for a backup mechanism (QR2) in case the AI algorithm is corrupted, is not completely fulfilled by the architecture proposal. However, due to the modular design and the decoupling of the AI algorithm from the business logic, he further argued, that the AI algorithm could be replaced manually with relatively little effort. In the worst case, the replacement could be a mechanism that automatically adds a classification with the lowest possible confidence level to the classification queue, for each construction drawing that is added to the construction drawing queue. This way feedback requests for all classifications would be generated, forcing all users to perform the classification themselves.

The architecture proposal as is does not support encryption of construction drawings (QR5), as identified by the product owner during his expert interview. The architecture expert suggested implementing End-To-End Encryption (E2EE) that would solve this problem. To do so, the architecture expert considered an asymmetric encryption algorithm that would provide users and external software with public keys to encrypt the construction drawings before uploading them, so that the AI classification algorithm could decrypt them with its private key. However, since the decryption would have to be done directly before the classification, this approach would make the AI classification algorithms even more heavy-weight. Another problem would arise with the Multipage Controller and Splitter, since they also need to work with unencrypted construction drawings. No effective solutions to these problems were found during the expert interviews.

Finally, the architecture proposal does not feature an anti-corruption mechanism (QR8), which was noted by the AI expert. As user feedback is vital for the evolution of the AI algorithm, the ability of users to provide feedback in the form of manual classifications cannot be removed. However, identifying intentionally misleading feedback from users can be a difficult task. A potential solution suggested by the AI expert could involve having another user, unknown to the one who gave feedback, verify the manual classification. In the context of the case study, however, this solution is not possible because construction drawings are highly sensitive data and must not be presented to anyone other than the user who uploaded them.

5.3.2 Quality Fulfillment

During the last phase of the expert interviews, SAAM was used to evaluate the fulfillment of the quality attributes in the architecture proposal. Due to the limited time available during an expert interview, not all quality attributes could be assessed. The interviews focused on the three most important ones, according to the focus group participants' prioritization (see Figure 5.2). These were maintainability (QA7), compliance (QA4), and scalability (QA1). The scenarios used can be found in Appendix E.

The evaluation of the maintainability of the architecture proposal was generally positive. The AI and architecture experts agreed that maintainability was good based on both the standard scenario and their individual scenarios. The modular design of the architecture makes it easy to add another component or use case, or move the system incrementally to a different cloud infrastructure. However, the architecture expert pointed out that maintainability is highly dependent on the implementation of a system and can only be marginally evaluated based on the architecture alone. The product owner was even more convinced of this, saying that the maintainability of the final software cannot be evaluated from the architecture alone.

The experts opinions differed on whether compliance was achieved. Based on the default scenario used by all three experts, the AI expert and the product owner agreed that compliance is given. They based their statement on the fact that the identity provider protects the system from unauthorized access, and that no interface is provided that would allow even authorized users to request compliance-relevant data. The architecture expert, however, stated that the scenario contained too many runtime parts to be evaluated based on architecture alone. Based on their own scenarios, the product owner and architecture expert rated compliance as met due to the absence of the aforementioned interface and the fact that no personal data other than user IDs is persisted by the system. The AI expert rated it as "must be better", considering that the missing anti-corruption mechanism would provide the possibility to corrupt the classification in such a way that false negatives would be possible, i.e., to classify a supplier construction drawing as an internal one with a high enough confidence level that the classification would not be questioned. He stated that this would be an indirect violation of compliance.

With regard to scalability, the architecture expert and the product owner pointed out that this would again depend heavily on the final implementation as well as the resources of the runtime environment. However, based on the architecture alone, all three experts with both the standard and their own scenarios agreed that scalability is well met. The reasons for this rating were largely the same for all three experts, including the ability to run the system with multiple AI classification containers, and the mitigation of load peaks of either the AI algorithm or the API application by the queue structure.

6 Discussion

This chapter discusses the results presented in Chapter 5 as well as the study design that produced them. Section 6.1 and Section 6.2 focus on reflecting on how the results can be interpreted as an answer to the research questions, as posed in Section 4.1. Since the architectural requirements catalog is intended to answer the first research question, its main limitation may be its lack of completeness. Section 6.1 addresses this topic. The second research question is to be answered by the architecture proposal, so the main problem here is whether the proposed architecture fulfills all the requirements. This topic is discussed in Section 6.2. Limitations to the external validity of the contributions of this thesis, as well as common pitfalls of the methods used in the study design and the countermeasures that were taken to avoid them are discussed in Section 6.3.

6.1 Architectural Requirements

This section discusses the completeness of the catalog of architectural requirements, by comparing it to catalogs of related papers, introduced in Chapter 3. The completeness of the catalog presented in Section 5.1 may be limited, due to missing quality attributes, such as usability, operability, and extensibility. They may, however, be indirectly included through the definition of other quality attributes and functional requirements. Furthermore, the basic functionality of the system is only marginally covered by the architectural requirements catalog.

None of the architectural requirements specifically define the basic functionality of the system, i.e., no requirement directly states that the system should accept construction drawings, classify them into one of two predefined classes, and output this classification to the client. At the same time, many requirements assume that the basic functionality of the system is given. This suggests that the participants of the focus group quietly assumed that the basic functionality will be addressed either way. This behavior was also observed by Kano et al. [KSTT84]. The quietly assumed requirements correspond to the must-be quality requirements of their Kano model.

As proposed by Kazman et al. [KAK01], their list of quality attributes can be applied to most cloud-based applications, and should therefore be comparable to the catalog of architectural requirements proposed by this thesis. However, the authors' list includes a few quality attributes that are not featured in this thesis' catalog. One of them is usability, which may have been quietly assumed by the participants of the focus group. In a commercial context, like the product development process with construction drawings, it is logical that quality attributes such as compliance, maintainability, and cost-efficiency are generally more important than usability. As with the basic functionality, this may have led participants to assume that if all other quality attributes were sufficiently satisfied, usability should also be maximized. Since certain aspects of usability are implied by other requirements, such as asynchronous results (FR8), it has at least not been completely neglected. Further quality attributes existing in the list of Kazman et al. [KAK01] are operability and extensibility, which are in the catalog of this thesis not mentioned specifically as well. However, since the definition of maintainability (QA7) includes the modifiability of the system, extensibility is implied. The corresponding International Organization for Standardization (ISO) standard 25010:2011 [ISO11] defines operability as the system being easy to operate and control, which includes conformity with user expectations. This behavior is at least marginally included by the definition of reliability (QA3). Most of what remains of operability then corresponds to the ISO definition of usability [ISO11], which is discussed above. The same differences exist with the catalog of quality attributes in the later paper of Kazman et al. [KIC05].

Rimal et al. [RJKG11], introduced in Chapter 3, propose three sets of requirements for provider, enterprise and user. Especially the set of requirements for enterprises contains many of the requirements presented as a result of this paper. Examples include security, scalability and cloudonomics, which is largely represented by this thesis' requirement for compatibility with different cloud infrastructures (FR5). The provider and enterprise requirement for interoperability is at least partially present in the catalog presented in this thesis in the form of API-capability (FR1). However, since this thesis' study works with a much more detailed description of the system, there are also many requirements in both catalogs that are not included in the other. One of these is user experience, featured in the user requirements catalog of Rimal et al. [RJKG11], which is discussed under the name of usability in the paragraph above. In addition, the enterprise requirements catalog includes third party engagement in hosting the application in the cloud, which was not mentioned in the focus group or expert interviews of this case study, possibly because they were focused on architectural requirements for the system rather than operational issues. Most of the functional requirements of the catalog presented in Table 5.1, are not present in all three catalogs of Rimal et al. [RJKG11], which may be due to the more abstract description of their system.

In summary, while the major parts of other architectural requirements catalogs are consistent with the one presented in this thesis, there are always a few requirements that are not directly included in the latter. Perhaps due to the very specific context of this case study, many requirements in other catalogs are presented in this thesis' catalog in the form of functional requirements that are completely missing in most of the more abstract catalogs.

6.2 Architecture Proposal

This sections addresses the discussion of the second finding of this thesis, the architecture proposal. Section 6.2.1 evaluates the extent to which the proposed architecture fulfills the architectural requirements depicted in Figure 5.1. Meanwhile Section 6.2.2 compares the architecture to various architecture designs of related papers, presented in Chapter 3.

6.2.1 Fulfillment of Architectural Requirements

The development of the architecture proposal was mainly based on the architecture requirements collected beforehand. However, as many different requirements influenced the architecture, it is possible, that the architecture proposal does not fulfill each requirement to a satisfactory degree. To make this explicit, this section discusses the fulfillment of all architectural requirements presented in Section 5.1 by the architecture proposal, presented in Section 5.2.

During the development of the architecture proposal, almost all architectural requirements were considered. However, standalone (FR4) opposes cloud-native (FR5). In this particular case, the architecture proposal only considered the higher priority requirement as determined by the focus group's dot voting, which is cloud-native with eight in favor and three against. This decision is further supported by the fact that the AI algorithm is very resource-intensive, and few users have computers that can provide enough resources.

Cloud-native (FR5) and three layer architecture (FR6) are the main requirements that influenced the basic shape of the architecture proposal, and are therefore satisfactorily fulfilled. Compatibility with different cloud infrastructures (FR5) is highly dependent on the final implementation; the architecture itself allows the system to be containerized, making it compatible with standard cloud infrastructures, such as Kubernetes[1] or Docker Swarm[2].

Functional requirements can be translated directly into architectural properties, so it is not necessary to discuss the fulfillment of most of them further in this section. The architecture proposal includes an API (FR1), a GUI (FR3), a mechanism that allows the AI algorithm to implement feedback-based learning (FR7), and a results database that allows users to receive their results asynchronously (FR8). A modular design (FR2) is at least partially present, as the AI algorithm is completely decoupled from the business logic. As Mcheick and Qi [MQ11] suggest, the MVC architectural pattern supports modular design as well. Therefore, all functional requirements, proposed in Table 5.1, are satisfied by the proposed architecture proposal, except the standalone-related requirement of compatibility to different OS (FR4).

Multiple classification containers (QR1) are supported by the queue structure, since multiple containers can all work with one queue. The queue structure also allows for failsafety (QR3), since even if all AI containers and the API application crash, the data in the queues will not be lost. This means that only data that has just been uploaded to the system and has not yet reached the queues will be lost if the API application crashes. Since integrating a failsafety mechanism for this relatively small amount of data would make the system more complex and thereby less maintainable, this weakness has been accepted. The Classification Controller's Classification Interpreter satisfies the requirement for a threshold for the confidence level (QR4). However, this feature could be implemented in a more user-friendly way by allowing users to adjust the threshold via the GUI. The Cleaning Controller ensures that construction drawings are deleted after processing, thus satisfying the requirement for deletion of construction drawings (QR6). The single responsibility principle (QR9) is also largely respected. The controllers of the API application each have more than one responsibility, but this is justified by the MVC pattern.

The fulfillment of the other requirements presented in Table 5.3 needs to be discussed more thoroughly. The proposed architecture does not incorporate a role concept (QR7), since there is no necessity for different roles in the suggested use cases. The main priority was to ensure simplicity and thereby high maintainability (QA7). As discussed in Section 5.3.1, the requirements for a backup mechanism (QR2), an encryption mechanism (QR5) and an anti-corruption mechanism (QR8) are not fully addressed by the architecture proposal. Since a dedicated backup mechanism would have made the system more complex, and in turn offered little advantage over the solution proposed by the AI expert, as presented in Section 5.3.1, simplicity was preferred in this case

[1]https://kubernetes.io/ (followed on October 26, 2023)
[2]https://docs.docker.com/engine/swarm/ (followed on October 26, 2023)

as well. The approach of implementing E2EE discussed in Section 5.3.1 is expected to satisfy the requirement for encrypted construction drawings, but due to the limited time after the expert interviews, it was not possible to make an informed decision on which encryption algorithm to use in the scope of this thesis. The anti-corruption mechanism could not be realized because the AI expert's solution was not ideal, as discussed in Section 5.3.1. A better solution could not be found due to time limitations as well.

Measuring the fulfillment of quality attributes can be difficult [KBAW94], especially when there is only an architecture proposal as a basis. Of course, the architecture proposal was developed with the quality attributes in mind. As reported in Section 5.3.2, during the expert interviews, the fulfillment of the three most important quality attributes was evaluated. The fulfillment of the remaining quality attributes is discussed as far as possible in the following paragraph.

Reliability (QA3) in terms of a certain output of a result is ensured by the asynchronous results mechanism, as well as by the failsafety of the queues. However, the weakness of queue failsafety discussed earlier in this section reduces reliability to some extent. Reliability (QA3) in terms of accuracy of the results depends mainly on the quality of the classification of the AI algorithm, but is supported by the threshold check of the confidence level. The system might provide higher cost-efficiency (QA2) if it was standalone, but as discussed earlier in this section, developing a standalone version of the system is no option. The further fulfillment of cost-effieciency as well as of the other quality attributes presented in Table 5.2 is strongly dependent of the implementation than the architecture, and the discussion of their fulfillment is therefore omitted.

6.2.2 Comparison with other Architectures

This section compares the proposed architecture of this thesis to architecture designs of related papers presented in Chapter 3.

Ding and Ben Salem's [DB18] proposed architecture features a distribution of the classification algorithms. Therefore, document classification can be performed close to the user's endpoint, while only model training and updates are performed centrally. This dataflow is quite different from the architecture proposal of this thesis, because there the classifications are created centrally in one place. The authors argue that their distributed design minimizes the risk of leaking sensitive data by processing data close to where it is uploaded to the system, since security was one of their key requirements, too. This statement may hold true for their specific scenario, but in the context of this study, cost-efficiency (QA2) is also a crucial requirement which would be compromised by the need to host multiple edge computing servers. Furthermore, since in an industrial context, most users of the system access it from the same place (the company's location) there is usually no need for multiple physical classification points if the single server is hosted close to the company's location.

The architecture design proposed by Appiani et al. [ACC+01] implements a client/server/database approach as it is called by the authors. Even though the objectives of their system differs, this approach is very similar to the architecture this thesis proposes, as the modified MVC pattern can be seen as a client/server/database separation as well. Furthermore the design of this thesis implements a modular approach and aims to be highly scalable and reliable, which are the main attributes of the server part of the authors architecture as well.

The architecture pattern used by Lai et al. [LTWL10] in their architecture proposal for an AI-powered social network closely resembles the pattern employed in the architecture proposal of this thesis. While the authors do not explicitly name the pattern they used, it shows great similarities with the MVC architectural pattern used by the architecture proposed by this thesis as well. This further reinforces the decision of using the MVC pattern.

Jung and Joe [JJ23] propose an architecture that resembles similarity to this thesis' architecture proposal in terms of modular design and high scalability. The authors' architecture differs from the one presented in this thesis in the architecture pattern used. Their architecture features four layers on the application sever, which contrasts with the MVC architectural pattern. This shows, that the MVC architectural pattern, while being a popular choice, it is not the only suitable architecture pattern for AI-related cloud computing systems.

The architecture proposed by Hendradi et al. [HKM19] differs notably from the one presented in this thesis. Important aspects of their architecture are different user roles and mobile support for the GUI, which are both not featured by the architecture proposal of this thesis. Both aspects may be attributed to the system's intended purpose.

6.3 Limitations and Threats to Validity

This section addresses the limitations of the contributions of this thesis along with potential threats to their validity. Since case studies are known to have extremely limited external validity [RH09], this is discussed in the dedicated Section 6.3.1. Section 6.3.2 presents and discusses the remaining threats to validity.

6.3.1 External Validity

The results, being the catalog of architectural requirements, together with its prioritization and division into mandatory and non-mandatory requirements, and the architecture proposal, can be applied to cases beyond the scope of this study. This includes, among other things, applications with the goal of classifying documents other than construction drawings, which may include the application being used in a context other than the product development process, and the use of an classification algorithm approach other than vector-based similarity search.

Due to the lack of requirements for the basic functionality of the system, as discussed in Section 6.1, the architectural requirements catalog, presented in Figure 5.1, is only slightly specific to construction drawings. It therefore can be applied with minimal effort to applications that aim to classify documents other than construction drawings and will be used in different contexts than the product development process. However, when being applied to specific usecases, it has be kept in mind, that the basic functionality of the particular case has to be added to the catalog. Most of the quality attributes can even be applied to cloud-based applications in general, which was found by Kazman et al. [KAK01], among others, as described in Chapter 3.

The majority of the requirements refining the quality attributes are applicable to different document types as well, since most of them reflect only functional approaches to their respective quality attribute. However, especially the requirements for encryption (QR5) and immediate deletion of construction drawings (QR6) (or, generally, the encryption and deletion of documents) may be less important or even unnecessary for document types that are less critical than construction drawings.

Most of the functional requirements can be applied without modification. Different interfaces, especially the requirement for an extended API (FR1) that provides an interface for the application to connect to an automated process, may vary depending on the case. A GUI (FR3) is probably useful in any case, because even if the main purpose of the application is to be used by other software systems, there may always be a few users who want to use it manually. The requirement for the application to be standalone (FR4) will not be feasible in most cases, for the same reasons as in this case. However, if the AI algorithm is replaced by a more resource-efficient approach, or if all users are equipped with sufficiently powerful computers, the requirement for a standalone version may be applicable. Since the primary goal of a document classification application is to provide its users with classifications that are as accurate as possible, some form of feedback-based learning (FR7) will prove useful in most cases. Asynchronous results (FR8) depend on the use case of the application, since in this case only users should be provided with this feature. If users use the application regularly, this requirement probably applies, but if a user using the system is merely an exception, it is probably not worth the increased maintenance effort.

Because the architectural requirements catalog is applicable to different document types, the prioritization and division into mandatory and non-mandatory requirements may also be applicable sometimes with minor modifications. Most of the high-ranking and mandatory requirements are quality attributes that are important for most types of cloud-based applications, as discussed above. However, compliance (QA4) and security (QA5) may rank lower for document types that are less critical than construction drawings. Although compliance with respect to user data is always important. The relevance of API capability (FR1) may also vary depending on whether or not the application needs to be connected to other software systems.

The architecture proposal is the result of translating the catalog of architectural requirements into the more formal form of the C4 model, with a number of additional adjustments. Therefore, if the architectural requirements are applicable to applications with the goal of classifying document types other than construction drawings or applications being used in contexts other than the product development process, the parts of the architecture proposal that are directly related to the architectural requirements are also applicable. The other parts, i.e., those derived from the case description and the basic functionality of the architecture proposal, are discussed below.

The architecture proposal as is only partially relies on specific properties of construction drawings. Most of the components, with the exception of the AI algorithm and the Multipage Controller and Splitter, simply forward construction drawings without further processing. These parts of the architecture could be applied to other document types without modification. Depending on the document type, and if there are documents similar to multi-page construction drawings, the Multipage Controller and Splitter can either be removed, with the API communicating directly with the Input Controller, or used to split the other multi-page documents. Of course, if the architecture proposal is to be used for other document types, the construction drawing data type should be renamed, as well as all elements with construction drawing in their names. Since the AI algorithm is only a marginal part of the architecture, it could be replaced by an algorithm operating on another document type.

Such a replacement of the AI algorithm is enabled by the architecture. The architecture considers the AI algorithm as a black box as much as possible, and although the Vector Database is part of the container view for comprehensibility reasons, it is not necessary for the functionality of the architecture. Therefore, the proposed architecture can be used with any document classification algorithm, AI-based or not, that meets the following three prerequisites. The algorithm must be able to adhere to the queue structure, i.e., it must be able to take jobs from the Construction Drawing Queue (or, generally, the Document Queue), mark the jobs as in progress, add the resulting classification to the Classification Queue, and remove the job from the Construction Drawing Queue. In addition, it must be able to provide a confidence level for each classification it generates, and finally, it must be able to process user feedback, taking it from the Feedback Queue in the same way that it takes documents from the Construction Drawing Queue.

As indicated by the requirement for feedback-based learning (FR7), the catalog of architectural requirements also assumes that the algorithm performing the classification is able to process user feedback and thereby improve its classification accuracy. The requirement for an anti-corruption mechanism (QR8) also assumes that users can influence the algorithm in some way. All other architectural requirements of Figure 5.1 are directly applicable to a system using any type of algorithm for classification, because they only address the business logic part of the system. The quality attributes are affected by the chosen algorithm, but if the implementation of the algorithm is done with the quality attributes in mind, they do not exclude specific types of algorithms.

In summary, both the catalog of architectural requirements and the architecture proposal are applicable to use cases other than the one for which they were developed. The further the actual use case is from the proposed use case, the more limitations apply to both results and the more changes need to be made for them to be applicable. The prioritization of requirements and the division into mandatory and non-mandatory requirements are more limited to this specific use case, as the importance of architectural requirements varies greatly even between very similar use cases.

6.3.2 Other Threats to Validity

This section addresses potential threats to the validity of the contributions of this study that may have arisen from the study design. It also discusses countermeasures taken to mitigate the risks.

The design of the focus group, i.e., the introduction presenting the basic functionality of the software, may have led the participants to quietly assume the basic functionality as given, while generating further architectural requirements. This probably did happen, as Section 6.1 suggests. However, the key question of the brainwriting phase of the focus group, during which the architectural requirements were gathered, suggests otherwise, as can be seen in Appendix A. The introduction for the focus group was necessary to provide a common starting point for all participants, since not all of them were part of the project.

After the focus group, a participant gave feedback, that some of the statements the moderator made may have added bias to some of the participants' ideas. This may have prevented the group from coming up with different ideas or limited the thought process in certain directions. However, because the brainwriting phase was intentionally placed at the beginning of the focus group, the fundamental idea generation was complete before anyone could add any bias during the presentation of the ideas.

According to VanGundy [Van84] it may reduce the creativeness of participants, if a person of higher status is present. They “may be reluctant to verbalize ‘wild’ ideas” and probably also examine the ideas of the person of higher status less critically. The basis for this problem was given in the focus group, because one of the three employees of the OEM is the superior of the other two. However, the impact of this happening was considered acceptable. All three participants affected by this difference in status assumed the same role, while the other roles were covered by independent participants who were unaffected by the OEM’s hierarchy and could therefore examine the boss’ ideas critically and verbalize also their own unconventional ideas.

The expert interview design might yield the problem, that the material the participants received beforehand was written in English. This could be seen as a threat to validity, since all interviewed experts are German native speakers. This problem has been accepted in order to avoid the problem of making mistakes during translation. The problem is mitigated by the fact that all three experts that were interviewed speak English fluently and are accustomed to research in English, especially regarding software-engineering-related topics. Furthermore, prior to asking for their opinions on the architecture proposal, the experts were asked if they had any questions or problems with understanding the architecture and/or its description at the beginning of each interview.

Nasar [Nas23] as well as Deakin and Wakefield [DW14] reported that once the recorder was placed on the table and the audio recording started, the participants spoke less freely. Since the audio recording is an indispensable foundation for data analysis, this risk had to be mitigated by making the audio recording as discreet as possible. Of course, participants were asked for confirmation, but otherwise both the focus group and the expert interviews were conducted as a casual online/hybrid meeting. The focus group was conducted as an online meeting anyway, so the audio was simply recorded from the moderator’s laptop, and the expert interviews were conducted in rooms with hybrid meeting equipment that all participants were familiar with. This equipment was used for the audio recording in order to integrate it as much as possible into the experts’ daily work and avoid the effect of an unknown recorder being placed in direct line of sight between the moderator and the participants.

Another threat to the overall validity of the contributions is the fact that they are largely based on the requirements generated by the employees of the OEM who participated in the focus group. This may have put a commercial spin on the results, making them potentially less scientifically accurate. However, as this is an industrial case study, its main objective is to provide a scientific view of commercial and industrial concerns and opinions. As the other participants in the focus group were acting in their roles, they balanced the potentially extreme commercial view of the OEM employees.

Finally, two of the three experts interviewed to evaluate the architecture proposal were already part of the focus group. Since the architecture proposal was derived directly from the architectural requirements, the study design may have allowed both the product owner and the AI expert to accept the architecture proposal without much criticism because it already fulfilled their requirements collected in the focus group. Although this is not ideal, this risk had to be accepted because the roles of the experts are extremely important and there is only one product owner. The same applies to the AI expert, as only the two AI experts from the focus group were available for interview in this case study. To mitigate the threat, a different architecture expert was chosen who had not worked on the project before. This way, the architecture expert was not biased at all and was able to evaluate the architecture proposal from a purely architectural point of view. As it turned out, the familiarity of both the product owner and the AI expert with all the architectural requirements enabled them to perform a deeper functional evaluation of the architecture proposal.

7 Conclusion and Outlook

This chapter provides a summary of the paper and outlines potential areas for further research related to the topics addressed in this thesis.

7.1 Conclusion

This thesis dealt with the design process of the architecture of a document classification system that would be able to classify construction drawings based on the company that created them and be utilized in the product development process. The first step was to define the architectural requirements for such a software. For this purpose, a focus group was conducted during which participants from different domains were brought together to generate and discuss their requirements for the software. The focus group utilized a mix of empirical methods, namely brainwriting, card sorting, and dot voting. The requirements from the focus group served as a basis to develop an architecture proposal. During the development, several trade-offs between conflicting requirements had to be made. As a final step, the architecture was evaluated through a series of expert interviews. During the interviews, additional architectural requirements were identified and subsequently implemented in the architecture proposal.

The results of this thesis include a catalog of architectural requirements, consisting of requirements identified during both the focus group and the expert interviews. This catalog at least partially answers the first research question, although its completeness could be doubted, as discussed earlier in this thesis. The second result is an architecture proposal which aims to answer the second research question by providing guidance on how to fulfill the architectural requirements. However, due to contradictory requirements, it was not possible to fully answer the research questions for all requirements. Additionally, some requirements identified during expert interviews needed further research in order to be optimally fulfilled, which was not possible during this thesis due to time constraints. Nonetheless, solution strategies for these requirements were outlined, but they were not sufficiently substantiated to be considered a result of the thesis.

This study contributes a concrete catalog of architectural requirements and a corresponding architecture proposal which can be applied to related problems in other domains. Depending on the extent to which the problem statement differs from the thesis case, it may be necessary to make some adjustments for both results to be fully applicable. The prioritization of architectural requirements and the division into mandatory and non-mandatory requirements, which are also results of this thesis, are less applicable to other cases because the prioritization is more case-specific than the requirements in general. To the knowledge of the author, there is neither an empirically based catalog of architectural requirements nor an exemplary example of how to fulfill the requirements for software similar to the one described above.

7.2 Future Work

Future research could collect architectural requirements to verify or extend the catalog presented in this thesis. This could be accomplished using various approaches, such as empirical methods that are similar to the focus group and expert interviews performed during this study, or by conducting a literature review to make the result more robust through the use of different methods.

Since the current architecture does not entirely meet all architectural requirements, future research could focus on this issue. For this purpose, an anti-corruption mechanism could be realized, either by exploring ways to implement the four-eyes principle, or by developing another mechanism. Additionally, a study could concentrate on encryption extending the architecture to provide E2EE between the user and the AI algorithm.

Lastly, as the architecture is currently completely theoretical work, future studies could validate the architecture proposal by implementing a prototype. The prototype could then be used to evaluate the quality attributes among the architectural requirements, as these are difficult to measure based only on a theoretical architecture.

Bibliography

[AAM+20] M. Alloghani, D. Al-Jumeily, J. Mustafina, A. Hussain, A. J. Aljaaf. “A Systematic Review on Supervised and Unsupervised Machine Learning Algorithms for Data Science”. en. In: *Supervised and Unsupervised Learning for Data Science*. Ed. by M. W. Berry, A. Mohamed, B. W. Yap. Unsupervised and Semi-Supervised Learning. Cham: Springer International Publishing, 2020, pp. 3–21. ISBN: 978-3-030-22474-5. DOI: 10.1007/978-3-030-22475-2_1 (cit. on p. 18).

[ACC+01] E. Appiani, F. Cesarini, A. Colla, M. Diligenti, M. Gori, S. Marinai, G. Soda. “Automatic document classification and indexing in high-volume applications”. en. In: *International Journal on Document Analysis and Recognition* 4.2 (Dec. 2001), pp. 69–83. ISSN: 1433-2833. DOI: 10.1007/PL00010904 (cit. on pp. 24, 52).

[Alb03] S. T. Albin. *The Art of Software Architecture: Design Methods and Techniques*. en. John Wiley & Sons, Mar. 2003, p. 25. ISBN: 978-0-471-46829-5 (cit. on p. 28).

[ANK18] J. Alzubi, A. Nayyar, A. Kumar. “Machine Learning from Theory to Algorithms: An Overview”. en. In: *Journal of Physics: Conference Series* 1142.1 (Nov. 2018), p. 012012. ISSN: 1742-6596. DOI: 10.1088/1742-6596/1142/1/012012 (cit. on p. 17).

[BBL06] M.-F. Balcan, A. Beygelzimer, J. Langford. “Agnostic active learning”. In: *Proceedings of the 23rd international conference on Machine learning*. ICML ’06. New York, NY, USA: Association for Computing Machinery, June 2006, pp. 65–72. ISBN: 978-1-59593-383-6. DOI: 10.1145/1143844.1143853 (cit. on p. 18).

[BMR+13] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. *Pattern-Oriented Software Architecture, A System of Patterns*. en. John Wiley & Sons, Apr. 2013, pp. 3–14. ISBN: 978-1-118-72526-9 (cit. on p. 20).

[Bro15] S. Brown. *The C4 model for visualising software architecture*. eng. Leanpub, Aug. 2015. URL: https://leanpub.next/visualising-software-architecture (cit. on pp. 27, 28).

[BW97] S. Blott, R. Weber. “A Simple Vector-Approximation File for Similarity Search in High-Dimensional Vector Spaces”. en. In: *ESPRIT Technical Report TR19, ca* (1997) (cit. on p. 19).

[CB07] N. Chen, D. Blostein. “A survey of document image classification: problem statement, classifier architecture and performance evaluation”. en. In: *International Journal of Document Analysis and Recognition (IJDAR)* 10.1 (June 2007), pp. 1–16. ISSN: 1433-2825. DOI: 10.1007/s10032-006-0020-2 (cit. on p. 17).

[CGG+09] Y. Chen, E. K. Garcia, M. R. Gupta, A. Rahimi, L. Cazzanti. “Similarity-based Classification: Concepts and Algorithms”. en. In: *Journal of Machine Learning Research* 10 (2009), pp. 747–776 (cit. on p. 19).

[Dal19] J. Dalton. “Dot Voting”. In: *Great Big Agile: An OS for Agile Leaders*. Berkeley, CA: Apress, 2019, pp. 165–166. ISBN: 978-1-4842-4206-3. DOI: 10.1007/978-1-4842-4206-3_27 (cit. on p. 27).

[DB18] L. Ding, M. Ben Salem. “A Novel Architecture for Automatic Document Classification for Effective Security in Edge Computing Environments”. In: *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. Oct. 2018, pp. 416–420. DOI: 10.1109/SEC.2018.00056 (cit. on pp. 23, 52).

[Dea09] J. Deacon. “Model-View-Controller (MVC) Architecture”. en. In: (2009). URL: https://www.academia.edu/30077059/Model_View_Controller_MVC_Architecture (cit. on pp. 20, 34).

[Dey11] T. Dey. “A Comparative Analysis on Modeling and Implementing with MVC Architecture”. en. In: *International Journal of Computer Applications* (2011) (cit. on p. 21).

[DP18] A. Demirsoy, K. Petersen. “Semantic Knowledge Management System to Support Software Engineers: Implementation and Static Evaluation through Interviews at Ericsson”. en. In: *e-Informatica Vol. XII* (2018), 2018, ISSN 18977979. DOI: 10.5277/E-INF180110 (cit. on p. 29).

[DW14] H. Deakin, K. Wakefield. “Skype interviewing: reflections of two PhD researchers”. en. In: *Qualitative Research* 14.5 (Oct. 2014), pp. 603–616. ISSN: 1468-7941. DOI: 10.1177/1468794113488126 (cit. on p. 56).

[DWD+16] S. Das, W.-K. Wong, T. Dietterich, A. Fern, A. Emmott. “Incorporating Expert Feedback into Active Anomaly Discovery”. In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. Dec. 2016, pp. 853–858. DOI: 10.1109/ICDM.2016.0102 (cit. on p. 18).

[Ech20] K. Echihabi. “High-Dimensional Vector Similarity Search: From Time Series to Deep Network Embeddings”. en. In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. Portland OR USA: ACM, June 2020, pp. 2829–2832. ISBN: 978-1-4503-6735-6. DOI: 10.1145/3318464.3384402 (cit. on p. 19).

[EZP21] K. Echihabi, K. Zoumpatianos, T. Palpanas. “New trends in high-D vector similarity search: al-driven, progressive, and distributed”. In: *Proceedings of the VLDB Endowment* 14.12 (July 2021), pp. 3198–3201. ISSN: 2150-8097. DOI: 10.14778/3476311.3476407 (cit. on p. 19).

[GE06] I. Guyon, A. Elisseeff. “An Introduction to Feature Extraction”. en. In: *Feature Extraction: Foundations and Applications*. Ed. by I. Guyon, M. Nikravesh, S. Gunn, L. A. Zadeh. Studies in Fuzziness and Soft Computing. Berlin, Heidelberg: Springer, 2006, pp. 1–25. ISBN: 978-3-540-35488-8. DOI: 10.1007/978-3-540-35488-8_1 (cit. on p. 19).

[GG10] P. Gupta, M. C. Govil. “MVC Design Pattern for the multi framework distributed applications using XML, spring and struts framework”. en. In: 02.04 (2010) (cit. on p. 21).

[Har89] S. J. Hart. “Collection and Analysis of Interview Data”. en. In: *Marketing Intelligence & Planning* 7.5/6 (May 1989), pp. 23–29. ISSN: 0263-4503. DOI: 10.1108/EUM0000000001046 (cit. on p. 31).

[HKM19] P. Hendradi, M. Khanapi, S. N. Mahfuzah. "Cloud Computing-Based E-Learning System Architecture in Education 4.0". en. In: *Journal of Physics: Conference Series* 1196.1 (Mar. 2019), p. 012038. ISSN: 1742-6596. DOI: 10.1088/1742-6596/1196/1/012038 (cit. on pp. 24, 53).

[Hut09] D. Hutton. "Clean Code: A Handbook of Agile Software Craftsmanship". In: *Kybernetes* 38.6 (Jan. 2009), pp. 1035–1035. ISSN: 0368-492X. DOI: 10.1108/03684920910973252 (cit. on p. 36).

[IHO02] M. T. Ionita, D. K. Hammer, H. Obbink. "Scenario-Based Software Architecture Evaluation Methods: An Overview". en. In: 2002. URL: https://api.semanticscholar.org/CorpusID:7874705 (cit. on p. 30).

[ISO11] International Organization for Standardization. *ISO/IEC 25010:2011*. en. 2011. URL: https://www.iso.org/standard/35733.html (cit. on pp. 35, 50).

[JJ23] T. M. Jung, I. Joe. "An AI-Based Platform Architecture with Situational Awareness for Travel Plans". en. In: *Software Engineering Application in Systems Design*. Ed. by R. Silhavy, P. Silhavy, Z. Prokopova. Lecture Notes in Networks and Systems. Cham: Springer International Publishing, 2023, pp. 376–384. ISBN: 978-3-031-21435-6. DOI: 10.1007/978-3-031-21435-6_32 (cit. on pp. 24, 53).

[JLL+22] Y. Jiang, X. Li, H. Luo, S. Yin, O. Kaynak. "Quo vadis artificial intelligence?" en. In: *Discover Artificial Intelligence* 2.1 (Mar. 2022), p. 4. ISSN: 2731-0809. DOI: 10.1007/s44163-022-00022-8 (cit. on p. 17).

[KAK01] R. Kazman, J. Asundi, M. Klein. "Quantifying the costs and benefits of architectural decisions". en. In: *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001*. Toronto, Ont., Canada: IEEE Comput. Soc, 2001, pp. 297–306. ISBN: 978-0-7695-1050-7. DOI: 10.1109/ICSE.2001.919103 (cit. on pp. 23, 49, 53).

[KBAW94] R. Kazman, L. Bass, G. Abowd, M. Webb. "SAAM: a method for analyzing the properties of software architectures". In: *Proceedings of 16th International Conference on Software Engineering*. May 1994, pp. 81–90. DOI: 10.1109/ICSE.1994.296768 (cit. on pp. 30, 31, 52).

[KBL08] J. Kontio, J. Bragge, L. Lehtola. "The focus group method as an empirical tool in software engineering". In: *Guide to advanced empirical software engineering*. Springer, 2008, pp. 93–116 (cit. on p. 26).

[KCK14] M. Kalelkar, P. Churi, D. Kalelkar. "Implementation of Model-View-Controller Architecture Pattern for Business Intelligence Architecture". en. In: *International Journal of Computer Applications* 102.12 (Sept. 2014), pp. 16–21. ISSN: 09758887. DOI: 10.5120/17867-8786 (cit. on p. 21).

[KIC05] R. Kazman, H. P. In, H.-M. Chen. "From requirements negotiation to software architecture decisions". In: *Information and Software Technology* 47.8 (June 2005), pp. 511–520. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2004.10.001 (cit. on pp. 23, 50).

[KJ13] M. Kircher, P. Jain. *Pattern-Oriented Software Architecture, Patterns for Resource Management*. en. John Wiley & Sons, Apr. 2013, pp. 9–10. ISBN: 978-1-118-72523-8 (cit. on p. 20).

[KKC00] R. Kazman, M. Klein, P. Clements. *ATAM: Method for Architecture Evaluation:* en. Fort Belvoir, VA, Aug. 2000. DOI: 10.21236/ADA382629 (cit. on p. 30).

[KOS+96] T. Kohonen, E. Oja, O. Simula, A. Visa, J. Kangas. "Engineering applications of the self-organizing map". In: *Proceedings of the IEEE* 84.10 (Oct. 1996), pp. 1358–1384. ISSN: 1558-2256. DOI: 10.1109/5.537105 (cit. on p. 18).

[KSTT84] N. Kano, N. Seraku, F. Takahashi, S.-i. Tsuji. "Attractive Quality and Must-Be Quality". In: *Journal of The Japanese Society for Quality Control* 14.2 (1984), pp. 147–156. DOI: 10.20684/quality.14.2_147 (cit. on p. 49).

[LTWL10] R. K. C. Lai, J. C. K. Tang, A. K. Y. Wong, P. I. S. Lei. "Design and Implementation of an Online Social Network with Face Recognition". en. In: *Journal of Advances in Information Technology* 1.1 (Feb. 2010), pp. 38–42. ISSN: 1798-2340. DOI: 10.4304/jait.1.1.38-42 (cit. on pp. 24, 53).

[Mar11] S. Marsland. *Machine Learning: An Algorithmic Perspective*. New York: Chapman and Hall/CRC, Oct. 2011, pp. 6–11. ISBN: 978-0-429-14038-9. DOI: 10.1201/9781420067194 (cit. on pp. 17, 18).

[May94] P. Mayring. "Qualitative Inhaltsanalyse". In: *Texte verstehen : Konzepte, Methoden, Werkzeuge*. Ed. by A. Boehm, A. Mengel, T. Muhr. Vol. 14. Schriften zur Informationswissenschaft. Konstanz: UVK Univ.-Verl. Konstanz, 1994, pp. 159–175. ISBN: 3-87940-503-4 (cit. on p. 31).

[MM21] R. Monarch, R. Munro. *Human-in-the-Loop Machine Learning: Active Learning and Annotation for Human-centered AI*. en. Simon and Schuster, July 2021. ISBN: 978-1-61729-674-1 (cit. on p. 19).

[MN07] M. D. Myers, M. Newman. "The qualitative interview in IS research: Examining the craft". In: *Information and Organization* 17.1 (Jan. 2007), pp. 2–26. ISSN: 1471-7727. DOI: 10.1016/j.infoandorg.2006.11.001 (cit. on p. 30).

[MNR+10] J. Magenheim, W. Nelles, T. Rhode, N. Schaper, S. Schubert, P. Stechert. "Competencies for informatics systems and modeling: Results of qualitative content analysis of expert interviews". en. In: *IEEE EDUCON 2010 Conference*. Madrid, Spain: IEEE, 2010, pp. 513–521. ISBN: 978-1-4244-6568-2. DOI: 10.1109/EDUCON.2010.5492535 (cit. on p. 31).

[MP14] D. Martens, F. Provost. "Explaining Data-Driven Document Classifications". In: *MIS Quarterly* 38.1 (2014), pp. 73–100. ISSN: 0276-7783 (cit. on p. 17).

[MQ11] H. Mcheick, Y. Qi. "Dependency of components in MVC distributed architecture". In: *2011 24th Canadian Conference on Electrical and Computer Engineering(CCECE)*. May 2011, pp. 691–694. DOI: 10.1109/CCECE.2011.6030542 (cit. on pp. 20, 21, 51).

[Nas23] W. Nasar. "Impact of Expert Interviews in Software Engineering: Challenges and Benefits". en. In: *Hong Kong* (2023) (cit. on pp. 31, 56).

[Par86] D. Partridge. "Engineering artificial intelligence software". en. In: *Artificial Intelligence Review* 1.1 (Mar. 1986), pp. 27–41. ISSN: 1573-7462. DOI: 10.1007/BF01988526 (cit. on p. 17).

[PGY+22] P. Ponnusamy, A. Ghias, Y. Yi, B. Yao, C. Guo, R. Sarikaya. "Feedback-Based Self-Learning in Large-Scale Conversational AI Agents". en. In: *AI Magazine* 42.44 (Jan. 2022), pp. 43–56. ISSN: 2371-9621. DOI: 10.1609/aaai.12025 (cit. on p. 19).

[Raj20] B. Rajoub. “Chapter 3 - Supervised and unsupervised learning”. In: *Biomedical Signal Processing and Artificial Intelligence in Healthcare*. Ed. by W. Zgallai. Developments in Biomedical Engineering and Bioelectronics. Academic Press, Jan. 2020, pp. 51–89. DOI: 10.1016/B978-0-12-818946-7.00003-2 (cit. on pp. 17, 18).

[RH09] P. Runeson, M. Höst. “Guidelines for conducting and reporting case study research in software engineering”. en. In: *Empirical Software Engineering* 14.2 (Apr. 2009), pp. 131–164. ISSN: 1573-7616. DOI: 10.1007/s10664-008-9102-8 (cit. on pp. 29, 53).

[RJKG11] B. P. Rimal, A. Jukan, D. Katsaros, Y. Goeleven. “Architectural Requirements for Cloud Computing Systems: An Enterprise Cloud Approach”. en. In: *Journal of Grid Computing* 9.1 (Mar. 2011), pp. 3–26. ISSN: 1572-9184. DOI: 10.1007/s10723-010-9171-y (cit. on pp. 23, 50).

[RKS11] N. Rubens, D. Kaplan, M. Sugiyama. “Active Learning in Recommender Systems”. In: *Recommender Systems Handbook*. Ed. by P. Kantor, F. Ricci, L. Rokach, B. Shapira. Springer, 2011, pp. 735–767. DOI: 10.1007/978-0-387-85820-3_23 (cit. on p. 18).

[RRC19] G. Rebala, A. Ravi, S. Churiwala. “Machine Learning Definition and Basics”. en. In: *An Introduction to Machine Learning*. Cham: Springer International Publishing, 2019, pp. 1–17. ISBN: 978-3-030-15728-9. DOI: 10.1007/978-3-030-15729-6_1 (cit. on p. 18).

[SA13] R. Sathya, A. Abraham. “Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification”. en. In: *International Journal of Advanced Research in Artificial Intelligence* 2.2 (2013), pp. 34–38. ISSN: 2165-4069, 2165-4050 (cit. on pp. 17, 18).

[SCD06] D. M. Selfa, M. Carrillo, M. Del Rocio Boone. “A Database and Web Application Based on MVC Architecture”. In: *16th International Conference on Electronics, Communications and Computers (CONIELECOMP’06)*. 2006, pp. 48–48. DOI: 10.1109/CONIELECOMP.2006.6 (cit. on p. 20).

[Sch97] K. Schwaber. “SCRUM Development Process”. en. In: *Business Object Design and Implementation*. Ed. by J. Sutherland, C. Casanave, J. Miller, P. Patel, G. Hollowell. London: Springer, 1997, pp. 117–134. ISBN: 978-1-4471-0947-1. DOI: 10.1007/978-1-4471-0947-1_11 (cit. on p. 29).

[SD19] R. Sharma, K. Davuluri. “Design patterns for Machine Learning Applications”. In: *2019 3rd International Conference on Computing Methodologies and Communication (ICCMC)*. Mar. 2019, pp. 818–821. DOI: 10.1109/ICCMC.2019.8819692 (cit. on p. 38).

[Sea99] C. Seaman. “Qualitative methods in empirical studies of software engineering”. In: *IEEE Transactions on Software Engineering* 25.4 (July 1999), pp. 557–572. ISSN: 1939-3520. DOI: 10.1109/32.799955 (cit. on p. 29).

[Set10] B. Settles. “Active Learning Literature Survey”. en. In: *Computer Sciences Technical Report 1648* (2010) (cit. on p. 18).

[SJ06] M. Sugiyama, C. T. A. Jp. “Active Learning in Approximately Linear Regression Based on Conditional Expectation of Generalization Error”. en. In: *Journal of Machine Learning Research 7* (2006), pp. 141–166 (cit. on p. 18).

[SSR+21] L. W. Santoso, B. Singh, S. S. Rajest, R. Regin, K. H. Kadhim. “A Genetic Programming Approach to Binary Classification Problem”. en. In: *EAI Endorsed Transactions on Energy Web* 8.31 (2021), e11–e11. ISSN: 2032-944X. DOI: 10.4108/eai.13-7-2018.165523 (cit. on p. 17).

[Van84] A. B. VanGundy. “Brain writing for new product ideas: An alternative to brainstorming”. In: *Journal of Consumer Marketing* 1.2 (1984), pp. 67–74. DOI: 10.1108/eb008097 (cit. on pp. 27, 56).

[WG06] C. B. Weinstock, J. B. Goodenough. *On System Scalability:* en. Fort Belvoir, VA, Mar. 2006. DOI: 10.21236/ADA457003 (cit. on p. 35).

[WUKG19] H. Washizaki, H. Uchida, F. Khomh, Y.-G. Guéhéneuc. “Studying Software Engineering Patterns for Designing Machine Learning Systems”. In: *2019 10th International Workshop on Empirical Software Engineering in Practice (IWESEP)*. Dec. 2019, pp. 49–495. DOI: 10.1109/IWESEP49350.2019.00017 (cit. on p. 38).

[WW08] J. R. Wood, L. E. Wood. “Card Sorting: Current Practices and Beyond”. In: *Journal of Usability Studies* (2008) (cit. on pp. 27, 31).

[Zim16] T. Zimmermann. “Card-sorting: From text to themes”. In: *Perspectives on Data Science for Software Engineering*. Ed. by T. Menzies, L. Williams, T. Zimmermann. Boston: Morgan Kaufmann, 2016, pp. 137–141. ISBN: 978-0-12-804206-9. DOI: https://doi.org/10.1016/B978-0-12-804206-9.00027-1 (cit. on p. 27).

All links were last followed on October 31, 2023.

A Focus Group Moderator's Guide

General Information about the Focus Group

Goal	Gather architectural requirements for an AI-based image recognition software.
Participants	8 (1 architecture expert, 3 OEM-clients, 1 product owner, 1 operator, 2 AI-experts)
Moderator	Patrick Reich
Date and Time	June 29, 2023, 10:00 - 12:00
Place	Remote (Microsoft Teams meeting)

A.1 Introduction and Consent

- Introduce myself
- Get verbal consent from everyone, that the focus group will be audio-recorded
- Start the audio recording
- Brief presentation of the objective of my study and the purpose of the focus group
- Short description of the AI algorithm as the basis of the architecture
- Quick overview of the focus group method

A.2 Ground Rules

- All group members have a right to their viewpoints and opinions. Ideas that do not make immediate sense are also considered and discussed.
- I, as moderator, will try to keep the group on time, so I may interrupt discussions that are taking too long.
- We will not discuss the inner architecture of the AI-algorithm, but the architecture around that.

A.3 Method

The main part of the focus group will be conducted using an online visual collaboration platform called Miro Board. All participants have access to this and can create, move and categorize sticky notes.
The Miro Board has been prepared in advance so that there are enough sticky notes and small dots for each participant. These have a different color for each participant so the results can be assigned to the respective participant during the evaluation.
For each of the following steps, copy the results of the previous step to a new sheet, to make sure that the process can be reconstructed afterwards.

A.3.1 Brainwriting

Goal	Gathering architectural requirements
Key question	Which requirements exist for the architecture of the software?
Duration	10 minutes
Method	Participants silently write down any architectural requirements that come to mind on sticky notes.

- Mention that each participant should only write on sticky notes in his[1] color.
- After 5-6 Minutes provide some categories from the Possible Assistance section that fell short until now.
- Let each participant present his results.

A.3.2 Card Sorting

Goal	Categorize the architectural requirements from the Brainwriting phase and give each category a meaningful name. Also discuss each requirement in terms of its relevance to the system.
Key question	How would you (as a group) categorize the architectural requirements? Is each requirement relevant to the system? Are there any requirements that follow from the existing ones?
Duration	About 1 hour (15 minutes clustering, 10 minutes naming, 10 minutes per category)
Method	First, the requirements previously noted are clustered and each category is given a descriptive name. Participants then go through each category and discuss the relevance of the respective requirements for the system and whether further requirements follow from them.

- First, let the participants cluster the requirements.

[1]This guide uses the generic masculine for readability reasons. Female and other gender identities are explicitly included as far as it is necessary for the statement.

- Then, let them name each category.
- After that, ask category by category for the relevance of the respective requirements and if there are any further requirements for this category.

A.3.3 Dot Voting

Goal	Prioritization of results, division into mandatory and excitement requirements
Key question	How important do you think each of the requirements is?
Duration	8 minutes (5 minutes voting, 3 minutes split to requirement categories)
Method	Each participant gets 10 colored points, which can be distributed to individual requirements as he likes. The points should be distributed according to the importance of the requirement for the system in their opinion.

- Take a break before this step so I can adjust the number of colored points for each participant (10 might be too many)
- Mention that each participant should use the same colour as he used for the brain writing.
- Mention that one can assign more than one dot per sticky note.
- After the voting, split the prioritization into mandatory and excitement requirements

Possible Assistance

If participants need suggestions, or if certain aspects fall short, reference can be made to some of the following categories:

- Integration and/or interfaces to existing systems
- Web service or local installation
- GUI or console
- Data flow
  - Data input
  - Is some kind of persistence necessary?
  - Data security
  - Presentation of results
- Automation
- Quality attributes

A.4 Wrap Up and Thank You

- Thank the participants for attending.
- Inform the participants that they will receive the transcript as soon as it is created to validate that all statements have been interpreted correctly.
- Brief presentation on how the results will be used and what the next steps are

B Expert Interview Guide

General Information about the Expert Interviews

Goal	Evaluation of the proposed architecture design
Interview type	Semi-structured, one-on-one
Interviewees	1 architecture-expert, 1 AI-expert, 1 product owner
Interviewer	Patrick Reich
Duration	About 1 hour per interview
Date	September 2023
Place	Böblingen, Germany (in presence)

B.1 Introduction and Consent

- Introduce myself
- Get verbal consent from the interviewee, that the focus group will be audio-recorded
- Start the audio recording
- Inform the interviewee of their role
- Ask for the interviewee's experience in software engineering, software architecture and AI
- Ask how long the interviewee has been working on the project
- Quick recap of the interview preamble and the results of the focus group
- Brief presentation of the interview methodology

B.2 Evaluation

The evaluation of the proposed architecture design is the main part of the expert interviews. An aid for this section is a beamer or large screen on which the various architecture diagrams are displayed. This way both parties can clarify which part of the architecture they are talking about.

B.2.1 Comprehension Questions

Goal	Eliminate problems of understanding to avoid misunderstandings later in the interview
Duration	10 minutes

- Ask the interviewee if they had difficulties understanding anything of the architecture proposal
- If they had difficulties, ask how the comprehensibility of the diagram/explanation can be improved

B.2.2 Feedback & Improvement Suggestions

Goal	Eliminate problems of understanding to avoid misunderstandings later in the interview
Duration	15 minutes

- Ask for general feedback
    - Try to lead criticism to improvement suggestions
    - If decisions are questioned, try to find out where I went wrong in their opinion
- What do they think of...
    - The distribution of components/containers into the MVC structure
    - The controller functionality and distribution
    - The buffer structure
    - The information flow (Are there possible shortcuts?)

B.2.3 SAAM

Goal	Evaluate the most important quality attributes
Duration	25 minutes
Method	Present the quality attributes to be evaluated 2. Make up 1-2 scenarios for each quality attribute 3. Rate the fulfillment of each quality attribute based on the “use cases”

- For this task, the three most important quality attributes from the focus group are shown on the screen as well
- The interviewee should rate the quality attributes based on my prepared scenario as well as their own 1-2 scenarios
- If a quality attribute is rated poorly, ask the interviewee what needs to be changed to fulfill it better

The following quality attributes with their standard scenarios will be evaluated:

Maintainability We want to add a backup mechanism that automatically sends feedback requests if the AI algorithm container crashes.

Compliance An external attacker tries to get access to a construction drawing somewhere in the system.

Scalability There is a sudden rise of users from a few hundred to 10.000.

B.3 Wrap Up and Thank You

- Thank the interviewee for attending
- Inform the interviewee of the next steps
  - Transcript will be sent to the interviewee for validation
  - After all interviews are finished: extension of the architecture requirements catalogue, improvement of the architecture proposal

C Expert Interview Preamble

In diesem Dokument werden die allgemeinen Leitlinien für die Reihe an Experteninterviews, die im Rahmen meiner Bachelorarbeit durchgeführt werden, zusammengefasst. Appendix C.3 und Appendix C.4 orientieren sich an der Interview Präambel von Bogner et al.[1].

C.1 Ziel der Studie

Ziel der Studie ist es ein fundiertes Architekturdesign für das im Nachfolgenden beschriebene System zu entwickeln. Dafür wurden bereits konkrete Anforderungen im Rahmen einer Fokusgruppe herausgearbeitet, welche ebenfalls im Folgenden zusammengefasst sind.

Bei dem System handelt es sich um ein KI-basiertes Bilderkennungstool, das die Klassifizierung von Konstruktionszeichnungen übernehmen soll. Die Klassifizierung erfolgt dabei nach der Urheberfirma der Zeichnung, entweder in interne OEM-Zeichnungen, oder in Lieferantenzeichnungen. Das System soll zentral in der Cloud gehostet werden, und sowohl von menschlichen Benutzern über eine GUI, als auch von anderen Systemen über eine Zentrale Schnittstelle genutzt werden können. Während der Fokusgruppe wurde konkret eine 3-Schichten-Architektur wie MVC erwähnt. Außerdem soll eine Art Backpropagation Mechanismus vorhanden sein, was so viel bedeutet wie "das System soll während es eingesetzt wird weiter dazulernen". Zusätzlich werden einige Qualitätsattribute vorausgesetzt. Hierzu gehören Scalability, Cost-efficiency, Reliability, Compliance, Security, Performance-efficiency und Maintainability.

Da der die Entwicklung des KI-Algorithmus, der die Klassifizierung durchführt nicht Teil dieser Studie ist, werden sich auch die Interviews nur am Rande mit dem KI-Algorithmus beschäftigen.

C.2 Ziel der Experteninterviews

Ziel der Experteninterviews ist es, einen bereits ausgearbeiteten Architekturvorschlag zu evaluieren. Dieser soll anschließend mit dem Feedback aus den Interviews weiter verbessert, und schlussendlich implementiert werden.

[1] https://github.com/xJREB/research-microservices-interviews/blob/master/interview-preamble.md

C.3 Vertraulichkeit und Anonymität

Alles, was in den Interviews gesagt wird, wird streng vertraulich behandelt und ohne die Erlaubnis der Teilnehmer an niemanden weitergegeben. Die Ergebnisse werden vor der Veröffentlichung aggregiert und anonymisiert. Einzelne Aussagen können in keiner Weise auf einen Mitarbeiter zurückgeführt werden. Auch nach dem Interview haben die Teilnehmer die Möglichkeit, Aussagen vor der Auswertung auszuschließen oder zu schwärzen. Die vollständigen Interviewtranskripte werden NICHT veröffentlicht.

C.4 Durchführung und Analyse

Die Befragten werden über das allgemeine Thema der Studie informiert, sodass sie damit vertraut sind und sich vorab informieren können, falls sie dies wünschen. Dazu gehört, dass die Teilnehmer auch den Architekturvorschlag mitsamt Erläuterung bereits im Vorhinein erhalten. Die Dauer des Interviews wird auf ca. 60 Minuten geschätzt. Während des Interviews wird eine Audioaufzeichnung laufen, um die spätere Auswertung zu erleichtern. Das Transkript wird den Teilnehmern zur endgültigen Validierung zugeschickt. Dies ist die letzte Möglichkeit, Aussagen aus dem Interview zu entfernen oder zu kürzen. Danach wird die Tonaufnahme vernichtet, und das eventuell angepasste und genehmigte Transkript wird die einzige Grundlage für die qualitative Inhaltsanalyse sein.

C.4.1 SAAM

Ein wesentlicher Teil der Interviews wird mit Hilfe einer Softwarearchitektur-Analysemethode namens SAAM durchgeführt. Um Zeit in den Interviews zu sparen, wird die Methode bereits hier vorgestellt.

SAAM wird verwendet, um Qualitätsattribute von Architekturentwürfen zu bewerten. Dazu werden zunächst mehrere Szenarien entworfen, in denen das jeweilige Qualitätsattribut relevant ist. Anschließend werden diese Szenarien durchgespielt und anhand dessen bewertet, inwiefern das Qualitätsattribut erfüllt ist.

C.5 Material

Zusätzlich zu diesem Dokument erhält jeder Teilnehmer bereits im Vorhinein den Architekturvorschlag, bestehend aus vier Architekturdiagrammen und einem erläuternden Text.

D Expert Interview Material

Dieses Dokument enhält das Material, mit dem die Befragten bekannt sein sollten. Die Beschreibungen der Komponenten und Datentypen sind auf Englisch, da sie später so in meine Thesis kommen sollen, und das Übersetzen Unklarheiten produzieren könnte, die auf diesem Weg vermieden werden können.

D.1 Architekturdiagramme

Die Architekturdiagramme wurden nach dem C4 Modell[1] erstellt, daher zeigt Figure D.1 nur die Umgebung des zu entwickelnden Systems, während Figure D.2, Figure D.3 und Figure D.4 rekursiv einen Teil des vorherigen Diagramms in größerem Detailheitsgrad darstellen.

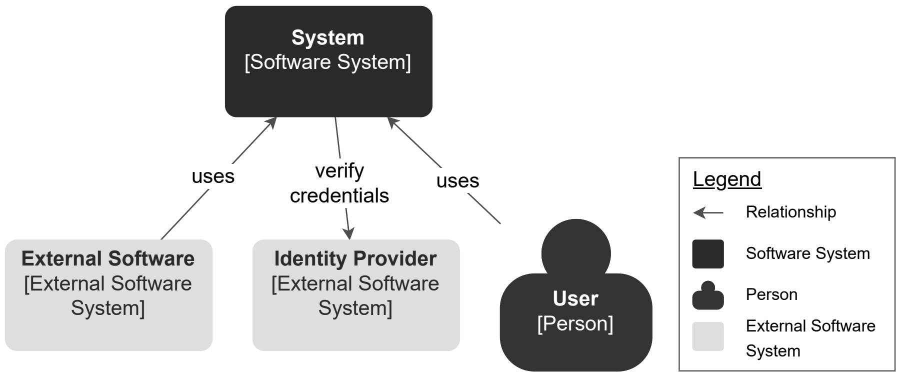


Figure D.1: C4 system context view

[1] https://c4model.com/

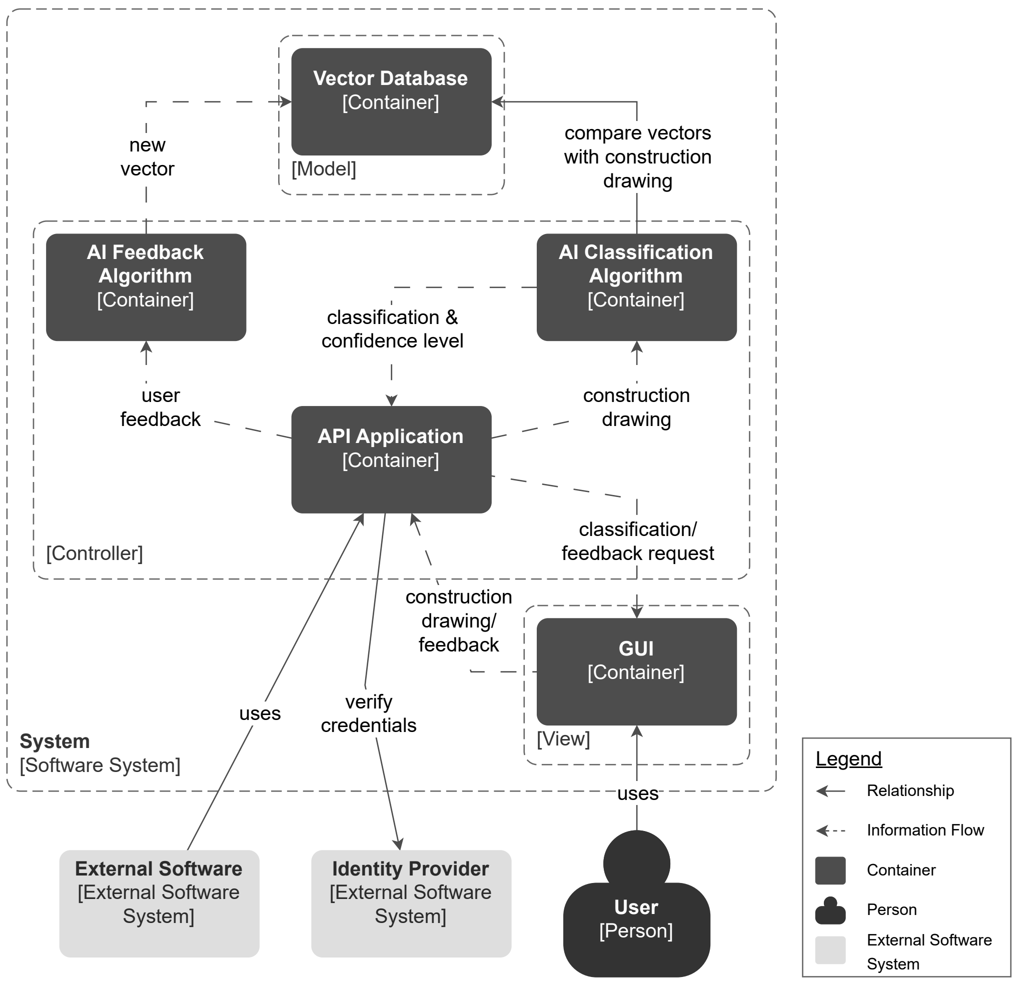


Figure D.2: C4 container view

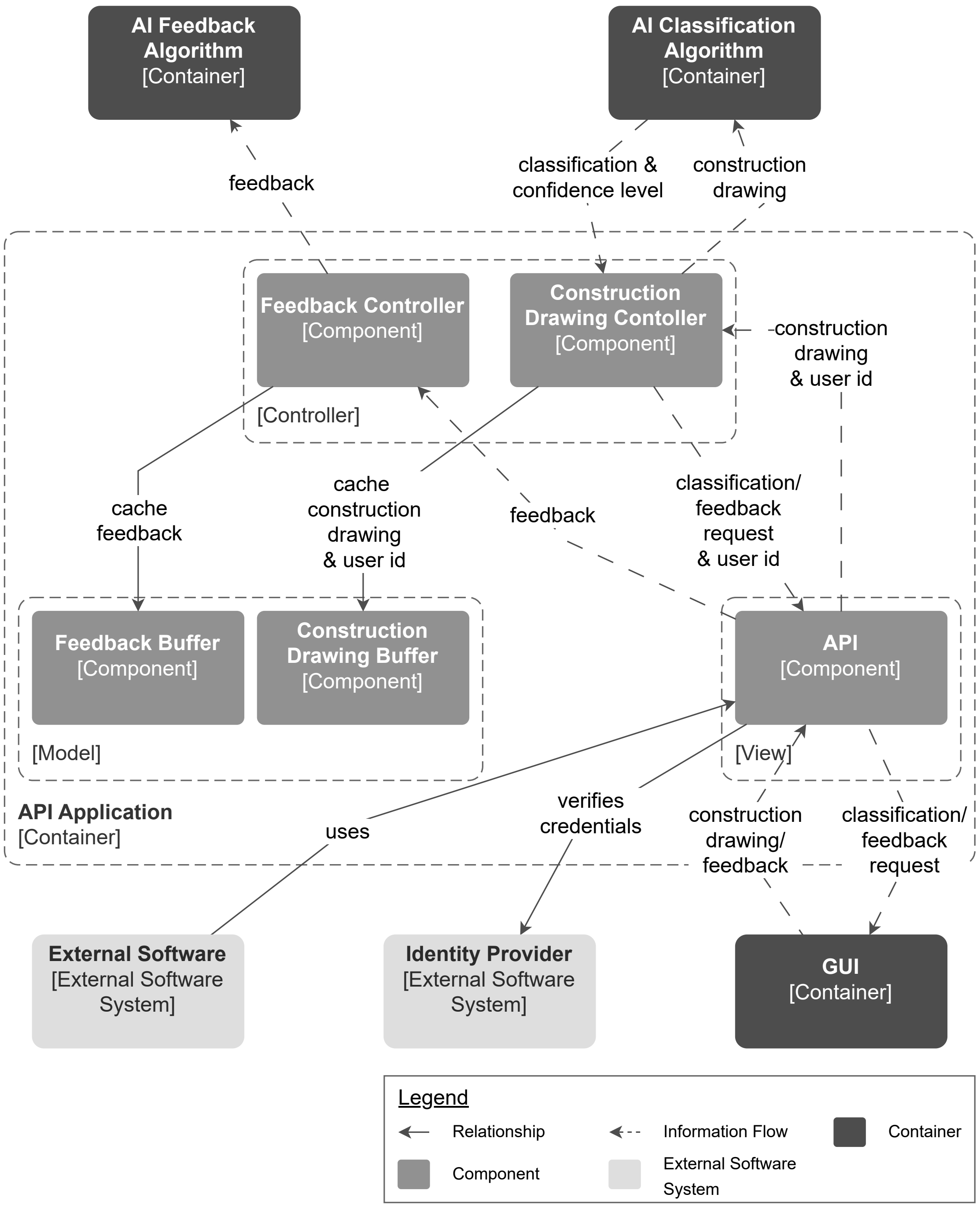

AI Feedback Algorithm [Container]
AI Classification Algorithm [Container]
feedback
classification & confidence level
construction drawing
Feedback Controller [Component]
Construction Drawing Contoller [Component]
construction drawing & user id
[Controller]
cache feedback
cache construction drawing & user id
feedback
classification/ feedback request & user id
Feedback Buffer [Component]
Construction Drawing Buffer [Component]
API [Component]
[Model]
[View]
API Application [Container]
uses
verifies credentials
construction drawing/ feedback
classification/ feedback request
External Software [External Software System]
Identity Provider [External Software System]
GUI [Container]
Legend
Relationship
Information Flow
Container
Component
External Software System

Figure D.3: C4 component view

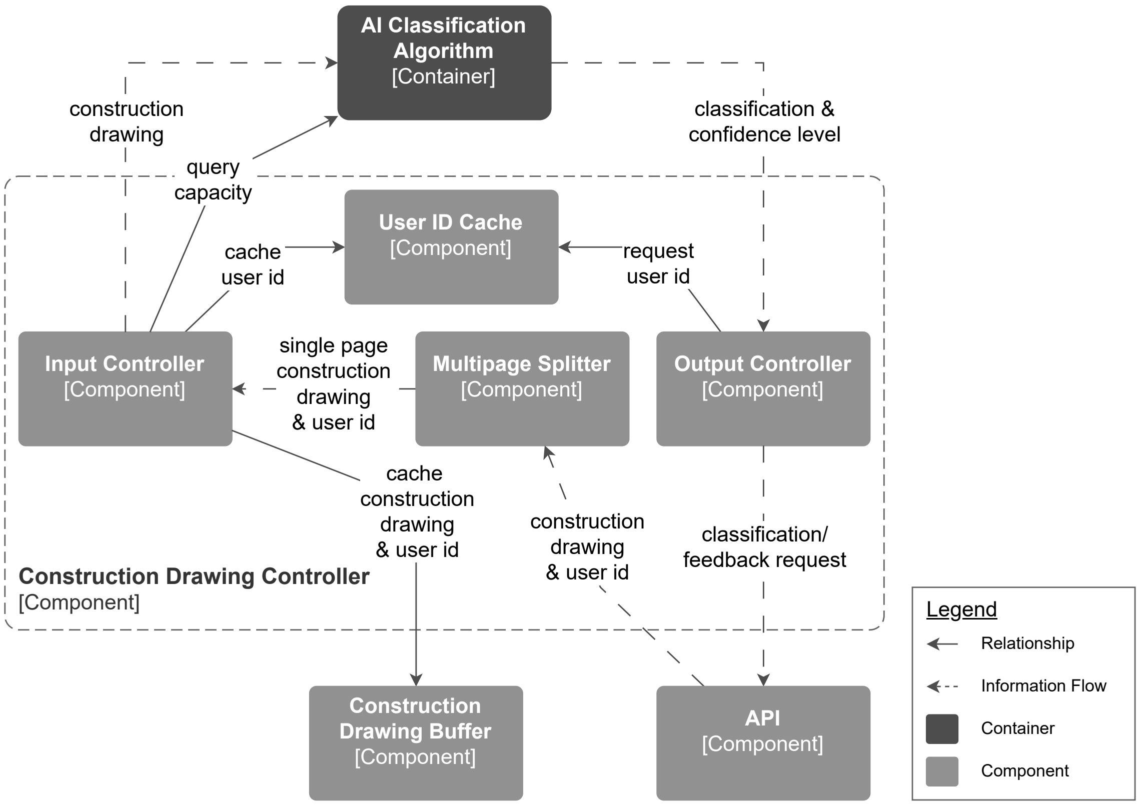

AI Classification Algorithm [Container]
construction drawing
query capacity
classification & confidence level
User ID Cache [Component]
cache user id
request user id
Input Controller [Component]
single page construction drawing & user id
Multipage Splitter [Component]
Output Controller [Component]
cache construction drawing & user id
construction drawing & user id
classification/ feedback request
Construction Drawing Controller [Component]
Construction Drawing Buffer [Component]
API [Component]
Legend
Relationship
Information Flow
Container
Component

Figure D.4: C4 component view of the Construction Drawing Controller

D.2 Datentypen

Construction Drawing A construction drawing is uploaded by a user or an External Software, who then awaits its classification.

Multipage Construction Drawing A multipage construction drawing consists of at least two pages, each containing a different construction drawing, possibly of different creator companies. Therefore, each page must be classified individually.

Feedback Feedback is given by the user, either in response to a feedback request from the system, or voluntarily, e.g. if a classification is incorrect. Feedback consists of a construction drawing and a simple boolean, stating that the construction drawing is either a supplier drawing or not.

Feedback Requests Feedback Requests are created by the Output Controller, if the confidence level of a classification is below a predefined threshold. They are sent to the user instead of the classification, requesting them to provide feedback for the construction drawing that the classification belonged to.

Classifications Classifications are generated by the AI Classification Algorithm and each belong to a specific construction drawing. A classification consists of a reference to the construction drawing and a simple boolean, stating that the construction drawing is either a supplier drawing or not.

D.3 Erklärende Beschreibung der einzelnen Komponenten

Software System The Software System accepts construction drawings from users or external software. It generates a classification for each construction drawing and sends it back to the user or external software.

User Users need to authenticate themselves before being able to upload any construction drawings. Each user has a unique user id, which is internally used to identify the owner of a construction drawing or classification.

Once they are authenticated, a user can upload a construction drawing to receive a classification of it. They then receive either the classification or a feedback request, prompting the user to classify the construction drawing by themselves.

After receiving a classification, the process would normally be complete. However, the user themselves can decide to give feedback, if the classification was wrong. It works the same as when the application prompts the user for feedback.

External Software External Software is similar to a normal user, with the one difference, that it communicates directly with the API instead of taking the detour over the GUI. External Software also needs to be authenticated, which is done by attaching a valid token to each request.

Even in an automated process, the external software should be able to handle a feedback request, as this can be the output of the system at any time, instead of a classification.

Identity Provider The identity provider shields the system from access by unauthenticated users. In addition, the identity provider receives authentication requests from the system, which it must process - usually by granting the user or external software access or returning a failure message.

API Application The API Application manages the business logic of the Software. As the API is part of if, External Software directly communicates with the API Application. It distributes construction drawings, classifications, feedback requests and feedback to the intended receivers, evaluates confidence levels of classifications, and creates feedback requests if said confidence level is too low.

GUI For an unauthenticated user, the GUI consists only of the login screen of the identity provider. Authenticated users can upload construction drawings here and get the classification displayed afterward. Furthermore, there is the possibility to give feedback to the construction drawings uploaded by them, which can be done by manually classifying the construction drawing.

AI Algorithm The AI Algorithm works with a vector comparison approach. For this purpose, header-tables of construction drawings are stored in a database in form of a vector. To classify a construction drawing, the algorithm derives a vector from the header-table of the construction drawing and compares it with each vector of the database using a method called similarity search. From the vector that is most similar to the one of the construction drawing, the classification and its confidence level are derived.

The AI Algorithm is divided into three main parts: The Vector Database, which stores all known vectors and thus is part of the model; The Classification Algorithm, which classifies construction drawings by performing the comparison of vectors; And the Feedback Algorithm, which processes feedback by converting the construction drawing header-table into a vector and inserting it together with the classification into the Vector Database.

Since both the classification of construction drawings and the processing of feedback are based on header-table-detection and vector computation, they are very resource intensive, and only a fixed number of tasks can run in parallel. Therefore, the AI Algorithm has only limited capacity.

API On one hand, the API is responsible to forward construction drawings and feedback received from a user or an external software to the respective controller. When forwarding a construction drawing, it also sends the user id of the user who uploaded it. On the other hand, it receives classifications or feedback requests from the Construction Drawing Controller, together with an associated user id, and is responsible to forward the output to the correct user.

Before being able to upload any construction drawings, a user needs to authenticate themselves, which is also handled by the API, by redirecting any authentication requests to the external Identity Provider.

Construction Drawing Controller The Construction Drawing Controller receives construction drawings and associated user ids from the API. Those are either forwarded to the AI Classification Algorithm if it has capacity, or sent to the Construction Drawing Buffer otherwise.

If a multipage construction drawing is received, it is split up into different documents, each containing exactly one construction drawing. These are then processed further like normal construction drawings.

Once the AI Classification Algorithm finishes a classification and thus has capacity again, the Construction Drawing Controller requests the next-in-line construction drawing from the Construction Drawing Buffer and sends it to the Algorithm.

Each construction drawing received from the API comes with a user id that belongs to the user who uploaded the construction drawing. When the construction drawing is sent to the Buffer, the user id is sent with it. The Construction Drawing Controller keeps track of all user ids associated to construction drawings that are currently being classified by the AI.

Upon receiving a classification from the AI, the Controller matches it with its associated user id before it sends a response to the API, requesting to send it to the associated user. The response can either be the classification received from the AI, or, if the confidence level of the classification is lower than a predefined threshold, a feedback request is sent instead of the classification.

Feedback Controller The Feedback Controller keeps track of the current capacity of the AI Feedback Algorithm and the number of feedback in the Feedback Buffer. Upon receiving feedback from the API, the Feedback Controller checks if the AI has capacity and no other feedback is waiting to be processed, and forwards the feedback to the AI Algorithm, if both checks pass. If there is no capacity or other feedback is waiting, the Feedback Controller sends the feedback to the Feedback Buffer.

After the AI Algorithm finishes the processing of feedback, and thus has capacity again to process another feedback, the Feedback Controller fetches the next-in-line feedback from the Feedback Buffer and sends it to the AI Algorithm.

Construction Drawing Buffer The Construction Drawing Buffer accepts construction drawings and associated user ids from the Construction Drawing Controller and stores both.

It keeps track of the order in which it received construction drawing user id pairs, hence it is able to determine which pair has been in the Buffer the longest. Upon request from the Construction Drawing Controller, it returns the next-in-line construction drawing user id pair.

Feedback Buffer The Feedback Buffer accepts feedback from the Feedback Controller and stores it. It keeps track of the order in which it received the feedback and is able to return it in this exact order, if requested to.

Multipage Splitter The Multipage Splitter construction drawings from the API. If it receives a multipage construction drawing, it splits it up into different documents, each containing exactly one single page construction drawing. These are then sent to the Input Controller together with the user id former belonging to the multipage drawing. Single page construction drawings together with their user id are simply redirected to the Input Controller.

Input Controller The Input Controller controls the input given to the AI Classification Algorithm. It keeps track of the capacity of said algorithm and sends construction drawings to the Construction Drawing Buffer if there is no capacity. Furthermore, if a classification finishes and the AI Classification Algorithm has capacity again, it requests a construction drawing from the Construction Drawing Buffer and sends it to the AI Classification Algorithm. While doing that, the user id that belongs to the construction drawing is sent to the User ID Cache, together with a reference to the construction drawing.

Output Controller The Output Controller receives classifications from the AI Classification Algorithm. It matches these classifications with a user id it requests from the User ID Cache. Then it checks if the confidence level of the classification surpasses a predefined value and sends both the classification and the user id to the API, which forwards the classification to the correct user. If the confidence level is below the predefined value, a feedback request is created and sent to the API, instead of the classification. The classification is deleted.

User ID Cache The User ID Cache receives user ids with a reference to the construction drawing that the user uploaded from the Input Controller and stores them. When requested by the Output Controller, the User ID Cache responds with a user id belonging to a specific construction drawing.

Since the User ID Cache only contains user ids of construction drawings that are currently processed by the AI Classification Algorithm, it suffices that this cache is not implemented through a database or file storage, but only consists of e.g. a map living in the main memory.

E Expert Interview SAAM Scenarios

This chapter lists the scenarios that were used during the last phase of the expert interviews to perform SAAM. In total, three quality attributes were evaluated, with a total of four different scenarios per quality attribute. One of these four scenarios was predefined so that the results of the evaluation could be compared between different experts. The other three were each created by a different expert on the fly during the interview. In the following sections, the four scenarios for each quality attribute are presented.

E.1 Maintainability

Standard scenario Extend the architecture with a mechanism that restarts the AI container in case of a crash.

AI expert The cloud service provider is to be changed.

Architecture expert The system is to be extended to be able to classify other types of documents.

Product owner A module of the software has been updated and needs to be restarted for the updates to take effect. Can the system continue to operate during this time?

E.2 Compliance

Standard scenario Can an external attacker access construction drawings contained in the system?

AI expert Is a user with malicious intent able to sabotage future classifications of construction drawings?

Architecture expert Is an external software able to obtain personal data of other users?

Product owner Will critical data always be deleted after processing?

E.3 Scalability

Standard scenario A few hundred users suddenly grow to tens of thousands.

AI expert The computing power of the runtime environment is reduced (e.g. to save money).

Architecture expert Can the system handle multi-page construction drawings with thousands of pages?

Product owner Is there any way to increase the speed at which classifications are delivered?

Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted hard copies.

place, date, signature