Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterthesis

Application Integration and Team Collaboration for Process Optimization in Intralogistics

Sabri Bektas

Course of Study:	Informatik
Examiner:	Prof. Dr. Benjamin Uekermann
Supervisor:	Dipl.-Ing. Thomas Rassmann
Commenced:	January 26, 2023
Completed:	July 26, 2023

Abstract

This work focuses on the potential benefits of integrating software applications and promoting team collaboration to enhance intralogistic processes at Robert Bosch GmbH in Feuerbach. The current study is a synthesis of a comprehensive review of pertinent literature and the practical application of an architectural framework.

This research addresses the gap in literature, which lacks comprehensive guidelines for choosing software architecture approaches aligned with non-functional requirements. The objectives are to determine the necessary functionalities for an integrated solution, evaluate existing solutions and technical models, design a target architecture and provide practical guidelines for implementation.

The key research question revolves around how already existing solutions can be integrated to enhance intralogistics processes. To achieve these objectives, a comprehensive literature review is conducted to identify suitable software architecture approaches. The study evaluates various methodologies, considering non-functional requirements, culminating in an architecture ranking matrix. Additionally, interviews with project teams and analysis of user stories guide architectural decisions. The research emphasizes the microservices architecture as the selected approach for seamless integration.

The message of this research is that the microservices architecture offers a scalable, modular, and efficient solution for optimizing intralogistic processes. By integrating existing software solutions and fostering effective teamwork, organizations can achieve enhanced sustainability and customer experience. This research contributes valuable insights and practical recommendations for leveraging existing solutions to develop comprehensive and efficient intralogistics application systems. The proposed architecture serves as a template and guideline for the Virtual Development Team at Robert Bosch GmbH, facilitating the realization of an optimized and sustainable target system for intralogistics.

Contents

List of Figures

List of Tables

Acronyms

AD Architecture Decision. 66

AI Artificial intelligence. 13

AKS Azure Kubernetes Service. 76

API Application Programming Interface. 18

B2B Business-to-Business. 49

B2C Business-to-Consumer. 49

BOM Bill of Materials. 90

C&C Component-and-connector. 17

CA Camera Area. 59

CI/CD Continuous Integration/Continuous Deployment. 23

CNA Cloud-Native Architecture. 20

CQRS Command Query Responsibility Segregation. 30

DDD Domain-Driven Design. 20

DevOps Development and Operations. 24

EDA Event-Driven Architecture. 20

HTTP Hypertext Transfer Protocol. 27

HU Handling Unit. 89

IdM Identity Management. 90

ISO International Organization for Standardization. 87

JSON JavaScript Object Notation. 37

KPI Key Performance Indicators. 50

LDAP Lightweight Directory Access Protocol. 90

MDA Model-Driven Architecture. 20

OCR Optical Character Recognition. 73

OIS Optical Inspection System. 59

OS Operating System. 76

PIM Platform-Independent Model. 29

PoC Proof of Concept. 87

PSM Platform-Specific Model. 29

QA Quality Attribute. 18

QMH Queued Message Handler. 88

REST Representational State Transfer. 22

RFID Radio-Frequency Identification. 72

SaaS Software-as-a-Service. 113

SFTP Secure File Transfer Protocol. 88

SOA Service-Oriented Architecture. 20

SOAP Simple Object Access Protocol. 27

UI User Interface. 6

UML Unified Modeling Language. 15

UX User Experience. 48

VDT Virtual Development Team. 13

XML Extensible Markup Language. 37

YOLO You Only Look Once. 89

1 Introduction

1.1 Background

The fourth industrial revolution introduced innovative technologies that provide new opportunities for improving production and logistics systems. Intralogistics, which entails the management of resources such as materials and data within a warehouse or distribution centre, is essential for guaranteeing efficient supply chain operations. For these facilities to meet customer demands and maintain profitability, it is essential that they efficiently handle products. However, optimising intralogistics processes is a difficult task that requires the coordination of multiple teams and the integration of numerous technologies.

Companies can optimise their intralogistics processes by employing the appropriate software applications and nurturing effective teamwork. This study examines the potential benefits of application integration and team collaboration to enhance intralogistics processes within the context of research conducted at Robert Bosch GmbH in Feuerbach. The research focuses on integrating applications, enhancing sustainability and enhancing the consumer experience.

The automated identification of products within intralogistics processes is a challenge. In the production route of Bosch, the automation of such processes is necessary. The benefits of automation include the rapid categorization of goods in warehouses, which reduces barriers in the goods receiving and disposal processes. Another advantage is the interface with other functions that use this automation data for further goods administration. Artificial intelligence (AI) is a crucial component of these automations, especially in goods recognition, which involves the analysis of image data acquired from a camera system at a warehouse's entrance.

The Bosch Corporation has a global presence, with departments and warehouses located in various regions. To improve their processes, some of these stations have developed an intralogistics system for automated product recognition. Currently, three geographically dispersed development teams in Germany, Portugal and the Czech Republic are autonomously working on similar solutions. These solutions include functionalities such as label recognition that are comparable. The proposed concept of a Virtual Development Team (VDT) seeks to unify these disparate teams around a central hub. This collaborative approach aims to promote knowledge exchange and mutual support, resulting in solutions that are more effective, optimised and deployed more quickly. In addition, this strategy endeavours to avoid duplicative solution development.

1.2 Scope of this Research

Software architecture design is crucial to software projects. According to Chen et al. (2022), it serves as a roadmap for the development team to follow while they create a system that can achieve the project's objectives [Cha20]. By outlining the framework and building blocks of the system, software architecture design facilitates efficient collaboration amongst developers.

The success of a software project might be jeopardized by poor architectural planning. Bishung et al. (2019) indicated that it might lead to technical debt, which would increase the difficulty and expense of system maintenance [BKO+19]. System performance, scalability and dependability might be negatively impacted as a result, leading to unhappy customers and financial losses for businesses.

Software projects benefit from effective architectural designs. As a consequence, it has the potential to provide a durable, scalable and low-maintenance system that boosts customer satisfaction and organizational success. Sievi-Korte et al. (2019) argued that a flexible software architecture might also aid in meeting evolving company needs [SRB19]. It is the reason that software architecture design aids programmers in making a product that accomplishes the desired goals.

This work will investigate and propose a report for integrating and combining existing applications. The main research question is: How can already existing solutions be integrated or combined? To achieve this the study seeks to accomplish the following specific objectives:

- Determine the necessary functionalities for an integrated solution.
- Evaluate a variety of extant solutions and technical models to determine which ones best meet the specified needs.
- Create a target architecture for combining and integrating the chosen solutions.
- Provide guidelines and recommendations for implementing the proposed framework in real-world scenarios.

By addressing these objectives, the study intends to contribute to the field of software integration by providing insights and practical recommendations for leveraging existing initiatives to develop more comprehensive and efficient application systems.

During the course of my thesis, I received valuable support and guidance from my colleagues at Bosch, who possess extensive expertise in the field of software architecture design. Their mentorship proved instrumental in guiding me through the entire process. However, the specific area of the shipment validation became the focal point of my individual efforts. I crafted the architectural draft for this essential component. Subsequently, I presented and proposed the drafted architecture to the VDT.

1.3 Structure

The work is structured as follows:

Chapter 2 – Literature Review: The literature review is presented in the second chapter, wherein an analysis of contemporary software architecture approaches is conducted. Furthermore, a ranking matrix is provided to facilitate the process of architecture selection.

Chapter 3 – Integration Projects Overview: This chapter showcases the various source projects that require integration, alongside the presentation of the user interaction within the target system.

Chapter 4 – User Stories: The fourth chapter focuses on the user stories that outline the functional and non-functional requirements of the target system. These stories are prioritized based on their importance.

Chapter 5 – Architecture Modelling: This chapter provides an in-depth analysis of the architecture pertaining to the shipment validation component of the target system. It elucidates the Component View, Runtime View and Deployment View through the utilization of Unified Modeling Language (UML) notation.

Chapter 6 – Conclusion: In the last chapter, the results of the work are summarized and points of reference are presented.

2 Literature Review

Software architecture designs refer to the overall structure and organization of software systems. The purpose of software architecture is to provide a high-level overview of how the system is designed, how its components interact with each other and how it fulfils its functional and non-functional requirements. The fundamental principle of software architecture is that every software system is designed and implemented to achieve an organization's business goals.

Project teams can gain valuable insights into the benefits and drawbacks of various design methodologies by conducting a comprehensive analysis of recent research and empirical studies. In addition, a literature review can provide a greater understanding of the current state of industry research and practice.

A collection of components joined by a relationship is referred to as a structure. These structures cover the individual software parts, their interactions and the characteristics of each. Numerous structures are used in the construction of software systems and none of those structures can be regarded as "the architecture". These structures can be classified into three primary categories, each of which provides a valuable framework for understanding the architecture [Len12] [Kra15]:

- Component-and-connector (C&C) structures
- Module structures
- Allocation structure

C&C structures are concerned with the interactions between elements at runtime to accomplish the functions of a system. C&C structures describe how a system is organized as a collection of components, which represent the main units of computation. Connectors, on the other hand, act as communication channels between components. Components can take various forms, including services, peers, clients, servers and filters. Meanwhile, connectors facilitate communication between components using methods like call-return, process synchronization operators and pipes. [Tov19a] [ARI15]

Module structures are used to divide systems into implementation units, known as modules, which indicate how a system is constructed or procured. Modules have specific computational tasks and are used as a basis for work assignments for programming teams. In a module structure, the system's elements are modules of some kind, such as classes, packages, layers or other functional divisions that are units of implementation. The focus of module structures is on static considerations of the system, with less emphasis on how the software appears at runtime. Therefore, modules represent a static way of analysing the system, with each module being assigned to a functional area of responsibility. [NPHK15]

Allocation structures define how software structures are mapped to non-software structures within a system, such as a system's organization, development, testing and execution environments [Len12].

All of the software architecture design techniques evaluated in the aforementioned literature review provide full support for these three frameworks. The following literature is recommended for obtaining additional information on the three structures and their associated patterns:

- L. Bass, P. Clements and R. Kazman, “Software Architecture in Practice (3rd Edition)”, Architecture, 2012 [Len12]
- Mark Richards, “Software Architecture Patterns”, O’Reilly Media, 2015 [Ric15]

This evaluation will employ the non-functional requirements described in the subsequent section. They are also referred to as Quality Attributes (QA) and are used to evaluate system architectures during development and operation. Consequently, this evaluation employs these characteristics to guide decision-making during the design phase of software architecture.

The prevalent modern approaches to software architecture are then described and compared. Next, a subjective ranking value is assigned to the architecture design approaches, which may differ depending on the specific implementation of the approach. Finally, a ranking table will be created and used in the succeeding thesis to select an approach to software architecture.

2.1 Non-Functional Requirements

A QA is a measurable or testable aspect of a system that indicates how effectively the system meets the needs of its stakeholders beyond the system’s core function. A quality characteristic can be thought of as quantifying a product’s “utility“ along some dimension of importance to a stakeholder.

- **Availability**: A Software’s ability to be active and prepared to perform its function whenever you need it, is referred to as availability. By introducing the idea of recovery, availability expands upon the idea of reliability by stating that when a system malfunctions, it fixes itself.
- **Scalability**: Is used to describe the capacity of a system to accommodate growing numbers of users or data records without degrading performance. Horizontal and vertical scaling, as well as load balancing and clustering, should all be considered while designing a scalable architecture.
- **Deployability**: This is a property that describes how quickly and easily a piece of software can be made available for use on a wide range of hardware and operating systems. Dependencies should be kept to a minimum, configuration efforts should be kept to a minimum and deployment consistency should be maintained across environments if the architecture is to be considered deployable.
- **Integrability**: The capacity of software architecture to integrate without friction with other systems and applications is referred to as Integrability. For an architecture to be easily integrated with other systems, it must be built to accommodate a wide range of integration patterns, data formats and protocols and have well-defined Application Programming Interfaces (API).

- **Performance**: Performance, in the context of software architecture, pertains to the capability of the system to operate with effectiveness and efficiency across diverse contexts and under varying workloads. A performant architecture is one that achieves the best possible results in terms of throughput, scalability and resource usage, with the least possible latency and reaction time.
- **Testability**: The capacity to test and verify the accuracy, resilience and performance of a software design is known as its testability. A testable architecture is one that can facilitate automated testing, has comprehensive documentation and provides accurate error reporting.
- **Ease of Development**: The simplicity and efficiency with which a software architecture may be created, maintained and updated by its creators is what this quality metric is measuring. Rapid prototyping, modularity and the obvious separation of responsibilities are all features that should be supported by an easily-developable architecture.
- **Modifiability**: Software architecture modifiability is the trait of being easily and effectively updated, expanded and changed.
- **Usability**: Usability is a quality characteristic that describes how easily, successfully and satisfactorily users may use a software system to accomplish their objectives. It includes features like user happiness, learnability, usability and fault tolerance. A software system that is easy to use can increase user productivity, cut support expenses and training time and increase user acceptance. [Len12]

2.2 Software Architecture Design Approaches

In the literature review, techniques based on the most prominent search terms for software architecture are considered. The following search terms were used:
"software development" OR "software engineering"
"modern software architecture" OR "modern software designing"
"software architecture patterns" OR "software design patterns" AND "software architecture" OR "architectural design"

Google Scholar, IEEE, ACM, ResearchGate and Science Direct were utilised to locate articles and publications. If the abstracts were considered suitable, papers whose topics corresponded to the interests of the review were included.

Following the selection of relevant literature, the abstracts were combined to produce a word diagram (Figure 2.1), with the magnitude of each word corresponding to its frequency in the list of keywords.

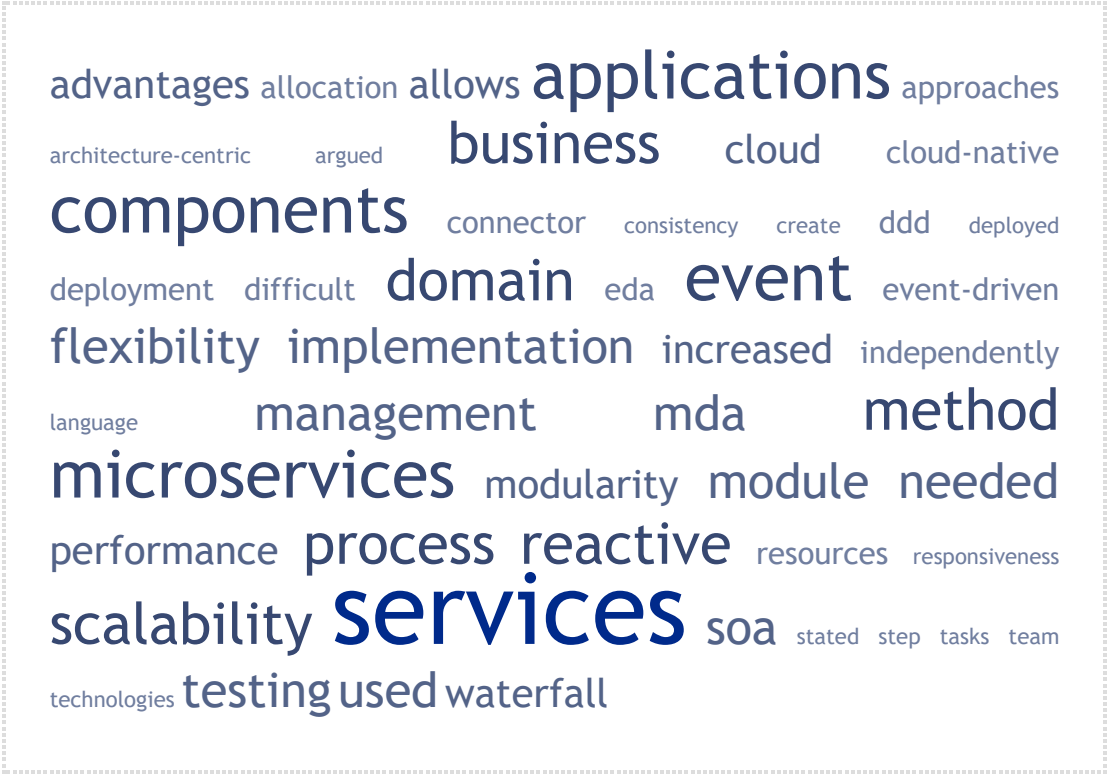


Figure 2.1: Key Word Frequency

On the basis of the frequency of the associated terms, the following strategies were investigated:

- Architecture-Centric Design
- Microservices Architecture
- Cloud-Native Architecture (CNA)
- Waterfall Software Development
- Domain-Driven Design (DDD)
- Service-Oriented Architecture (SOA)
- Event-Driven Architecture (EDA)
- Model-Driven Architecture (MDA)
- Reactive Architecture

The explanations that follow will first describe the specific approach, followed by a description of the phases involved, then list the advantages and disadvantages and lastly describe how the fundamental structures are integrated.

2.2.1 Architecture-Centric Design

Architecture-centric design is the process of designing software with an emphasis on architecture over implementation. This method, according to Siek (2011), ensures that software architecture is adhered to during implementation [Sie11].

The following are typical phases of an architecture-centric design:

1. The planning stage focuses on what the system requires and how it should be constructed.
2. The design stage plans the system's architecture [MOTG17].
3. The authentication and confirmation phase includes analysing and evaluating the architecture to ensure that it is of adequate quality and meets all system requirements [BKO+19].
4. The implementation phase entails the actual execution of the system architecture designed in previous phases.

Advantages	Disadvantages
Encourages early consideration of system architecture [FYX20]	Can lead to over-engineering or premature optimization [DL01]
Ability to develop and adapt to new circumstances by a well-architected system can keep up with the evolving needs of a business [SRB19]	May not accommodate changing requirements or evolving technologies as easily as other approaches [Dev17]
Can facilitate the creation of more robust and scalable systems [CSW21]	May require significant up-front investment in time and resources [Jai19]
May be easier to test and debug due to clearly defined and modular architecture [RJ06]	Can limit creativity and innovation in the development process [Cha20]
Can help identify potential risks and issues before they become major problems [DL01]	May not be as adaptable to certain types of projects or team structures [Cha20]

Table 2.1: Architecture-Centric Design - Advantages & Disadvantages

The Architecture-Centric Design methodology focuses on elucidating the system's components and connections. It emphasises breaking down complex systems into smaller, more manageable components, each of which performs a narrowly defined set of duties and connecting those components via suitable communication channels. The strategy recommends a modular architecture that simplifies maintenance and adaptation, as well as a clear separation of concerns [XYBZ19] [GPNV02]]. This method also takes allocation structures into consideration, as it attempts to assign system components to actual hardware resources [Bos04]. The end result of this method is an efficient system with well-defined interfaces between its components and judicious use of available resources.

2.2.2 Microservices Architecture

The term “microservices architecture“ alludes to a type of software architecture in which an application is divided into several small, independent components. Each service serves a distinct purpose and may interact with other services via APIs that are well-defined. [AC22]

In recent years, microservices architecture has gained popularity due to its capacity to facilitate rapid software development and deployment, as well as the straightforward scalability of individual services. The process of implementing a microservices architecture includes isolating the various services responsible for the application's essential operations and features. [BQT22]

Microservice architecture prioritizes modularity, concern separation and scalability. It involves deploying services independently using appropriate technology and resources. Kubernetes and Docker Swarm can be used for service management and communication between services is enabled through gateways or meshes. [AAE16]

The following are typical phases of a microservice architecture design [BQT22]:

1. Microservice architecture involves analyzing a monolithic application, identifying its components and breaking them down into smaller, independent services.
2. Clear boundaries for each microservice are defined, determining their responsibilities and communication protocols.
3. Effective communication mechanisms are designed using Representational State Transfer (REST), APIs or message queues.
4. Deployment and scalability aspects are also crucial, with a robust infrastructure designed to handle high traffic loads and scale up or down based on demand.
5. Monitoring and management of microservices are vital for smooth operation, with effective logging, error handling and performance monitoring techniques helping identify issues and optimize system performance.

Advantages	Disadvantages
Can be scaled independently, allowing for greater scalability and flexibility [WLS22]	Can add complexity, requiring careful design and management [OHM+15]
Can be designed to be resilient, with the ability to isolate and recover from failures [SAAA23]	With many services, there is a greater need for monitoring, management and coordination [MMd18]
Can be developed using different technologies, allowing for greater flexibility and innovation [WLS22]	May require additional infrastructure and tooling, which can increase costs [RMM16]
Can enable greater autonomy for development teams, allowing for faster decision-making and innovation [AAE16]	Integrating multiple microservices can be complex, requiring careful planning and management [AAE16]
	Susceptible to network delays and disruptions [AAE16]

Table 2.2: Microservices Architecture - Advantages & Disadvantages

The microservices architecture design concepts provide support for C&C, Module and Allocation Structures. With this architecture, each microservice operates independently but can exchange data with other microservices via well-defined communication channels [Len12] [Kra15]. The modularity of microservices empowers developers to create a loosely coupled architecture wherein each module assumes responsibility for a singular task and can be autonomously updated, deployed, and scaled. This attribute of microservices allows for effective allocation of real or virtual resources in response to demand.

2.2.3 Cloud-Native Architecture

The term CNA alludes to a style of construction that has been optimised for cloud-based computer systems. It is the next stage beyond "conventional" on-premise software development, in which apps are developed with cloud compatibility in mind. Using CNA, developers construct and release software applications utilising cloud-native tools such as containerization, microservices and serverless computing [Lin17]. This technique decreases development and deployment durations while enhancing scalability, robustness and portability[BHJ16].

The following are typical phases of a cloud-native design [KEP18]:

1. Design and Planning is the initial phase of cloud-native application development, gathering requirements and defining the architecture. This includes defining microservices, selecting appropriate cloud technologies, and planning the development process.
2. Development and Coding involves software development, implementing features, integrating with external services, and ensuring the code is cloud-native and scalable.
3. Containerization and Packaging involves packaging the developed software into containers, enabling consistent application running across various environments.
4. Continuous Integration/Continuous Deployment (CI/CD) practices are essential in cloud-native development, automating code changes into a shared repository and testing it.
5. Orchestration and Management is the phase where a container orchestration platform like Kubernetes is used to manage and automate container deployment, scaling, and monitoring, ensuring the application is highly available, scalable, and resilient.

Advantages	Disadvantages
Scalability: Easy to scale horizontally and vertically [AP16]	Complex architecture, may require more expertise[KEP18]
Resilience: High availability and fault tolerance [R G22b]	High cost, requires significant investment in infrastructure [Pet17]
Agility: Fast and easy to deploy changes [AP16]	Requires a mature Development and Operations (DevOps) culture and toolset [KEP18]
Portability: Can be deployed across different cloud platforms [DL22]	Complexities in managing distributed data and state [Pet17]
Efficiency: Optimized for cloud resources [Clo20]	Increased network traffic and latency [Lin17]
	Entails risk of vendor lock-in, in which applications are restricted to using a single cloud service or set of technologies [BHJ16]

Table 2.3: Cloud-Native Architecture - Advantages & Disadvantages

In a study conducted by the Cloud-Native Computing Foundation in 2020, 83% of respondents reported using Kubernetes as their container orchestration platform and 92% used containers in production. This demonstrates the increasing popularity of cloud-native techniques. IBM found that businesses utilising cloud-native technology experienced a 50% reduction in time-to-market for new applications and a 25% increase in developer productivity. This suggests that companies utilising CNA may gain substantial benefits. [Clo20]

The components of CNA are independently deployable and upgradable services. Components are small, loosely coupled services and the connectors are the APIs and message protocols used by this design [Len12]. Modules in CNA are frequently organised by enterprise-relevant functionalities or domains. During development and deployment, each module is regarded as its own, independently scalable function. Using this method, developing, testing and deploying distributed applications is effortless [DL22]. CNA uses containerization and orchestration techniques to dynamically allocate resources based on the needs of each application[Lin17]. This method is well-suited for cloud deployment because it allows for dynamic resource allocation and efficient hardware utilisation.

2.2.4 Waterfall Software Development

Since the 1970s, waterfall development methodology has been the industry standard. According to Ajmal and Ali (2016), the software is developed sequentially and each stage must be completed before proceeding [AA16].

The phases of waterfall software development are as follows:

1. During needs collection, project requirements are evaluated and documented.
2. The criteria from the previous phase are used to construct the architecture and design of the application.
3. Following software design, code is written to implement the plan.

4. During testing, the software is evaluated to determine if it conforms to preexisting standards and criteria.
5. Deployment is responsible for preparing the production environment for software operation.

Advantages	Disadvantages
Clear structure and documentation throughout the process [AA16]	Little room for flexibility or changes during the development process [Sam19]
Phases and milestones are well-defined, making it easier to manage and monitor progress [AA16]	Testing is only done at the end, which can make it difficult to identify and fix issues earlier on [Sam19]
Easier to estimate time and resources needed for each phase [Bas]	Can lead to a slower time-to-market due to the sequential nature of the process [DM18]
Clients and stakeholders have a clear understanding of what to expect and when [Bas]	Lack of collaboration between team members can lead to siloed work and less innovation [DM18]
More suitable for projects with well-understood requirements and a fixed scope [Sam19]	Can lead to a higher risk of project failure if initial requirements are incorrect or incomplete [AA16]
	Ineffective for developing complex software [AA16]

Table 2.4: Waterfall Software Development - Advantages & Disadvantages

The Waterfall model focuses on the sequential flow of activities, with each phase having specific tasks. It provides a framework for defining system requirements, identifying necessary components and their interactions. The identification of components and connectors occurs during the requirements gathering and analysis phase. The model does not explicitly prescribe a specific model structure, but system requirements are documented and analyzed during the early phases, which serve as a basis for creating high-level architectural models. The Waterfall model does not explicitly address allocation structure, but during later stages, the system is divided into smaller modules or components, which can be allocated to different development teams or individuals. [PWB09]

2.2.5 Domain-Driven Design

In 2003, Eric Evans published “Domain-Driven Design: Addressing Complexity at the Heart of Software“, the first comprehensive DDD guidebook [Eva04]. DDD is a software development technique that emphasises business domain knowledge, domain modelling and communication with stakeholders.

The development team and business stakeholders must reach a consensus on standardised terminology. Even when using specialised terminology, the language should be uncomplicated. This necessitates dividing the system into smaller, more manageable components. Each environment must have a scope-defining boundary.

The phases of DDD are as follows [Nic15] [Ver13]:

1. Identifying the core domain of the system. This includes understanding the main purpose and unique aspects of the application that differentiates it from other systems.
2. Defining bounded contexts within the system. Bounded contexts help to clearly define and isolate different subdomains or components within the larger application, enabling better organization and separation of concerns.
3. Identifying and modelling the relationships and interactions between the different bounded contexts. This helps to ensure that the components within the system work together seamlessly and efficiently.
4. The final phase is the implementation phase, where the defined bounded contexts and their relationships are implemented in the actual system. This involves coding, testing and integrating the various components to create a functioning and cohesive application.

Overall, the process of identifying the core domain, defining bounded contexts, modelling relationships and implementing the system is a crucial step in developing a successful software application.

Advantages	Disadvantages
Encourages collaboration between domain experts and developers [Ver13]	High learning curve for developers unfamiliar with DDD [Eva04]
Focuses on the core business logic and domain complexity [Eva14]	Can be more time-consuming to implement compared to other approaches [Eva14]
Helps prevent technical debt and maintainability issues [Abe07]	Requires a solid understanding of the business domains [Nic15]
Allows for greater flexibility and adaptability to changing business needs [Nic15]	May require more testing and validation to ensure business rules are accurately implemented [Abe07]
Can lead to better scalability and modularity in complex systems [HH06]	Can be over-engineered for simpler systems [HH06]

Table 2.5: Domain-Driven Design - Advantages & Disadvantages

Each bounded context encapsulates a distinct domain of the application and domain entities represent the domain objects and logic, in DDD's C&C structure. Domain services, which offer a consistent set of functions within a limited context, are an example of module structures, while aggregates, which specify transactional consistency limits for a collection of domain entities, are an example of allocation structures. [MMCF18] [Nic15]

2.2.6 Service-Oriented Architecture

When SOA is applied to the design of software architecture, applications can be viewed as a collection of services. Since these services are interoperable, Hustad & Olsen (2021) assert that they can be merged and reused to create more complex applications or to implement them into existing infrastructure [HO20]. SOA, according to Mishra and Sarkar (2022), is founded on a

design paradigm known as “service orientation“, which emphasises the production of modular, reusable and replaceable components [MS21]. Some of the SOA’s phases and approaches include the following [Erl] [MW07]:

1. The service identification phase entails identifying the architecture’s required services and analysing business requirements and procedures to determine which functionalities can be encapsulated.
2. The focus of service specification is the definition of specifications, such as inputs, outputs, behaviour, constraints and policies.
3. Service realisation entails translating the architectural design into actual implementation, choosing suitable technologies and frameworks and defining data models and communication interfaces.
4. The deployment of a service entails setting up servers or cloud environments, configuring networking and security settings and ensuring that all dependencies are implemented correctly.
5. Followed by service monitoring and management, which includes monitoring performance metrics, administering service updates, addressing scalability concerns and ensuring efficient resource allocation.

Advantages	Disadvantages
Services can be developed and deployed independently and reused across multiple applications [KK09]	The architecture is complex and requires significant planning and design [Law04]
The modular design of SOA allows for scaling of individual services as needed [MBKN09]	The added layers and communication between services can impact performance [MBKN09]
Services can be added, removed or modified without affecting the entire system [RGKS20]	Integration with non-SOA systems can be challenging [NIG+20]
Services can be deployed on different platforms and locations, increasing availability [NIG+20]	Effective governance is needed to ensure consistency and adherence to standards [KK09]

Table 2.6: Service-Oriented Architecture - Advantages & Disadvantages

Connectors in SOA are the communication protocols and mechanisms that permit interaction between services. Using standardised protocols like Simple Object Access Protocol (SOAP) and REST over Hypertext Transfer Protocol (HTTP), they facilitate data exchange and collaboration. Components are independent services, whereas connectors facilitate their interaction. A module is a logical unit that incorporates a specific set of functionalities or business processes in the context of SOA. A module is typically a cohesive and independent element of code that is responsible for implementing a particular aspect of the functionality of a service. On various physical or virtual machines, containers or cloud platforms, services can be deployed. The allocation structure takes scalability, performance, defect tolerance and resource utilisation into account. [NIG+20] [PH07]

2.2.7 Event-Driven Architecture

EDA software design encourages the production, monitoring and processing of system-level events. Events are occurrences that have a significant effect on the system or its procedures. EDA decoupling enables events to communicate across components that may be geographically distant. This strategy increases the adaptability and scalability of the system. [IE06][BD14]

Typically, event detection and analysis consist of four steps: [IE06].

1. The event detection stage focuses on identifying system-relevant events as a first step.
2. In the second phase, event routing is taking place where message intermediaries and publish-subscribe systems are employed.
3. The third stage involves gaining insight from the event and selecting future actions.
4. In the persistence phase, events are stored for auditing or additional analysis.

Advantages	Disadvantages
Highly scalable and flexible [CB12]	Complex to design and implement [DW05]
Supports real-time processing and asynchronous communication [DW05]	Requires extensive event logging and monitoring [CB12]
Decouples components and allows for independent development and deployment [CB12]	Difficult to debug and troubleshoot [IE06]
Enables event-driven workflows and business logic [Woo21]	Inconsistent performance due to variable event processing times [IE06]
Can handle large volumes of data and events [CB12]	Difficult to maintain consistency and transactional integrity [IE06]
Promotes modularity and reuse of components [DW05]	Requires specialized tools and expertise [DW05]
Supports distributed and cloud-based systems [Woo21]	Can result in a high number of redundant events [DW05]
Can improve system responsiveness and agility [Woo21]	Can result in increased network traffic and latency [IE06]

Table 2.7: Event-Driven Architecture - Advantages & Disadvantages

Components in an EDA are typically categorised as either event producers or event consumers and are connected via an event bus or message broker. Instead of considering the typical control flow between components, the event-driven method focuses on the flow of events and the responses to those events. Each module in an EDA system may contain multiple event producers and consumers, with business capabilities serving as the organising principle. Allocation structure in EDA refers to the distribution and assignment of event producers, event consumers and event processing components within the architecture. [IE06] [Woo21]

2.2.8 Model-Driven Architecture

MDA is a software development methodology that emphasises the use of models as the primary artefacts for designing, defining and generating software systems. It provides a framework for distinguishing the functionality of a system from its implementation. [09]

In MDA, the development process revolves around models, which are abstraction-level-specific representations of the system. These models encapsulate the essential structure, behaviour and functionality of the system. They serve as a common language for stakeholders such as business analysts, developers and architects. [MAC+04]

MDA has several distinct phases, including the following [09] [GYE22]:

1. During the requirements collection phase, every need that the system must fulfil is meticulously mapped out.
2. During the platform-independent modelling phase, a domain-specific modelling language is used to convert the system requirements into Platform-Independent Model (PIM).
3. Automated model transformations are used to convert PIMs into Platform-Specific Model (PSM). These PSMs are then utilised in later modelling phases.
4. During the phase labelled “code generation“ the PSMs are converted into machine-readable code.

Advantages	Disadvantages
Automates repetitive tasks [Bro04]	Requires initial investment in MDA tools [Joh01]
Improves development productivity [GYE22]	Learning curve for MDA tools can be steep [GYE22]
Provides consistency across the system [Sol00]	Can be difficult to integrate with legacy systems [AJW03]
Promotes code reuse and maintainability [Bro04]	May be less flexible than traditional development [Joh01]
Simplifies maintenance and updates [MAC+04]	May require additional effort for customization [09]
Enables faster time-to-market [GYE22]	May not be suitable for all project types [Bro04]

Table 2.8: Model-Driven Architecture - Advantages & Disadvantages

MDA prioritises model structure over defining C&C structures. C&C can be represented as encapsulated units of functionality with well-defined interfaces, with connectors serving as means of communication and interaction. Modelling languages, such as UML, offer constructs for modelling C&C, which depict architectural structure and relationships. MDA differentiates PIM and PSM, enabling hierarchical relationships and transformations. MDA’s allocation structure incorporates the distribution and allocation of system components and resources across the target architecture. It provides a framework for capturing allocation influencing system requirements and design decisions. This data can direct code generation or deployment processes to reflect the allocation of system components and resources. [09] [AJW03]

2.2.9 Reactive Architecture

In modern software systems, Reactive Architecture is an architectural approach that prioritises responsiveness, scalability and resilience. It emphasises managing and responding to large numbers of concurrent and asynchronous events while maintaining a consistent user experience. Responsiveness, message-driven and event-based communication, elasticity and scalability, resilience, reactive streams and backpressure, event sourcing and Command Query Responsibility Segregation (CQRS) are key characteristics of reactive architecture. [DSM+17]

Reactive architecture prioritises delivering a high-quality user experience while guaranteeing the stability and responsiveness of contemporary software systems.

The following are typical phases of the reactive architecture approach [AA15] [Sob10] [Tov19b]:

1. Collecting and analysing the application's functional and non-functional requirements, performance expectations and scalability needs.
2. Using modelling techniques such as event storming or domain-driven design for identification of the system's key components, data flows, interactions and relationships.
3. Reactive architecture significantly depends on event-driven design principles and emphasises the utilisation of reactive components and patterns.
4. Data management is essential for effective data management and techniques such as distributed databases, cache and stream processing are utilised.
5. Introduce mechanisms like fault-tolerant clustering, replication and self-healing capabilities, reactive systems should be able to gracefully manage failures and maintain high availability.
6. Deployment and scaling are crucial stages in deploying the application to the desired environment, including configuring infrastructure, establishing deployment pipelines and scaling the system horizontally or vertically to accommodate increasing load or demand.
7. Continuous monitoring of a system's health and efficacy requires monitoring and optimisation. Analysing collected data aids in identifying bottlenecks, optimising performance and enhancing overall productivity.

Advantages	Disadvantages
Highly responsive and resilient [ELV+12]	Requires significant expertise to design and implement [DSM+17]
Scalable and adaptable to changing requirements [Tov19b]	Increased communication complexity and potential for errors [Tov19b]
Promotes loose coupling and better separation of concerns [SAM+19]	Increased network overhead [AA15]
Efficient use of resources [AA15]	Debugging and tracing can be challenging in distributed systems [Tov19b]
Supports event-driven and real-time systems [SAM+19]	Not suitable for all types of applications or systems [SAM+19]
Promotes modularity and reusability [AA15]	Difficulty in testing and validating the system as a whole [AA15]

Table 2.9: Reactive Architecture - Advantages & Disadvantages

Connectors are the means through which two or more components may exchange data with one another in Reactive Architecture, with components themselves being tiny, self-contained and autonomous entities. Connectors in Reactive Architecture are often event-driven, allowing for non-blocking, asynchronous communication between components. Module organization in Reactive Architecture often borrows ideas from microservices architecture, which breaks down large systems into smaller, more manageable components. Services in a Reactive Architecture are often implemented using a distributed allocation structure that optimizes performance and availability and scales quickly to meet fluctuating demand. [DSM+17] [Tov19b] [AA15]

2.3 Software Architecture Design Approaches Comparison

Overall advantages, disadvantages and key features of all approaches are presented in following Table 2.10.

Approach	Advantages	Disadvantages	Key Features
Architecture Centric Design	Modularity, flexibility, reusability, interoperability, better system understanding, system-wide consistency	Difficult to accommodate late changes, requires significant upfront design, may be over-engineered for smaller projects	Components, connectors, interfaces, design patterns
Microservices Architecture	Scalability, agility, resilience, independent deployment, reusability, fault isolation and resilience	Complexity of testing, operational overhead, latency and overhead of network calls	Services, API based communication, load balancer, small and decoupled services, containerization

Continued on next page

Table 2.10 – *Continued from previous page*

Approach	Advantages	Disadvantages	Key Features
Cloud-Native Architecture	Scalability, high availability, fault tolerance, portability, cost efficiency (pay-as-you-go)	Complexity of orchestration, requires learning new tools, security concerns, vendor lock-in	Leveraging cloud services and platforms orchestration, service discovery, immutable infrastructure
Waterfall Software Development	Well-defined phases, predictable outcomes, clear documentation, rigorous control, suitable for small projects	Limited flexibility, difficulty adapting to change, poor collaboration	Sequential development, comprehensive documentation
Domain-Driven Architecture	Modular design, improved communication, agile development, flexibility	May require significant refactoring of existing code, complexity of implementation, Learning curve for DDD concepts	Bounded contexts, ubiquitous language, aggregates entities
Service Oriented Architecture	Interoperability, scalability, reusability, maintainability, flexibility	Complexity of orchestration, service bloat, coupling of services	Loose coupling, standardized communication, focus on service and interfaces
Event-Driven Architecture	Scalability, loose coupling, extensibility, responsiveness, traceability, fast responses	Debugging can be challenging, eventual consistency, complexity of data flow	Event bus, event sourcing, event-driven messaging (produces and consumer)
Model Driven Architecture	Reusability, consistency, automation, improved communication, adaptability to changes	Requires significant upfront design, learning curve, may be over-engineered for smaller projects	Models, transformations, metamodels, platform independent modelling
Reactive Architecture	Scalability, responsiveness, fault tolerance, high performance	Complex to implement, requires skilled developers, debugging can be challenging	Actors, streams, message driven communication, CQRS, asynchronous processing

Table 2.10: Architectures - Advantages & Disadvantages & Key Features

2.4 Software Architecture Design Approaches Ranking based on Non-Functional Requirements

One of the major characteristics of deciding on a software architecture approach is the non-functional requirement evaluation.

As discussed in Section 2.1 the nine non-functional requirements are evaluated regarding to the architecture approaches. These variables aid in assessing and contrasting the efficacy of various architectural strategies in achieving the required levels of these traits.

In this section, an assessment is conducted, followed by the construction of a matrix (Table 2.11) wherein evaluation scores, ranging from 1 to 5, are assigned to each architecture method. A score of 1 corresponds to a poor grade, while a score of 5 denotes an excellent grade.

2.4.1 Non-Functional Requirements Evaluation for Architecture Centric Design

Through the use of redundant components and systems, fault tolerance techniques and load balancing techniques, architecture-centric design ensures availability. These measures guarantee continuous operation in the event of a failure and evenly distribute the workload across multiple resources. However, improper maintenance or testing of redundancy mechanisms can result in system failure and load-balancing techniques may not always prevent component overburden in the presence of unpredictable or unequally distributed workloads. [Dev17]
Availability Grade: 4

Scalability is achieved via modular components and adaptable infrastructure. By decomposing a system into smaller, independent components, this method enables systems to accommodate increased workloads and adapt to shifting requirements. A well-designed architecture has a flexible infrastructure that permits the addition or withdrawal of resources without affecting the system as a whole. Due to significant modifications and potential outages or performance disruptions, it may be difficult to scale a monolithic architecture where components are tightly coupled. [Dev17]
Scalability Grade: 4

The procedures ensuring availability also guarantee continuous operation and eliminate singular points of failure. On the other hand, they may encounter obstacles when systems rely on volatile or unreliable external resources, resulting in deployment and maintenance difficulties. [DL01]
Deployability Grade: 4

Standardised interfaces and protocols are essential for architecture-centric design's integrability. Components can interact based on agreed-upon protocols by designating clear communication interfaces, such as APIs or service contracts. However, interface incompatibilities and conflicts can impede seamless integration. Custom adapters or middleware may be required to ensure interoperability and compatibility between components by bridging the communication gap between them. [GPNV02]
Integrability Grade: 4

Beginning the design process with performance in mind simplifies the achievement of performance objectives. By including performance as a primary requirement, design decisions and compromises can be made with performance in mind. Architecture-centric design often leverages well-known architectural patterns, best practices and technologies such as caching, load balancing and asynchronous processing can facilitate performance optimisation. [Bos04]
Performance Grade: 5

Modular architecture promotes testability by decomposing the system into independent, loosely coupled components that can be isolated and tested separately. Well-defined interfaces enhance testability by facilitating the replacement of external dependencies with controlled test data.

Nevertheless, this method may introduce additional complexities, such as dependency injection, test-friendly interfaces and additional abstraction layers. Managing this complexity can be difficult, particularly when it compromises the architecture's clarity and simplicity. Test maintenance is essential for preserving the efficacy and currency of the architecture over time. [RJ06]
Testability Grade: 3

By establishing a clear architecture and supplying well-defined guidelines, developers are better able to comprehend architectural principles and design constraints. Clear guidelines on component responsibilities, communication patterns and data flows allow developers to make well-informed design decisions and create software that conforms to the architecture. [RJ06]
Ease of Development Grade: 5

Modifiability in architecture-centric design involves using modularity, component-based design, separating concerns, minimizing coupling, providing open extension points, abstraction, encapsulation, leveraging design patterns, continuous refactoring and documentation sharing. These key points enable flexible, adaptable and easily modifiable architecture. Balancing complexity without hindering understandability or performance can be challenging and design decisions to enhance modifiability may impact system performance, as loose coupling or abstraction layers can add overhead to the system. [GPNV02]
Modifiability Grade: 3

Architecture-centric design prioritises user requirements, incorporates principles and provides user-friendly interfaces, efficient workflows, explicit documentation, user testing and feedback. Understanding requirements, conducting research, designing intuitive interfaces, simplifying complex processes, providing documentation, tutorials and responsive support, as well as perpetually iterating based on user feedback to improve system usability and user experience, are essential elements. [Bos04]
Usability Grade: 5

2.4.2 Non-Functional Requirements Evaluation for Microservices Architecture

The approach microservices architecture incorporates availability through redundancy, load balancing, failover, circuit breaker and monitoring and alerting. Even in the event of faults or failures, these strategies ensure that the system remains operational and accessible to its users. [WLS22]
Availability Grade: 5

The best possible grade for scalability was awarded to the microservices architecture since it allows for highly scalable and flexible service deployments through its loosely coupled, independent and exchangeable components. [Dav21]
Scalability Grade: 5

The continuous and simple deployment of individual services earned a good evaluation in the deployability category. One counterargument to the evaluation of deployability is that managing and deploying a large number of individual services can become complex and time-consuming, potentially leading to deployment errors or delays. [BQT22]
Deployability Grade: 4

The loose coupling and straightforward API and message-based communication fostered by microservices architecture earned it a good grade in the integrability category However, a counterargument to the microservices architecture's evaluation in integrability is that coordinating and maintaining the interactions between numerous services can become challenging, potentially resulting in compatibility issues or communication failures. [AC22]
Integrability Grade: 4

Microservices architecture permits great performance via the usage of lightweight and focused services. Since each microservice could be scaled independently and therefore adapt to the varying workload, it can increase its performance by acquiring computational power on heavy-duty tasks. [AC22]
Performance Grade: 5

With this approach, it is possible to test each service separately and additionally encourages automated testing. However, testing each service separately does not guarantee that the overall system will perform well when the services are interconnected and automated testing may not capture all possible issues that can arise in complex systems. [WLS22]
Testability Grade: 4

Since it encourages the creation and deployment of services independently, microservices architecture received a high grade for development simplicity. However, this independence can lead to difficulties in coordinating and managing the interactions between different services, resulting in added complexity and potential integration issues. [WLS22]
Development Simplicity Grade: 4

It encourages modularity and permits simple adjustment of specific services without impacting the overall system. However, this modularity can also lead to a higher maintenance burden as changes made to one service may require adjustments in other services that interact with it, potentially increasing complexity and integration challenges. [Dav21]
Modifiability Grade: 4

The design can enhance the user experience by allowing for greater flexibility, customisation and personalisation of services. A microservices architecture can facilitate the application of user-centric design principles, such as user personas, user stories and user feedback, to guide the development of individual microservices. [BQT22]
Usability Grade: 5

2.4.3 Non-Functional Requirements Evaluation for Cloud-Native Architecture

Availability is crucial for creating durable, controllable and observable loosely connected systems. However, there are instances where availability can be compromised, such as when a cloud provider experiences a major outage or system failure, causing widespread disruptions for users and businesses. [Lin17]
Availability Grade: 4

Scalability is achieved through containerization and orchestration technologies, which allow for automatic scaling of resources based on demand. This feature enables applications to efficiently handle varying levels of traffic and adapt to changing business needs. [R G22a]
Scalability Grade: 5

Depolyability is crucial for easy application deployment across different environments. Containerization enables easy packaging and deployment of applications, but it can be counterproductive in cases where external dependencies or resources are not easily containerized. Complex configuration changes and setup may introduce delays and errors during deployment. [Lin17]
Deployability Grade: 4

Promoting the use of microservices and APIs to enable independent development and communication between components. However, a lack of proper documentation or standardization in the development process can lead to inconsistencies in APIs and communication protocols, leading to errors and failures during the deployment process. [DL22]
Integrability Grade: 4

Advocating the utilization of automation and monitoring tools to enhance performance and address bottlenecks is paramount. Nevertheless, it is important to note that a cloud-native application that lacks performance optimization may still encounter challenges related to scalability and overall performance. [R G22c]
Performance Grade: 4

Facilitating seamless and effective testing is achieved through the implementation of automated testing, containerization orchestration technologies, and modular components. However, it should be noted that complex architectures often lack distinct boundaries, which poses challenges in isolating and testing individual components. CNA principles may not include proper testability practices, such as modular components or automated testing. This can cause inefficient testing processes and negatively impact the application's overall quality. [R G22b]
Testability Grade: 3

It offers ease of development through containerization and orchestration technologies, simplifying deployment and testing for developers. However, dealing with complex microservices can be challenging due to their interconnected nature. This complexity can result in longer development cycles, increased debugging efforts and increased risk of introducing bugs or errors during deployment. [Tel22]
Ease of Development Grade: 3

Modifiability is achieved through containerization and orchestration technologies, such as Docker and Kubernetes, which enable the deployment and management of microservices independently. However, achieving modifiability through these technologies does not guarantee a seamless deployment process, as complex interdependencies between microservices or multiple updates simultaneously can still be challenging and prone to errors. [Lin17]
Modifiability Grade: 4

Usability is another aspect, incorporating intuitive user interfaces, clear documentation and robust monitoring and troubleshooting tools. However, even with these measures in place, there may still be instances where the usability of a CNA falls short. [DL22]
Usability Grade: 4

2.4.4 Non-Functional Requirements Evaluation for Waterfall Software Development

To prioritize availability-related decisions during the design phase, it is crucial to consider redundancy, fault tolerance, scalability, and backup and recovery mechanisms. Implementing failover strategies, load balancing techniques, and appropriate hardware or software architectures are key to maximizing system availability. However, the rigid and sequential nature of the cascade methodology can hinder the timely identification of availability issues, which may only surface during the testing phase. Furthermore, the waterfall methodology lacks a continuous feedback cycle between development and operations teams, resulting in delayed feedback on observed availability issues in production environments. [AA16]
Availability Grade: 3

Prioritising scalability during the requirements gathering phase by working closely with stakeholders to identify potential requirements such as user traffic, data volume and performance requirements. Considering load balancing, caching, vertical and horizontal scaling and database partitioning as scalability patterns. Changes in scalability requirements or the emergence of new factors can make it difficult to alter the design and implementation phases to accommodate these modifications. [AA16]
Scalability Grade: 4

Defining deployment requirements during the requirements gathering phase, including target environments, hardware specifications, operating systems and constraints. Performing pre-deployment testing in a staging environment that closely resembles the production environment in order to verify the process and validate the functionality of the software. The waterfall methodology has limited iterative feedback and adaptability, making it challenging to incorporate deployment related feedback-based changes. [DM18]
Deployability Grade: 3

Specify inputs, outputs, communication protocols and data formats with precision to ensure compatibility and interoperability. Stable, standard protocols and data formats, such as REST API or Extensible Markup Language (XML)/JavaScript Object Notation (JSON), facilitate the seamless integration of components or systems. As each phase is concluded before moving on to the next, iterative integration can be difficult, causing delays in identifying and resolving issues and making it more difficult to achieve a seamless integration. [Sam19]
Integrability Grade: 3

Defining performance requirements, such as response time, throughput and resource utilisation and take into account variables such as data structures, algorithms, caching mechanisms and system scaling. Reducing computation, reducing resource contention and using appropriate data structures while optimising code for efficiency. The waterfall methodology follows a sequential, linear approach, which limits performance enhancement flexibility. Changes to the design, architecture or implementation may necessitate extensive revision or delay the completion of the project. [Sam19]
Performance Grade: 3

Testability requires specific, measurable requirements for the efficient development and evaluation of test cases. Prioritising the allocation of time for test planning and design, determining testing scope, defining objectives and developing a comprehensive test plan. Waterfall methodology, which

is frequently implemented at the conclusion of the development process, can result in a lack of resources and make it difficult to resolve defects and issues discovered during testing. [Bas]
Testability Grade: 3

It is essential to have well-documented, clear and unambiguous requirements for a software development process to run smoothly. This makes it easier for developers to comprehend and implement the software. It is essential to invest time and effort in the design and planning phases, as they guide the development process and help developers comprehend system architecture and component interactions. The sequential, linear nature of the waterfall methodology makes it difficult to incorporate changes or modifications. In addition, delayed validation can lead to the identification of issues after substantial development effort has been invested, resulting in rework, additional effort and delays in achieving the desired development simplicity. [Bas]
Ease of Development Grade: 2

Inflexible modifications can hinder the software's modifiability, as they may necessitate extensive revision or disrupt project timelines. Waterfall methodology delays change identification until later phases, limiting consideration of changes during testing or deployment. Additionally, a lack of iterations hinders the software's modifiability. [PWB09]
Modifiability Grade: 1

Involving end users and stakeholders in the initial phases of the design process in order to comprehend their requirements, preferences and workflows. Ensure that requirements are distinct and that functionality, interfaces and interactions are defined. A sequential, linear methodology, waterfall may restrict iterative design and refinement of usability aspects. [DM18]
Usability Grade: 4

2.4.5 Non-Functional Requirements Evaluation for Domain-Driven Design

DDD emphasises ensuring accessibility via techniques and principles, such as bounded contexts and asynchronous communication patterns. Bounded contexts define distinct boundaries and responsibilities within a system. This entails determining which portions of the system must be available at all times and which can tolerate some downtime. DDD also encourages asynchronous communication, which enables components to exchange data without impeding or waiting for responses. Nevertheless, separating a system into bounded contexts may result in data inconsistency and synchronisation problems, as changes made in one context may not propagate instantaneously to other contexts. [Nic15]
Availability Grade: 4

Using bounded contexts DDD accomplishes scalability. These contexts can scale independently to accommodate increased demand and traffic without affecting other contexts. Nonetheless, scaling one bounded context may impact other contexts due to conflicts and discrepancies in shared data, posing coordination challenges and the possibility of inconsistencies in the system's overall state. [Eva14]
Scalability Grade: 4

DDD achieves deployability by its manageable bounded contexts, each focusing on a specific subdomain. However, deployability can be challenging if interdependencies exist between these contexts. For instance, if one context requires data from another, changes in the dependent context can affect the entire system's deployability. [Ver13]
Deployability Grade: 3

Utilising integration patterns and techniques to manage interdependencies between bounded contexts effectively ensures integrability. Among these are mechanisms for defining interfaces, contracts, messaging, EDA and data synchronisation. On the other hand, the lack of distinct boundaries and communication protocols can result in conflicts, inconsistencies and system instability. Inadequate integration patterns and methods can also lead to performance and scalability issues. [Nic15]
Integrability Grade: 3

Analyzing the domain model helps identify potential bottlenecks and high computational complexity areas, allowing developers to optimize them. Domain-specific optimizations, like caching or precomputing data, minimize repetitive calculations and reduce response times. However, these optimizations may introduce additional complexity and maintenance overhead, potentially outweighing potential performance gains in certain scenarios. [Ver13]
Performance Grade: 3

Separating concerns enables the isolation of domain logic from infrastructural dependencies, allowing unit tests to concentrate on the model's behaviour without requiring external systems or databases. This method also permits the substitution or simulation of dependencies during testing. But separating domain logic from infrastructure may increase development and maintenance complexity and overhead. [Eva14]
Testability Grade: 3

Ease of development is enabled by focusing on a clear, well-defined domain model, allowing developers to focus on core business logic without infrastructure concerns. Encapsulating domain logic within entities, value objects and aggregates improves code understanding and modification, leading to faster development cycles. Ubiquitous language and domain experts' involvement ensure accurate code reflects business requirements. This approach can make the codebase more complex and harder to maintain, especially with frequent changes or updates. Additionally, relying heavily on domain experts may limit system flexibility and adaptability in response to changing business needs. [Eva04]
Ease of Development Grade: 2

The modular approach allows developers to modify and evolve a system's domain logic without disrupting other parts. Domain events and event sourcing enhance modifiability by capturing and storing change history. However, this approach may not be suitable for complex interdependencies, as changes may affect other domains, causing unexpected behaviour and difficult debugging. Additionally, domain events and event sourcing introduce complexity and overhead, making the system harder to understand and maintain for developers unfamiliar with these concepts. [Abe07]
Modifiability Grade: 3

Each bounded context focuses on a specific aspect of the system's functionality and has its own well-defined language and set of models. This allows developers to have a clear understanding of the domain they are working on and enables them to build intuitive and user-friendly interfaces that align with the users' mental models. [Abe07]
Usability Grade: 4

2.4.6 Non-Functional Requirements Evaluation for Service-Oriented Architecture

SOA assures availability through mechanisms and best practices, including redundancy and fault tolerance techniques. Multiple instances of services operate simultaneously, ensuring system continuity even if one fails. Load balancing ensures optimal performance and availability by distributing incoming requests equitably. [Law04]
Availability Grade: 5

The architecture uses load balancing and caching mechanisms to improve scalability by evenly distributing requests and reducing service burden. Microservices architecture breaks down complex applications into independent services, allowing flexibility and agility in scaling based on demand. SOA, on the other hand, emphasizes loose coupling and interoperability between services. [KK09]
Scalability Grade: 4

SOA encourages loose coupling between services, enabling each service to be independently developed and deployed without affecting the overall system. Services are accessed via standardised protocols such as HTTP, SOAP or REST, allowing for simple platform and technology integration. Service registries are also utilised by SOA to facilitate discovery and deployment. However, SOA can be undermined when services have significant dependencies on each other's data structures or interfaces, causing changes to a service's data structure or interface to have an effect on all dependent services and thus contradicting the concept of independent development and deployment. [Erl] [MW07]
Deployability Grade: 4
Integrability Grade: 5

Performance is achieved through optimizing communication between services, minimizing latency and overhead, using efficient protocols, lightweight data formats and reducing network round trips. Caching mechanisms store frequently accessed data and load balancing and horizontal scaling distribute workload across multiple instances, improving overall system performance. Despite efforts to minimize latency and overhead in message passing, certain situations may not be feasible. For instance, in highly distributed systems with geographically dispersed nodes, physical distance can introduce network delays and increase latency. [Law04]
Performance Grade: 4

Designing services with distinct interfaces and boundaries facilitates the isolating and testing of individual components. Standard protocols such as SOAP and REST make testing easier. Injection of dependencies and inversion of control help decouple services, making them easier to evaluate. However, the use of non-standard or custom protocols can impede the testing process, as it may necessitate additional effort to develop and maintain specific testing frameworks. [Law04]
Testability Grade: 4

SOA development can be facilitated by strategies such as modular design, which permits services to be developed and deployed independently and standard protocols, which provide a common language for communication and interoperability among services. [PH07]
Ease of Development Grade: 5

In SOA, modifiability is accomplished via service contracts, which define the interface and behaviour of a service. These contracts permit autonomous evolution that does not influence other dependencies. Loose coupling and abstraction reduce the influence of modifications on other components. In contrast, when there is a tightly coupled service dependency, minor changes

can cascade across multiple dependent services, making the process complex and prone to error. [NIG+20]
Modifiability Grade: 4

It focuses on simplicity and ease of use through clear interfaces, well-documented APIs and comprehensive user documentation. Interoperable services enable seamless integration and interaction with other services. Standard protocols and formats enhance usability by facilitating communication and data exchange between architecture components. [MBKN09]
Usability Grade: 5

2.4.7 Non-Functional Requirements Evaluation for Event-Driven Architecture

Through distributed systems and fault-tolerant mechanisms, availability is ensured. Implementing redundancy permits multiple event processors to be deployed across multiple nodes, thereby minimising downtime and guaranteeing uninterrupted service. Event-driven systems additionally employ message queues or archives to store and buffer incoming events, ensuring scalability and resilience against traffic spikes or event volume spikes. [CB12]
Availability Grade: 5

EDA achieves scalability through techniques like distributed systems, message queues and event logs. These systems distribute workload across multiple processing units, allowing them to handle higher event volumes. However, a counterexample to scalability is when a single node becomes overloaded with events, causing a bottleneck. [DW05]
Scalability Grade: 4

EDA accomplishes deployability via containerization and microservices, separating the system into independent units for simple deployment and scalability. This modular design increases the system's flexibility and adaptability to change. Event-driven system deployment necessitates managing multiple components, ensuring proper configuration and managing dependencies, posing complexities and increasing the likelihood of errors. [DW05]
Deployability Grade: 3

It accomplishes integration via event-driven messaging systems and application programming interfaces. APIs provide a standard interface for integrating external systems or services, whereas these systems enable seamless communication between components. Yet, outages or unavailable messaging systems can disrupt the flow of information, resulting in data inconsistencies and system failures. [IE06]
Integrability Grade: 4

EDA accomplishes performance via asynchronous messaging, parallel event processing and component decoupling for enhanced responsiveness. However, it may not be the best option in situations such as financial transactions where the precise order of events is essential. The asynchronous nature of messaging can lead to problems with consistent event order and debugging and troubleshooting can be complicated by the management of multiple independent components. [BD14]
Performance Grade: 3

Testability in EDA entails simulating external system or dependency behaviour with mock or stub components during testing. This permits developers to test individual components in a controlled environment. Frameworks for automated testing validate the response and behaviour of a system while monitoring and logging tools to trace event flow. However, there may be circumstances in which the behaviour of an external system or dependency cannot be readily replicated or controlled in a testing environment, such as when relying on real-time data from an external API. Event-driven systems require thorough testing and debugging to ensure correct behaviour and interactions across components and services, as events may be distributed and asynchronous. [Woo21]
Testability Grade: 3

By decoupling components and using event-driven messaging, EDA simplifies development. This approach allows developers to concentrate on individual components or services without having to consider their interactions. This modular approach promotes code reuse and scalability by allowing components to be replaced or added as necessary. However, when there are too many events and handlers, it becomes more difficult to monitor dependencies and interactions, which can lead to bugs and inconsistencies. [IE06]
Ease of Development Grade: 4

It decouples events and handlers, allowing one component to be modified independently of the others. Flexibility and adaptability in a system are essential but can be difficult to maintain efficiently when synchronous communication is extensively utilised. Tightly coupled architectures can cause ripple effects throughout the entire system and adding or removing new events and handlers may necessitate substantial modifications to existing components, thereby limiting flexibility and adaptability. [IE06]
Modifiability Grade: 3

Event processors are crucial for determining the behaviour and responsiveness of a system. Designers should create handlers that are both intuitive and efficient to facilitate ease of use and navigation for the end users. Clear documentation and guidelines enhance efficacy by assisting users in comprehending how to interact with the system effectively. [Woo21]
Usability Grade: 5

2.4.8 Non-Functional Requirements Evaluation for Model-Driven Architecture

The modelling phase may incorporate redundancy considerations, such as duplicating critical components or services, instituting failover mechanisms or employing load-balancing techniques. Availability is a complex characteristic that requires fault tolerance, redundancy, failover mechanisms and recovery strategies. It can be difficult to precisely and effectively model these aspects using available abstractions. In some instances, modelling languages and tools may lack the necessary expressiveness to convey the complexities of availability. [Bro04]
Availability Grade: 3

MDA abstracts and separates scalability concerns such as load balancing and partitioning by utilising models to represent various system components. This methodology emphasises automated code generation based on models, allowing for the construction and deployment of scalable components or services using code generators and deployment tools. Modelling languages and tools for capturing scalability requirements and mechanisms have limitations. Traditional models emphasise

design-time modelling and code generation, which may not support real-time adaptation and therefore limit dynamic scale systems in response to changing conditions. [Sol00]
Scalability Grade: 3

Well-defined and automated model transformations and code generation procedures are essential for transforming models into deployable artefacts for target deployment platforms. The tools and platforms for creating, transforming and deploying models may have their own requirements and dependencies, which can create complications when moving or deploying models across various environments or updating to newer versions. [09]
Deployability Grade: 4

The standardization of modelling languages, notations, and frameworks provides significant benefits to MDA. It ensures interoperability among various tools, platforms, and systems, promoting seamless communication and integration between them. MDA can aid in the generation of API definitions and the corresponding code or configuration files. Modelling languages again may have limitations when it comes to expressing integration-related concepts. [MAC+04]
Integrability Grade: 4

Using techniques such as queuing models, state models and performance annotations can enhance performance. Runtime performance monitoring and optimisation mechanisms, collecting data to identify bottlenecks, detect anomalies and initiate optimisations or auto-scaling actions are possible. However, model transformations can introduce additional latency, such as processing time, memory consumption or code complexity. In addition, modelling languages can limit the ability to encapsulate performance requirements in their entirety. [AJW03]
Performance Grade: 3

Testability considerations are incorporated into the modelling process by designing testable models, such as by separating concerns, modularizing components and defining explicit interfaces. However, it can be difficult to generate tests from complex models, particularly when they are large or intricate. Developing automated processes that accurately reflect the system's behaviour and encompass all relevant scenarios can have an effect on the system's overall testability. [Joh01]
Testability Grade: 4

Utilize intuitive modelling languages and low learning curves to simplify system requirements, behaviour and structure. MDA introduces new tools and concepts, which can be steep for developers unfamiliar with them. Complex tool interfaces and languages can further complicate learning and development. Debugging and troubleshooting generated code or artefacts can be more complex in MDA compared to traditional coding approaches, affecting diagnosing and resolving issues during development. [GYE22]
Ease of Development Grade: 3

Designing models that encourage abstraction and concern separation, delineating the responsibilities and interactions of various system components. This permits autonomous modification without affecting the system as a whole, thereby enhancing its modifiability. However, modelling languages used in MDA may have limitations. Additionally, compatibility and interoperability between tools and platforms can have a negative effect on the modifiability of generated artefacts, making it difficult to employ newer versions. [Bro04]
Modifiability Grade: 3

Using user-friendly and intuitive modelling languages and tools with plain syntax and an intuitive interface. Focusing on user-centred design principles, taking into account user preferences, requirements and abilities. Creating interactive tools that provide immediate feedback and identify potential errors or inconsistencies. Modelling tools in MDA may be limited by a lack of features, performance issues or compatibility issues. [AJW03]
Usability Grade: 5

2.4.9 Non-Functional Requirements Evaluation for Reactive Architecture

Reactive architecture incorporates availability by designing systems to be highly resilient and handle failures using techniques like replication, load balancing and fault tolerance. This ensures high availability and responsiveness by distributing the workload across multiple instances and isolating failures without affecting the overall system. [AA15]
Availability Grade: 5

It emphasizes scalability by designing systems to handle increasing workloads and adapt to changing demands. Techniques like horizontal scaling distribute workload across nodes, ensuring capacity increases without compromising performance. It promotes asynchronous communication patterns, enabling simultaneous handling of multiple requests and improving overall performance. However, when horizontal scaling is not effectively implemented it can result in decreased performance and responsiveness rather than scalability. [Tov19b]
Scalability Grade: 4

Deployability is achieved through containerization and orchestration technologies, enabling seamless system scaling. Nevertheless, reactive systems rely heavily on third-party dependencies or external services, which can impact deployability. Critical dependency failures or performance issues can lead to deployment failures or degraded system performance. [DSM+17]
Deployability Grade: 4

Through well-defined APIs integrability is accomplished, which enables seamless data and message exchange between components and services. Event-driven communication patterns and asynchronous messages indicating changes or actions, also contribute to seamless integration. However, tightly coupled systems, where components and services are heavily dependent on each other's implementation details, hinder seamless integration and can lead to cascading failures. [Sob10]
Integrability Grade: 4

Developers must design systems with testability in mind, implement unit tests, integration tests and performance tests and use testing frameworks and tools. Clear boundaries between components and services facilitate testing by facilitating isolation. Certain practices, such as a heavy reliance on external dependencies that are difficult to mock or simulate, may not assure an efficient and testable architecture. [SAM+19]
Testability Grade: 4

The architecture offers ease of development through loose coupling and modularity principles, allowing developers to work independently on individual components. Complex interdependencies, on the other hand, can make development harder. Changes or revisions to one component can

inadvertently impact other components, causing development and testing issues and challenges. [SAM+19]
Ease of Development Grade: 4

Modifiability is accomplished in reactive architecture through the loose coupling of its components therefore modifying or updating one component does not inherently effect the other components. Exceptions may arise when a change or update to one component necessitates modifications in multiple other components due to complex dependencies or shared resources. This can result in a cascading effect of changes throughout the system, making it difficult to maintain and modify individual components without impacting the stability of the system as a whole. [DSM+17]
Modifiability Grade: 4

The objective of reactive architecture is to accomplish usability through intuitive, navigable and responsive user interface design. This includes considering the requirements and preferences of end users, providing timely feedback and adapting the interface based on the user's device or location. Still, cluttered and overwhelming user interfaces can reduce efficacy by making it difficult for users to locate and utilise desired features. Moreover, a lack of clear feedback or prompt response to user inputs can result in frustration and hinder the overall efficacy of a product. [Tov19b]
Usability Grade: 4

2.4.10 Non-Functional Requirements Evaluation Matrix

All QA evaluation marks are summarized in the following Table 2.11.

Approach/Criteria	Availability	Scalability	Deployability	Integrability	Performance	Testability	Ease of Development	Modifiability	Usability
Architecture Centric Design	4	4	4	4	5	3	5	3	5
Microservices Architecture	5	5	4	4	5	4	4	4	5
Cloud-native Architecture	4	5	4	4	4	3	3	4	4
Waterfall Software Dev.	3	4	3	3	3	3	2	1	4
Domain-Driven Architecture	4	4	3	3	3	3	2	3	4
Service Oriented Architecture	5	4	4	5	4	4	5	4	5
Event-Driven Architecture	5	4	3	4	3	3	4	3	5
Model-Driven Architecture	3	3	4	4	3	4	3	3	5
Reactive Architecture	5	4	4	4	4	4	4	4	4

Table 2.11: Architecture Evaluation Matrix

2.5 Conclusion

Enterprises may use this rating to find the best software architecture design method for their dispersed development teams. The rating represents a subjective evaluation of how well each strategy satisfies the characteristics indicated as crucial in the supplementary research question and as such, each strategy has advantages and disadvantages.

Businesses may make better judgments on the most appropriate software architecture design strategies by taking these rankings and the reasoning behind them into account. The final ranking of all software architecture design approaches is presented in Table 5.1 on the basis of the average score calculated from the elements involved in decision criteria.

Approach	Average Score	Rank
Microservice Architecture	4.44	1
Service Oriented Architecture	4.44	1
Architecture Centric Design	4.11	2
Reactive Architecture	4.11	2
Cloud-Native Architecture	3.89	3
Event-Driven Architecture	3.78	4
Model-Driven Architecture	3.56	5
Domain-Driven Architecture	3.11	6
Waterfall Software Dev.	2.89	7

Table 2.12: Architecture Ranking Table

In conclusion, software architecture design success hinges on its ability to use the most effective method of software architecture design while using remote development teams. Microservices architecture, reactive architecture, cloud-native architecture, event-driven architecture, architecture-centric design, domain-driven architecture, service-oriented architecture, model-driven development and the traditional waterfall method were all considered in this literature review. Using quality qualities including availability, scalability, deployability, integrability, performance, testability, ease of development, modifiability and useability the efficacy of each of these methodologies may be evaluated. Yet, the three fundamental structures of software design C&C, module and allocation are incorporated by any approach.

While deciding on a software architectural design strategy, it is essential to take into account the unique needs of the business and the development team. A strategic choice may boost software development's productivity, quality and efficiency, giving the business an edge in the market.

3 Integration Projects Overview

This chapter focuses on the source projects that have been analyzed and evaluated for potential integration points. The analysis and evaluation of these source projects allow for a comprehensive understanding of their strengths and weaknesses, enabling the design of the target system to optimally combine their features. By integrating these projects, the target system can leverage the best elements from each source project, ultimately resulting in a more robust and versatile solution.

With the main goal of automating the scanning and organizing of pallets during warehouse operations, the target system is built out of different Bosch solutions covering the same goals. In order to successfully read the labels attached to the pallets, precisely detect their presence and decide on their ideal placement within the warehouse facility, the system will make use of cutting-edge technologies, such as cameras and visual AI. Additionally, the application will be created to meet unique customer needs, giving users a tailored experience and offering real-time feedback during the process.

To get an overview of the single projects my colleagues and I conducted interviews with the responsible teams working on the different projects which can be found in the Appendix A.

For a variety of reasons, interviewing project teams may be quite helpful. They may aid in establishing clear project goals and objectives, gathering information, seeing possible problems, developing relationships with team members and getting feedback. Using interviews, project teams can make sure that everyone is on the same page and works toward the same goal, as well as determine what features and functionalities are essential for success, address potential problems before they arise, build relationships between team members and continuously improve processes.

The prepared questions are as follows:

1. How does a live demo into the system look like?
2. How is the system used on a daily basis?
3. What are the strengths and weaknesses of the system?
4. How is the system setup currently done?
5. What are the possibilities of integrating the system?

The following Table 3.1 shows a quick comparison matrix, which resulted as the outcome of the project team interviews.

	Description	ShipSmart	OIS	Camera Area	AutoID	ShipQ
Plants	How widely used is the system	★	★	★★	★★★	★★★
Software Rollout	How quickly and reliably can new features be rolled out	★★	★★★	★★	★	★★
Hardware Rollout	How simple is the physical deployment and related maintenance	★★	★★	★★	★★	★★★
Technology	How mature and state of the art are the used technologies	★	★★★	★★★	★★	★
Extensibility	How easy is it for the system to be extended by features	★	★★	★★	★	★
Integration	How easy is it to integrate the system with other systems	★	★★	★★★	★★	★
Operations	How mature is the operational model of the system	★	★	★★★	★★	★★★
User Experience (UX)	How mature is the user experience	★	★★	★★	N/A	★

Coverage scale: Low = ★ *Medium* = ★★ *High* = ★★★

Table 3.1: Project Comparison Matrix

3.1 Business Understanding

This section describes the system from a business perspective, targeting questions like who the customer is and what their key challenges are. Furthermore, the stakeholders' roles with the related amount of people and their connection to the target solution are listed.

Question: Who is the customer?

Explanation: This question does not address the final end-user, but rather who will be the system's contractual customer. In other words, who will pay for Bosch's use of the system? The response should indicate not only the client type (Business-to-Business (B2B) or Business-to-Consumer (B2C)), but also the targeted market group. The rationale is to find out from which perspective requirements should be analysed.

Response: Internal B2B - Bosch Production Plants or Logistics Entities

---

Question: What are the customer's key challenges?

Explanation: Determine the customer segment's primary challenges or pain points. This study should already reveal a problem that the system under development will solve and the qualities that it should have.

Response: Packaging of pallets for the outbound logistic process requires customer-specific labelling in terms of size, placement, number of labels, straps, etc. This is manually checked based on customer requirement documents and often leads to shipments that cannot be processed by the customer, leading to complaints/claims and returns.

There is no central tool and database available for the plants that include all customer labelling and packaging requirements with user-friendly visualization and guidance for the operator that can be used on both monitor and mobile device.

Validation of customer requirements is a completely manual process at the moment. Operators conduct a 4-eyes-check of the pallet before it is loaded into the truck, which leads to long process times and mistakes. For Bosch: Each plant maintains and consolidates the customer requirements on its own in different formats: PDF, PowerPoint, Excel or ShipQ App.
Goal: Zero complaints, returns and incorrect shipments based on wrong, insufficient or missing labelling of pallets or generally incorrectly packaged shipments.

Question: What problem shall the system solve?

Explanation: This should be a brief, succinct description of why buyers should purchase the solution. It indicates which functionality is vital and therefore has the highest priority in terms of quality standards.

Response:

1. Provides one source of centrally maintained customer requirements with user-friendly visualization to the packer and forklift driver.
2. Fast and automatic visual outer package/pallet validation based on the same requirements during packaging to prevent labelling or packaging errors (correct placement of labels, barcode readability, straps etc.).

3. Avoid time-consuming 4-eye principle check and generally speed up of validation process.
4. Enable easy claim checking by archived inspection result.

Question: Will the final product or solution include components that are special to the customer?

Explanation: The question probes the extent of the solution's adaptability. If there are a lot of customer-specific components needed, either the solution needs to be constructed from the ground up to support that (cost driver) or it needs to support several versions, which require maintenance. This could be a sign of the solution's distribution model (Software-as-a-Service vs. separate on-premise installations).

Response: Specific customer needs will be applied through input parameters and solution configuration, such as required packaging, straps, label location and so forth. In addition, the quantity and type of tests to be performed will differ depending on the customer.

Dashboards displaying Key Performance Indicators (KPI)s and status information may also need to be adaptable to meet the needs of different customers. However, it is assumed that no unique features (software codes) are provided per client, but that all customers can use the whole capabilities if necessary.

Question: Is the intention to commercialize the system?

Explanation: What is the business strategy and techniques for profiting from the solution? Is it necessary to be self-sufficient? Are there any thoughts on pricing methods (pay-per-use, one-time license, free trial, etc.)?

Response: One aspect of commercialization is to end the existing practice of consumers levying fees for incorrect delivery. Handling time is reduced or eliminated, as is the manual 4-eye-check during validation.

Reduced time and effort in managing client requirements. Depending on the client portfolio and the number of new projects, it is anticipated that the work for establishing a new requirement takes around two hours per customer and maintaining the manual handbook takes around ten hours per month.

Question: What are typical constraints in the target market?

Explanation: Depending on the target industry and market, there might be typical constraints that have to be considered during the system design. For example, business-critical data is often required to be kept on premises in large, conservative companies, especially in particular industries.

Response:

- Very diverse local environments regarding space, condition and network connectivity.
- Logistics processes are time critical and thus a very high resilience of the solution is needed or a simple fallback.
- Inspection software should run on-premise (due to image transferring time to a cloud infrastructure).

- Ambient lighting conditions can be a challenge for the correct processing of tests and should be mitigated (e.g. lightning system).
- Logistics processes must be flexible and cannot be entirely rigid. They also differ significantly between plants and Bosch units. Validation setup must cater to tight/narrow spaces on-site and thus hardware must be as less intrusive as possible.

Question: What is the product's unique selling proposition?

Explanation: It should be clear how the system will differentiate itself from others on the market. The rationale is to find out which parts might be more sensitive and critical to the overall success.

Response: Management and identification of correct labelling requirements by customers to provide guidance. User-friendly and simple process guidance for pallet labelling and packing for the shipping operator as well as for the forklift driver during the pallet validation. Packed and labelled pallet is automatically validated by camera-based recognition before loading, based on the same source of shipping requirements. A photo of the loaded pallet is archived for documentation in case of claims/complaints. Test types are extensible and can be performed in a fast and automated fashion. Overarching solution including hardware and software.

Question: What are the three most important quality characteristics of the system? (And why?)

Explanation: Ideally, the most important ones should become apparent by the answers provided to the questions above. These are especially important since they should reflect or support the unique selling proposition (e.g. usability, security, performance).

Response:

- Usability (ease of use for workers, clear feedback, clear instructions)
- Performance (validation speed to avoid any delay in loading speed)
- Availability (fallback possible but important for logistics processes)
- Modifiability (to add additional validation logics or steps)

Question: How many customers does the current system have?

Explanation: If there are already existing customers, how satisfied are they? If there are none, has the product idea and business model been discussed with potential customers and are customer representatives available?

Response: At least six plants in Germany and two plants in Portugal are willing to have the system as soon as possible.

Question: Are there solutions in production? (If not, when is the release planned?)

Explanation: This should help to provide a timeline for any required tasks and prioritization.

Response: Yes, the Feuerbach solution is already in use. The other projects are in different states of maturity.

Question: In which countries should the solution be offered?

Explanation: The questions should aim at countries that are “trickier“ to support such as Russia, India or China given the customer is from Europe or North America. The impact can be additional software hosting environments and strong requirements towards data location, privacy etc.

Response: Worldwide for any Bosch plant.

Question: How critical is the solution for the (business) success of the customer?

Explanation: The questions shall serve as input regarding the required level of support and operation of the solution. It can also hint that an important quality aspect is the reliability of the solution.

Response: Highly critical. It can significantly increase shipment quality and avoidance of complaints and delivery delays.

Question: Are there any collaborating partners/suppliers on which the solution heavily depends?

Response: Yes.

- SAP
- Hardware (AutoID)

Question: What is the result of the Data Classification and Risk Analysis?

Explanation: Data classification can be driven from two perspectives: data privacy (e.g. is person-relatable data processed?) and data security (e.g. what is the financial or reputational impact of losing data?). This classification should be done as early as possible to drive architectural decisions.

Response: System needs to consider that the label’s content will be recorded and needs to anonymize persons in the camera’s field of view.

Question: Does the solution need to support Multi-Tenancy (i.e. need to manage users from different organizations in multiple, isolated environments?)

Explanation: As soon as multiple users that share data are bundled into “working groups“, it can be very likely assumed that multi-tenancy is required. This normally increases the complexity of the solution regarding tenant isolation (performance, data, access) and often conflicts with the intended type of software delivery or the presentation towards the customer (fully branded customer specific environment).

Response: Yes, Salesperson, plant key user, packer, forklift driver.

Question: What is the typical way to become a customer and use the application?

Explanation: Are users onboarding themselves or are there more elaborate onboarding processes requiring extensive validation? Are power users involved that manage a tenant’s users?

Response: Direct connection to the business unit and rollout within the corresponding plants. Driven by the solution provider.

3.2 Stakeholder Overview

Stakeholders are people or organizations that are interested in a project's success. Customers, users, developers, project managers, executives, stockholders and regulatory authorities can all be stakeholders in software projects.

Each stakeholder in the software project has a distinct function and it is their responsibility to contribute to the project's success by offering suggestions, criticism and support in a variety of ways. Depending on their position, stakeholders in software projects, may have different duties. However, the stakeholders regarding the target project include:

Project Lead: The project lead is in charge of leading the project team, planning and overseeing the project schedule, keeping track of developments, controlling the project budget and making sure the project is completed on schedule, on target and to the needed quality standards.

Solution Architect: An organization's needs and overall business objectives must be met by the proposed solution for the solution architect to provide technical knowledge and direction. They collaborate closely with other stakeholders to establish the solution's technical architecture, identify and address technical risks and make sure the system is scalable, dependable and secure.

Process Owner: The procedures that support the aims and objectives of the organization must be defined, documented and improved over time. The process owner collaborates closely with stakeholders to suggest process enhancements, guarantee adherence to relevant regulations and norms, assess the efficacy of the process and execute modifications as needed.

Coordination Lead: The coordination lead is in charge of overseeing how activities are coordinated between various teams or departments within a company as well as with outside partners or contractors. Their responsibilities also include identifying dependencies and risks, establishing communication routes and protocols, resolving disputes or concerns, monitoring development, spotting potential improvement areas and regularly updating stakeholders on the status of the project.

Process Manager: As a stakeholder, the process manager oversees monitoring and enhancing procedures within a division or company. This entails examining present procedures, finding potential areas for improvement and putting new procedures into place to boost output, cut expenses and improve quality. They converse with other parties to achieve alignment with company goals and objectives, including top management and department leaders.

Product Expert: A product expert participates in numerous activities linked to the creation, promotion and upkeep of a product. They interact with the development team, do research, specify needs, solicit input, offer guidance on product features, produce documentation and training materials and take part in launch and marketing efforts.

Technical Lead: A technical lead's role includes ensuring that a product is developed, delivered and maintained successfully. They are responsible for managing the development team, supervising the technical architecture and design, taking part in product planning and roadmap creation, working with cross-functional teams, conducting code reviews, troubleshooting technical problems, staying current with the newest technologies and spotting opportunities for improvement.

Software Developers: Stakeholders who are in charge of creating the software are developers. Their responsibility is to design, develop and test the program using their technical expertise. They could also provide suggestions on how to improve the software's usability and usefulness.

Project Managers: Stakeholders in charge of managing the software project are called project managers. Their responsibility is to make sure the project is finished on schedule and on budget. They could also engage with other stakeholders and offer the development team direction and assistance.

Sponsors: Sponsors are stakeholders who have invested in the company developing the software. Their job is to ensure that the project is profitable and provides a return on their investment. Stakeholders in software projects have a collective responsibility to make sure the project is successful and serves the needs of all parties involved.

Users: Once the software is finished, users are the stakeholders who will use it. Their responsibility is to offer feedback on the user experience and give suggestions for enhancements or extra features that would increase the software's use for them.

Role	Number	Area of Knowledge	Project
Project Lead	1	Standardize and create solution for rollout	InTrack
Solution Architect	3	System understanding and architecture	N/A
Process Owner	2	Domain Understanding and User Journey, Oder to cash	ShipSmart, ShipQ
Coordination Lead	3 at different Plants	Plant Environments and Process Journey	ShipSmart
Process Manager	1	SAP Co-Innovation	ShipQ
Product Expert	1	SAP	Sap
Technical Lead	4	Architecture, Integration, Technology Stack, Architecture, Computer Vision	OIS, Camera Area, InTrack
Software Developer	1	Architecture, Software Implementation and Image Recognition, Hardware	OIS, ShipSmart, AutoID, CrossTalk
Program Integration Manager	1	Packing instructions, SAP Program integration manager	Nexeed Packaging Control
Project Sponsors	3	N/A	N/A
Project User Groups	3	N/A	N/A

Table 3.2: Stakeholder Overview

3.3 System Context

The designed system will support a wide range of user jobs, including Managers, Packers, Forklift Drivers, Logistics Key Users, Customer Managers, Local Support Engineers and Central Support Engineers. Each user will be given access to purpose-built user interfaces that correspond to their particular responsibilities and tasks inside the warehouse operation. The Table 5.2 presents the actors involved in the target system and user interaction is illustrated in Figure 3.1.

Actor	Description
Logistics Key User (Planner)	Manages customer requirements, claims and notifies packers and/or forklift operators of any changes. Gets packaging information per consumer or item through a streamlined procedure and distributes it to the shipping area.
Packer	Responsible for the proper packaging and labelling of pallets, can access customer requirements from the system, initiate the scanning process and validate the scanning results.
Forklift Driver	Transports deliveries to the loading dock using a forklift. May need to manually initiate the scanning process or validate scanning results.
Salesperson	Responsible for collecting and managing client requirements and effectively disseminating them throughout the organisation.
Manager	Displays aggregate packing quality control and flow data for one or more facilities. Analyses procedure effectiveness.
Other Teams	For example the shop floor team receiving notifications regarding instruction update etc.
Hardware Support Engineer	Installing and maintaining on-site hardware for any edge-based processing or data storage (e.g., image compression, image storage)

Table 3.3: Actor description

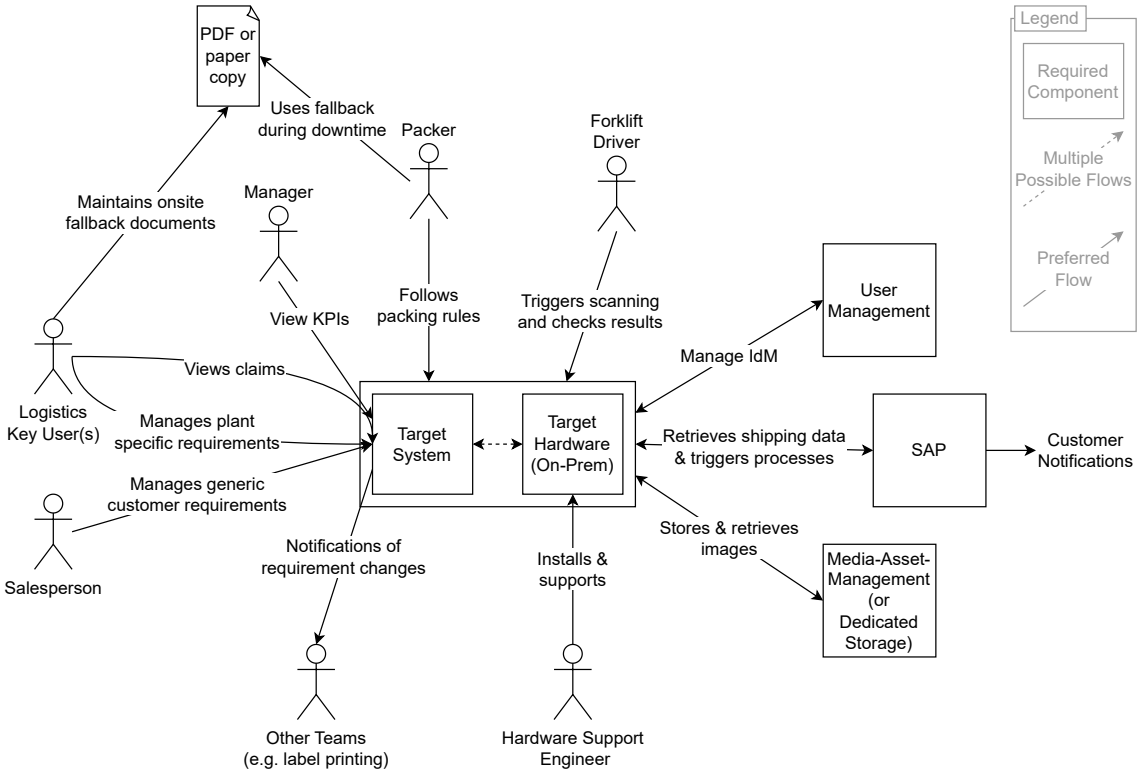


Figure 3.1: User Interaction of Target System

4 User Stories

User stories are created to convey the requirements of the project from the perspective of a user or client. The purpose of a user story is to clearly and concisely describe what the user desires for the product to accomplish. The user narrative consists of a concise statement outlining the users' objectives or problems. It also includes additional information, such as how the stories are prioritized, the sort and perspective of each story and the maturity of each project's cover.

The purpose of the user narrative is to ensure that the development team is aware of the users' or clients' requirements and that the resulting software effectively meets those needs. User stories are frequently used by agile software development methodologies to manage and prioritise the development process based on the most important user requirements.

The following section describes the functional requirements regarding the target system. Furthermore, the non-functional requirements of the target system are prioritised, which will later guide and factor in the architecture selection.

4.1 User Stories Functional Requirements

The following Table 4.1 summarizes the resulting functional requirements in the form of user stories. Regarding shipment validation, the most pertinent user stories are derived from the packer and forklift driver perspectives.

Prioritizing the functional requirements in accordance with their importance is essential. This approach proves particularly beneficial when multiple requirements coincide or certain deadlines need to be met, ensuring that the one with higher priority is addressed first. "A" represents the highest priority, while "C" denotes the lowest priority.

ID	Priority	Requirements (I want to ...)	Ship Smart	OIS	Camera Area	AutoID	ShipQ
Actor: Planner							
PL1	A	have packaging information per customer	★	★	nc	nc	★★★
PL2	A	have a standardized template for packaging information	nc	★★★	nc	nc	★★★
PL3	C	have a document management system for customer unstructured data like PDFs	nc	nc	★★★	nc	★★★

Continued on next page

Table 4.1 – *Continued from previous page*

ID	Priority	Requirements (I want to ...)	Ship Smart	OIS	Camera Area	AutoID	ShipQ
PL4	A	have a structured worklist and easy access to relevant data	nc	nc	★	nc	nc
PL5	B	have access to a repository of packaging information	nc	★★★	nc	nc	★★★
PL6	C	have an overview of claims	nc	nc	nc	nc	nc
PL7	A	have correct material master data with weight, dimensions etc.	nc	★★★	★★★		★★
PL8	B	have overview of changes made by customer with deadline of implementation	nc	nc	nc	nc	★★★
PL9	A	have a digitalized form of customer requirements in a machine-readable format	nc	★★★	nc	nc	★★★
PL10	C	have a digitalized form of internal and external customer requirements	nc	nc	nc	nc	nc
PL11	C	have the correct time zone in SAP	nc	nc	nc	nc	nc
PL12	A	forward the updated info about the customer requirements to operators, notify operators, notify sales team about changes	nc	nc	nc	nc	nc
Actor: Packer							
PA1	A	have packing instructions in the right language	nc	★	nc	nc	★★
PA2	A	have immediate (<3 sec) feedback, if packaging applies to agreed instruction	★★★	★★★	★	★★★	nc
PA3	A	have easy access to packaging information	★★★	★★★	★★★	nc	★★★
PA4	A	avoid many clicks – easy to handle	★★★	★★★	★★★	★★★	★★★
PA5	B	to have a process which is guided (only focus on execution)	★★★	★★★	★★★	★★★	★
PA6	C	see how much time I have left to finalize the packaging	nc	nc	nc	nc	nc

Continued on next page

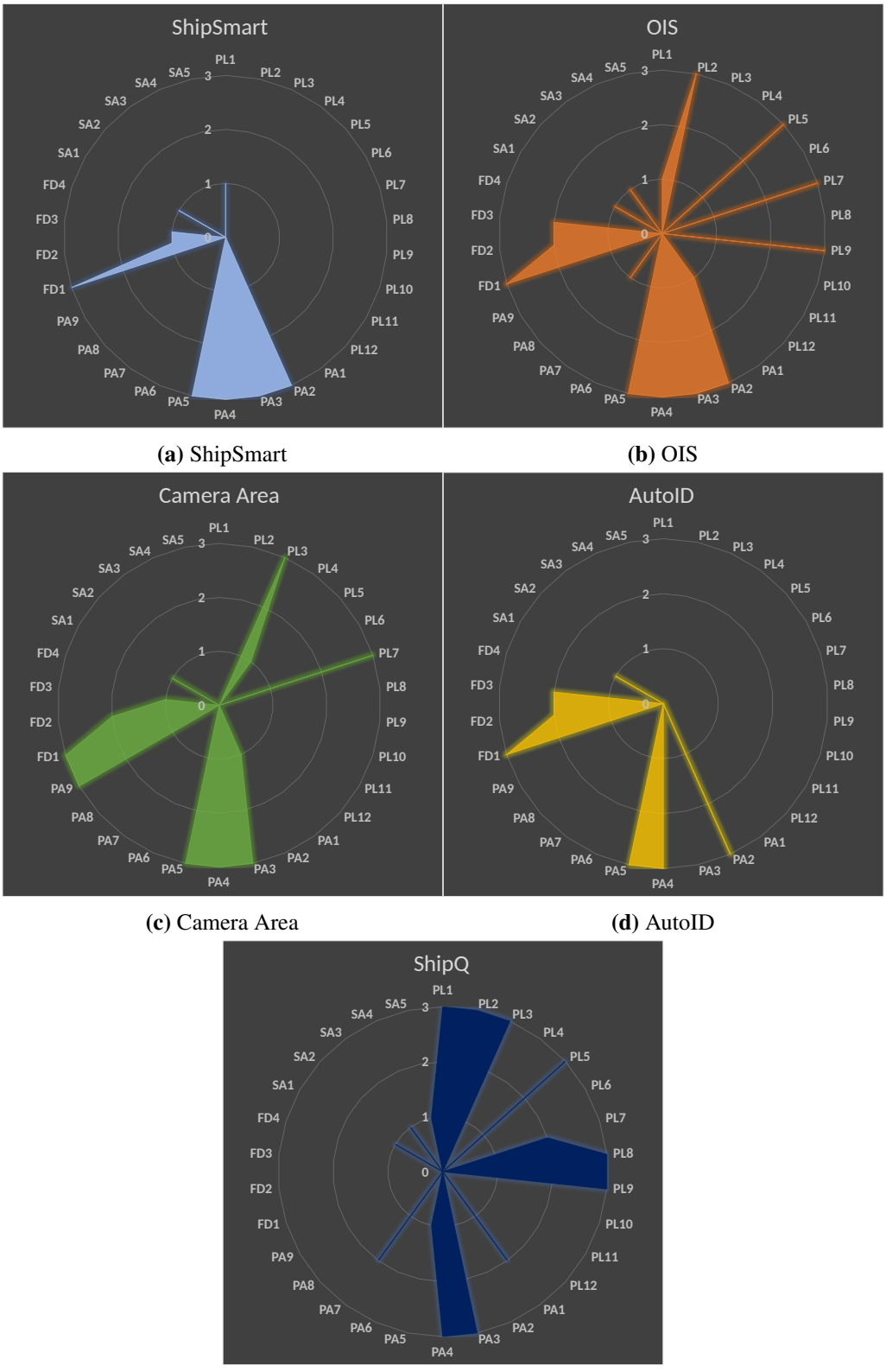
Table 4.1 – *Continued from previous page*

ID	Priority	Requirements (I want to ...)	Ship Smart	OIS	Camera Area	AutoID	ShipQ
PA7	B	be informed on time about the customer changes	nc	★	nc	nc	★★
PA8	B	to have a station/place where the pallets are photographed before loading	nc	nc	nc	nc	nc
PA9	C	that the delivery documents can be sent automatically from SAP to the customers	nc	nc	★★★	nc	nc
Actor: Forklifter							
FD1	A	handle the pallets that validation only takes 3 sec	★★★	★★★	★★★	★★★	nc
FD2	A	work manually as little as possible	★	★★	★★	★★	nc
FD3	A	have clear instructions and visualization after validation	★	★★	★	★★	nc
FD4	A	have good and safe working environment (camera station safety e.g. cables, lights)	nc	nc	nc	nc	nc
Actor: Salesperson							
SA1	A	avoid incoming claims, because of wrong packaging	★	★	★	★	★
SA2	A	to have clear information to whom to inform about customer requirement changes	nc	nc	nc	nc	nc
SA3	A	have a clear process flow (get requirement, distribute) & changes	nc	★	nc	nc	★
SA4	B	have feedback for further conversations with customers (e.g. when the plant is ready)	nc	nc	nc	nc	nc
SA5	A	know, which format is best to distribute information about customer requirements	nc	nc	nc	nc	★

Coverage scale: nc = not covered Low = ★ *Medium* = ★★ *High* = ★★★

Table 4.1: Requirements of User Stories

Figure 4.1 visualizes which requirements are satisfied to what extent by each project in order to gain a more precise understanding. A combination of Optical Inspection System (OIS) and Camera Area (CA) will aid the architecture modelling in the coming sections.



(a) ShipSmart **(b)** OIS

(c) Camera Area **(d)** AutoID

(e) ShipQ

Figure 4.1: Use Case Coverage Diagram

4.2 Non-Functional Requirements

Availibility	
Question	What is the system's predicted uptime?
Answer	• Warehouses require 24/7 operational systems • Critical to ensure reliability and uninterrupted operations
Question	Is there a specified time when the system must be available or unavailable?
Answer	• System must be operational at all times • Cannot be offline during peak shipping and receiving hours • Interruption might cause shipment delays and disruptions
Question	How soon should the system be able to recover from a malfunction or outage?
Answer	• Rapid system recovery is crucial to prevent downtime and disruptions • Requiring efficient disaster recovery plans and reliable fallback solutions
Scalability	
Question	What is the predicted rate of user growth over the next 1-3 years?
Answer	• User base growth predicted at 15% annual pace due to warehouse expansion
Question	How many concurrent users must the system support?
Answer	• System supports 5 concurrent users, varying by warehouse
Question	What is the anticipated volume of data that the system must handle?
Answer	• The system handles 65 pallets daily, capturing 30 MB photographs for each pallet • Resulting in a daily data volume of 7800 MB
Question	What are the system's performance objectives at various degrees of load?
Answer	• System aims for a 29-second pallet scanning time for 95% receiving operations • Handling double load without significant performance loss
Deployability	
Question	How frequently do you anticipate the system being deployed?
Answer	• Quarterly deployment of the system including upgrades and new features
Question	What are the system's deployment environments?
Answer	Three deployment environments: • Development environment: develop and test new features • Staging environment: user acceptance testing • Production environment: deployment
Question	Is it necessary to employ any specific deployment tools or processes?
Answer	• IT department approves Docker and Jenkins to ensure consistency and efficiency

Continued on next page

Table 4.2 – *Continued from previous page*

Question	What are the system configuration and customization requirements during deployment?
Answer	• Flexibility during deployment, allowing for updates to parameters and integration with other systems • Adaptable to user roles and permissions, ensuring being customized to fit specific warehouse needs
Integrability	
Question	What other systems must integrate with the system and how will they do so?
Answer	• Integrate customer relationship management (CRM) system with REST API for proper order and customer sync between platforms
Question	What integration protocols or data formats are required?
Answer	• JSON and XML are essential data transmission formats for lightweight, widely used and easy-to-read data in enterprise systems
Question	Are there any integration performance or security requirements?
Answer	• Integration performance is crucial for real-time inventory and order status updates, ensuring quick, dependable processing and preventing errors • Security is essential for safeguarding customer information and preventing data breaches
Question	What are the system monitoring and logging needs during integration?
Answer	Extensive logging and monitoring are essential for: • Detecting issues during integration • Providing real-time alerts and notifications • Ensuring system performance and resource utilization
Performance	
Question	What are the system's estimated response times under various usage scenarios?
Answer	• System should respond to user requests for all pallet responses within 3 to 5 seconds, depending on the number of pallets scanned simultaneously
Question	What are the performance objectives for user actions or system transactions?
Answer	• Planner should rapidly edit and publish packaging information to keep inventory and orders current • In less than 5 seconds, forklift driver should scanning results to ensure that products are packed correctly and effectively
Question	What are the system monitoring and recording requirements for performance testing?
Answer	• Performance testing involves extensive logging and monitoring to ensure system meets requirements, tracking response times, error rates and resource utilization • Simulate usage degrees, identifying bottlenecks and areas for improvement
Testability	
Question	What are the system's expected testing scenarios?

Continued on next page

Table 4.2 – *Continued from previous page*

Answer	Functional testing: • Confirm system's functioning meets the business requirements • Evaluate system's ability to handle a variety of inputs and processes Integration testing: • Evaluate system's ability to integrate with other systems • Evaluate system's capacity to exchange data and communicate with other systems Usability testing: • Determine how simple it is for users to operate the system • Identify any locations where users may struggle or have difficulty navigating the system
Question	What exactly are the testing requirements for each scenario?
Answer	• Prior to functional testing, specific requirements and use cases must be created and tested, including order status updates and order processing • Integration testing involves data interchange, communication and overall functionality • User testing is crucial to ensure the system is easy to use and understand, with focus on interface design, navigation and user experience
Ease of Development	
Question	What are the development tools and technologies that will be required for the system?
Answer	Development tools and technologies: • Cloud services: Microsoft Azure and Google Cloud • Container services: Docker • Messaging backbones: Kafka and REST API • Programming languages and frameworks: Python Django, Spring and Angular • Monitoring and visualization: Grafana and LabView
Question	What are the documentation and knowledge transfer requirements during development?
Answer	• Documenting code inside the source code itself • Additionally producing and maintaining extensive documentation in a centralized location • Documenting system architecture, processes, coding standards and testing procedures, to ensure understanding of the system and its components
Question	Is there a set of development rules or criteria that must be followed?
Answer	• Software architects should create precise development rules and criteria based on industry-standard methods • Including test-driven development and CI/CD • As well as covering coding standards, version control and code reviews
Modifiability	
Question	What changes to the system are expected in the future?

Continued on next page

Table 4.2 – *Continued from previous page*

Answer	• Expected to undergo upgrades, including new label types and integration with inventory tracking systems • Accurate real-time inventory updates may require linking to existing systems • Additional workflows for managing pallets and boxes are planned, addressing challenges in handling large or odd-shaped items
Question	What are the system's specific regions that are expected to change frequently?
Answer	• Label recognition algorithms and integration points undergo frequent changes, requiring updates as new labels or inventory tracking systems are introduced
Question	What are the requirements for system backward compatibility when making changes?
Answer	• Any changes made to the system should be thoroughly tested to ensure that they do not interfere with the system's existing functionality
Question	Are there any design patterns or architectures that must be adhered to ensure simplicity of modification?
Answer	• A SOA that separates forklift tracking integration points from the system, ensuring changes don't affect each other • Modular design pattern simplifies component changes without affecting other parts
Usability	
Question	What user personas should the system expect?
Answer	• Planner, packer, forklift operator and salesperson are the user personas that are anticipated for the system • System must be created to meet the needs of each user persona
Question	What particular user actions or situations does the system need to support?
Answer	• User duties like reading text from labels, processing data and deciding what steps the warehouse delivery process will take next
Question	What conditions must the system meet to handle errors and get user feedback?
Answer	• Visual indicators and error notifications • Gracefully handle errors and exceptions while guiding users on how to fix the problem • Prioritize usability and simplicity to increase user pleasure and productivity

Table 4.2: Non-Functional-Requirements Evaluation

5 Architecture Modelling

As a result of thorough analysis and evaluation of each project, the following applications have been excluded from consideration for integration into the target system.

- ShipQ: This application is built on the SAP Solution and integrates Fiori UX tool suite. Since it does not pertain to shipment validation and due to the long development processes, it will not be considered further. But for future integration, it is unquestionably worthwhile to prepare interfaces beforehand.
- AutoID: The AutoID team is more concerned with the tools in warehouse settings. The entity has experimented with smart cameras, but the high costs makes them inappropriate for the intended solution.
- ShipSmart: This initiative is based on the LabView application developed by a third party. Since the entire project logic is incorporated into this application, it is not considered due to its difficult integration options and high licence fees per user.

As discussed in the literature review, the ranking table (Table 5.1) is utilised at present to determine the architecture approach. To adapt the ranking table to the requirements of the project, the results of the questionnaire in the Section 3.1 are required. Usability, Performance, Availability and Modifiability are deemed crucial in this section. Consequently, these attributes are double-weighted and the architectures are reordered.

Approach	Average Score	Rank
Microservice Architecture	4.54	1
Service Oriented Architecture	4.46	2
Architecture Centric Design	4.15	3
Reactive Architecture	4.15	3
Cloud-Native Architecture	3.92	4
Event-Driven Architecture	3.85	5
Model-Driven Architecture	3.54	6
Domain-Driven Architecture	3.31	7
Waterfall Software Dev.	2.85	8

Table 5.1: Architecture ranking table updated based on questionnaires

Resulting from the updated ranking table the selected architecture for the target system is the microservice architecture approach. The methodology described will be utilized and incorporated into the architecture drafts, as further elaborated in Section 5.3.

5.1 Architecture Decisions

As discussed in the aforementioned “Non-Functional Requirements“ section, the Figure 5.1 depicts the integration of the OIS and CA projects. It is evident that the combination of these two projects falls short of meeting several requirements. Notably, the integration of the ShipQ project would effectively address a majority of the requirements. However, as previously explained, it will not be pursued.

The target system requires significant attention to address the functional requirements that are either absent or have a low level of implementation. Specifically, for the shipment validation, the *PA6, PA7, PA8 and FD4* should not be neglected.

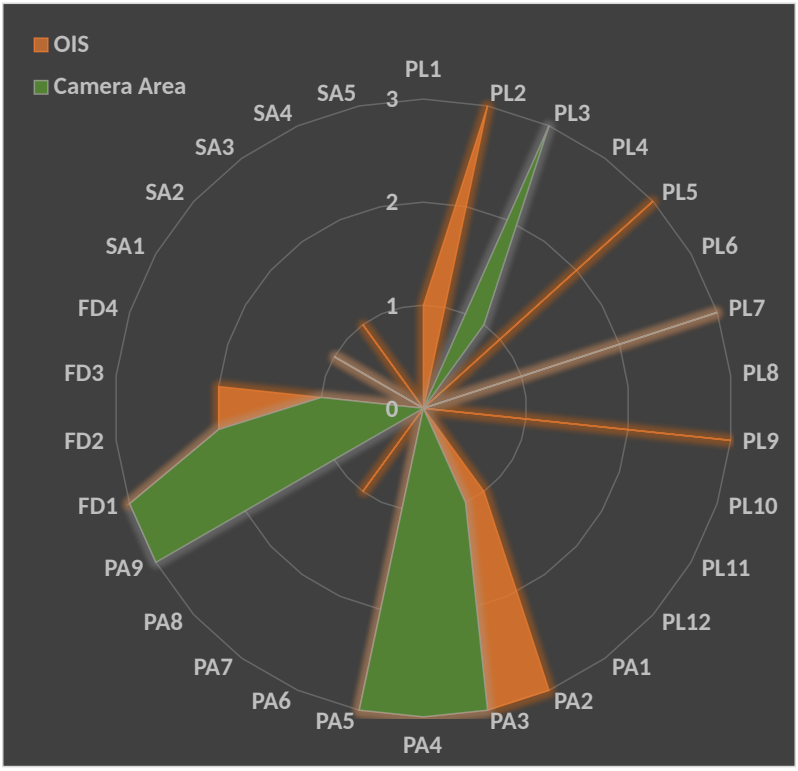


Figure 5.1: OIS and Camera Area

Based on an understanding of the users needs and general assumptions regarding the target system, the subsequent Architecture Decisions (ADs) were made. The problem and the decision of each AD are disclosed. The detailed analysis can be found in Appendix B. The decisions encompass the overall architecture, with particular emphasis on decisions *1, 5, 6 and 7*, which significantly shape the architecture for the shipment validation part.

AD-01: Camera Hardware	
Problem Description	Validation of shipments require optical inspection using the 4-eye principle or image capture through cameras and image recognition technologies.
Decision	**Plain Cameras**
AD-02: Customer Requirements Definition	
Problem Description	Pallet packaging for shipping goods varies based on plant, shipping party and customer requirements. Finalizing these requirements takes several iterations. Currently, plants maintain their own set of requirements as PDF or offline handbooks, which are not machine-readable and have limitations in re-use, comparison and searchability. They are also not suitable for automated validation. Thus it is crucial to have customer requirements for packaging: • in machine-readable and standardized format • that can be used as a validation target for images • that must be flexible to cater to all kind of edge cases • that must reflect the different dimensions of an instruction (plant, product, ship to party, etc.) • that can be defined with a visual editor
Decision	**OIS**
AD-03: Customer Requirements Visualization (Packer UI)	
Problem Description	The packaging for shipping goods via pallets varies based on plant, shipping party and other factors. Those requirements by the customers must be available during the pallet packaging to the packer to correctly label the package, add straps, sticky dots etc. Currently, every plant maintains its own set of requirements often as PDF or offline handbooks. These are not always following the exact pattern and make it harder for the planner. Also, offline instruction is sometimes not as convenient as a digital version. Thus the requirement is to have customer requirements visualized for packaging: • usable on different end-devices (android tablet, computer, packaging stations) that might not have full internet access • web-based for interoperability • with no room for interpretation and very clearly guides the packer (UX relevance) • automatic visualization based on an initial scan of the product via barcode or manual input
Decision	**OIS Labeling App**
AD-04: Customer Requirements Fallback	

Continued on next page

Table 5.2 – *continued from previous page*

Problem Description	Shipping of goods is a critical process in the supply chain to ensure delivery to other Bosch plants but to also avoid claims from customers that are waiting for parts. A technical incident with the visualization system of customer requirements (network, software bug, hosting provider issue, latency) should thus not impact the shipping of the goods in a way that it comes to a standstill. A fallback is required. The fallback should work: • without an IT system • be easy to use • not rely on another technical system that might face the same problems (network, hosting outage)
Decision	**Digital Files: Local PC and remote storage**
AD-05: Packaging Validation Hardware Setup	
Problem Description	The finished pallets have to be validated in a timely manner based on image recognition. There are multiple implementations available that are based on images. Usually, multiple images from different angles are analyzed to deduct if the package conforms to a defined rule set such as: • label placement • label existence • number of sticky dots and their placement • number of straps The requirement is to quickly indicate to the forklift driver if the pallet is ready for shipment. Thus the recognition must be: • reliable in different conditions (dust, humidity, temperature, lighting) • fast in terms of validation results (<2-3 seconds total) • easy to use by means of pallet positioning (margin for error or enforced accurate placement) • fast in terms of usage (driving through the camera gate or quickly positioning and picking it up again) • should support future validation options (e.g. correct boxes, colour, damage detection etc.)
Decision	**Suspended / Gate (Stopping)**
AD-06: Packaging Validation Software	

Continued on next page

Table 5.2 – *continued from previous page*

Problem Description	The finished pallets have to be validated in a timely manner based on image recognition. There are multiple implementations available that are based on images. The requirement is to quickly indicate to the forklift driver if the pallet is ready for shipment. Thus the recognition must be: • fast in terms of validation results (<3 seconds total) until feedback • easy to use in terms of indicating what is wrong • should support future validation options (e.g. correct boxes, colour, wear and tear) • should have means to integrate towards a customer requirements specification system and SAP • must support proper long-term operations and maintenance of the software stack • must allow for a joint collaboration effort to bundle know-how and resources
Decision	**Combination of OIS and CA**
AD-07: Packaging Validation Fallback	
Problem Description	In case the intended system for validation fails different options for a fallback exist. We can assume there are multiple reasons for a validation outage such as: • customer requirements for validation input are not available (e.g. OIS is down) • camera or other hardware is damaged • the network connectivity is impaired • the validation software has a bug or is not available • there is a major outage at a provider or hosting environment
Decision	**Manual 4-eye principle**
AD-08: Delivery Model Backend	
Problem Description	This decision will evaluate different delivery options for the planned solution, meaning how will the solution be sold and delivered to the customer. This is a fundamental decision when developing a new solution since it affects a lot of aspects of the solution and has a lot of implications. Those implications will be compared in the following. An important requirement to fully evaluate the impact is the planned business model to monetize the solution.
Decision	**Fully managed services offering (Software-as-a-Service)**

Table 5.2: Architecture Decisions

5.2 Building Block View

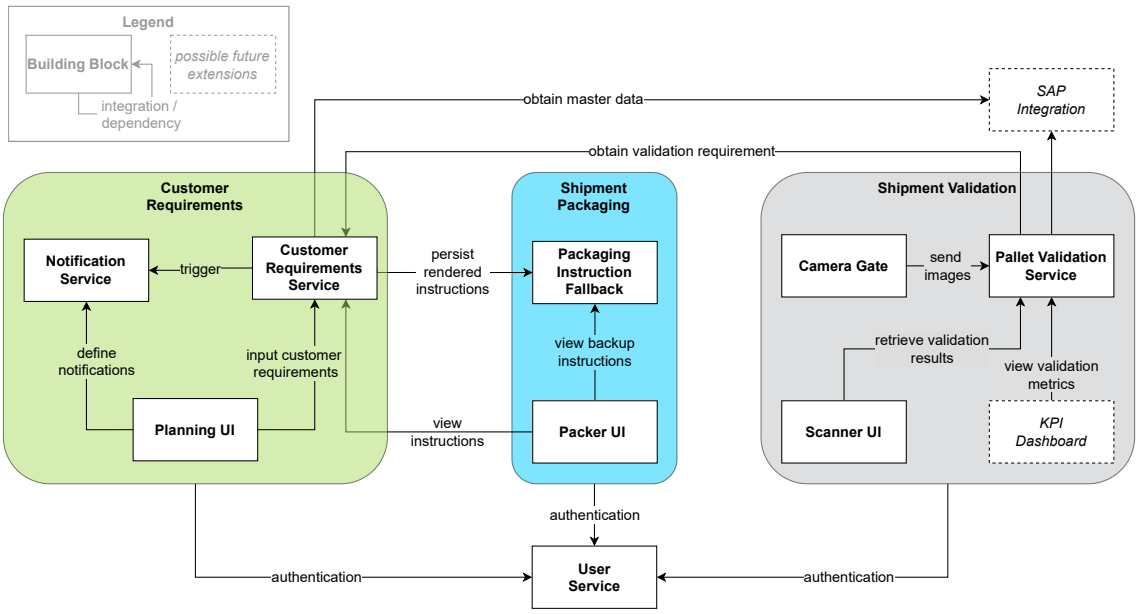


Figure 5.2: Building Block View

The accompanying Figure 5.2 depicts the building block perspective. It incorporates the ADs from Section 5.1 into one view and is built up of three parts: Customer Requirements, Shipment Packaging and Shipment Validation. Each part comprises distinct authentication-related components linked to the User Service. Let's examine every element in greater detail.

The Customer Requirements section oversees all customer management aspects. There are three main components. The **Notification Service** is responsible for transmitting timely notifications to relevant users regarding the arrival of new pallets or any other urgent updates. Second, the **Customer Requirements Service** is responsible for managing and processing the requirements of each customer. It ensures that any special instructions or limits provided by the customer during pallet management are taken into account by the application. Finally, the **Planning UI** incorporates a Manager- and Logistics-specific user interface. Using this interface, they can effectively plan and manage the placement and arrangement of pallets in the warehouse. They can review and modify requirements, allot resources and monitor the entire process. This UI is a result of the *second AD*. To ensure that only authorised users can access and interact with the application, all Customer Requirement components are linked to the **User Service** for authentication.

The primary emphasis of the Shipment Packaging block is the packaging and preparation of pallets for warehouse storage. This block is propelled by two significant components. The **Packaging Instruction Fallback** applies when specific packaging instructions are unavailable or cannot be processed, fallback serves as an alternative mechanism. It ensures that the packaging process will proceed efficiently even if precise instructions are not provided. As discussed in the *fourth AD* this fallback is realized through a local computer which accesses a remote storage to gather the instructions. The second component, the **Packer UI**, provides an intuitive interface designed specifically for the Packers. Shopfloor workers are guided through the packaging procedure with

step-by-step instructions and real-time feedback. The *third AD* is concerned with this component. As they complete their duties, packers can update the system with pertinent information. As with the other blocks, the components of the Shipment Packaging block are linked to the User Service for authentication, allowing authorised Packers to access and utilise the Packer UI.

The final Shipment Validation block is concerned with the validation and verification of pallets using cameras and AI vision. This block consists of four essential components. The **Camera Gate** is a physical gate equipped with cameras that take pictures of incoming containers. The cameras are plain cameras as decided in the *first AD* and hanging from a suspended gate setup. The images are taken while the pallet stays still (*fifth AD*). These images are then transmitted to the **Pallet Validation Service** for further processing. Using vision AI algorithms, the Pallet Validation Service reads and analyses pallet labelling. It verifies the presence of every pallet and searches for any irregularities or issues. It provides crucial feedback and direction for the next stages. This component will be a combination of CA and OIS regarding the *sixth AD*. If the system experiences outages the *seventh AD* guides to a manual 4-eye principal fallback (two sequential shopfloor workers visually checking). The **Scanner UI** is an interface designed specifically for Forklift Operators. It permits users to scan pallet labels with handheld scanners and receive real-time feedback on the authenticity and placement of the containers. The **KPI Dashboard** contains a comprehensive dashboard for Managers, Customer Managers, Local Support Engineers and Central Support Engineers. This dashboard displays KPIs and metrics associated with scanning and validating pallets. Users can utilise the provided insights to monitor overall performance, identify any obstacles and make the necessary adjustments. As with the other blocks, the components of the Shipment Validation block are securely connected to the User Service for authentication, ensuring safe access and data protection.

The system is planned to be delivered as Software as a Service. This decision (*eighth AD*) is not visualized in Figure 5.2, yet it bears equal significance to the other ADs.

In conclusion, these three building components consisting of numerous interconnected elements create a system that facilitates the scanning, validation and organisation of pallets during warehouse processes. The depicted system satisfies the ADs, thereby enhancing productivity and ensuring a seamless workflow within warehouse operations.

5.3 Component View

The following architecture documentation only considers the shipment validation block. This component view in Figure 5.3 is not yet implemented and serves as a guideline for the target system. This architecture integrates components of CA and OIS as illustrated in Figure 5.3. In the component view, the color scheme utilizes orange to represent the OIS components and green to indicate the CA components. All remaining colors are employed to enhance the clarity of the separation between different components. For simplicity reasons, the SAP and Bosch Identity Manager components are not outlined in these architectures. The Bosch Identity Manager is used to authenticate the users interacting with the shipment validation service. After initially authenticating the session cookie is used during the connection to gain access and communicate with the Apache Webserver.

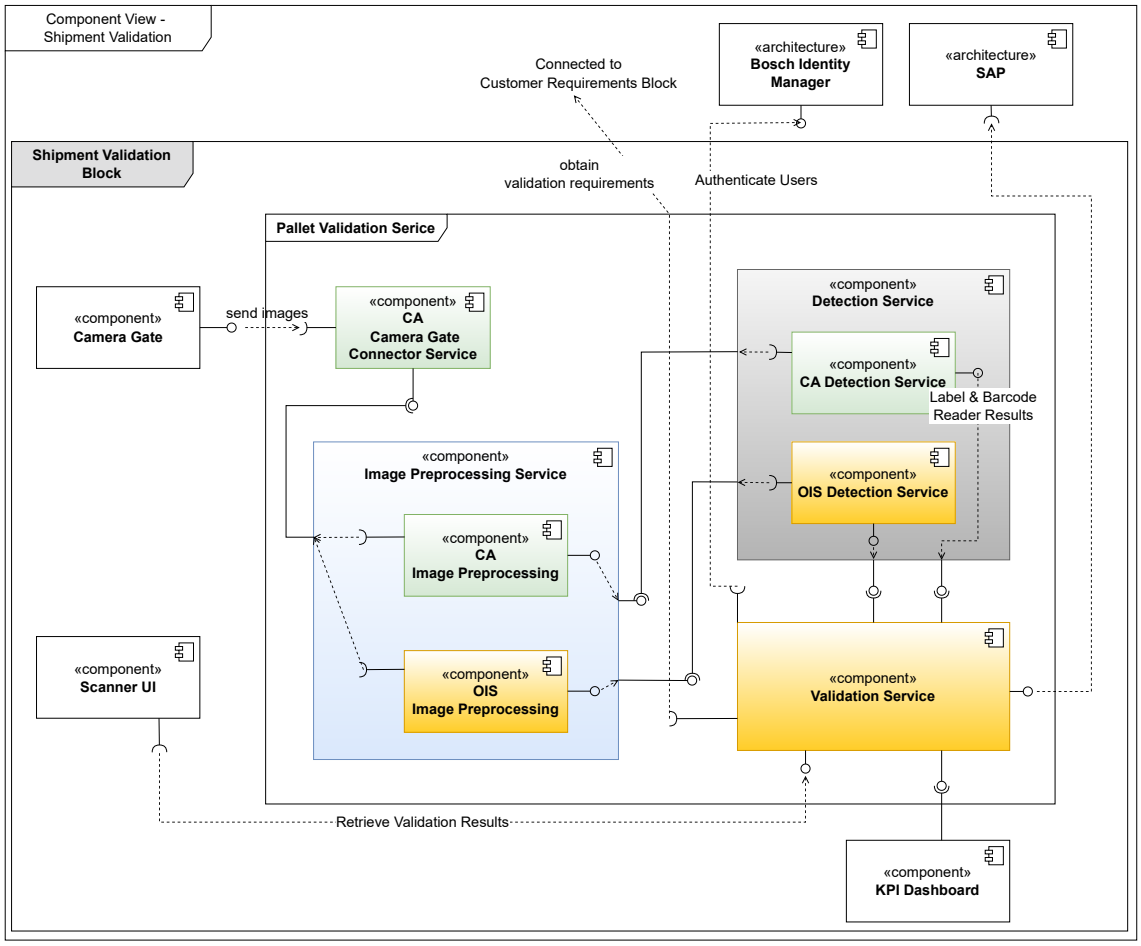


Figure 5.3: Component View - Shipment Validation Overall

The Camera Gate Connector, originating from the CA project, is integrated into the OIS procedure. This particular component serves to establish a connection with the Camera Gate and facilitate the retrieval of pictures. Functioning as an intermediary, it acts as the interface between the Camera Gate Hardware and the Computer situated within the warehouses. Ideally, both devices operate within the same network, enabling seamless data transfer via Ethernet. In the most Bosch plants the Radio-Frequency Identification (RFID) technology is used to identify the pallets and packages. The RFID infrastructure including the computers are used to deploy this service. The Camera Gate Connector Service encompasses an event-handler mechanism designed to process incoming pictures. When the forklift driver initiates the picture-taking process, the event-handler promptly receives the images and subsequently forwards them to the Image Preprocessing Service.

The image preprocessing module incorporates the integration of the CA and OIS image preprocessing components, as depicted in Figure 5.4. The architecture has been intentionally designed to facilitate the preprocessing on the same machine where the Camera Gate Connector is located. The integration enables simultaneous processing of the images. The CA preprocessing component focuses primarily on label detection. This detection is essential for extracting the Handling Unit number from the label, which subsequently plays a crucial role in the validation service to ensure adherence to conformity

rules. On the other hand, the OIS preprocessing component is dedicated to detect various elements of the pallet, such as sticky dots, straps, and the overall structure. Both processes include image correction techniques to enhance the readability of the images for the AI algorithms.

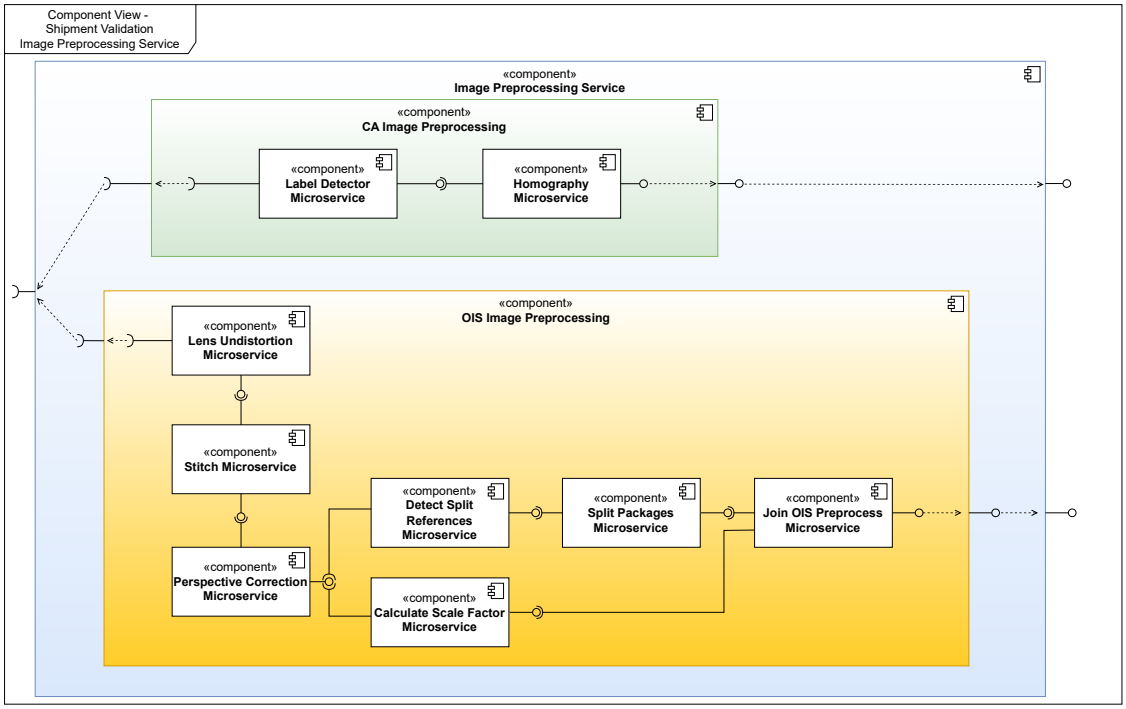


Figure 5.4: Component View - Shipment Validation - Image Preprocessing Service

The decision to perform preprocessing on the on-premises computer is primarily driven by the time it takes to upload images to the cloud. The upload speed is directly dependent on the internet speed, and to mitigate potential delays, smaller preprocessed images and data are prioritized for upload. Once on the cloud, the detection and validation services are executed. The detection service of the CA is responsible for identifying the Handling Unit, a 13-digit number, as mentioned previously. Herefore they use the Optical Character Recognition (OCR) functional service of the Google Cloud. On the other hand, the OIS focuses on detecting items, their positions, and quantities on the pallet. The collaborative relationship between these components is illustrated in Figure 5.5.

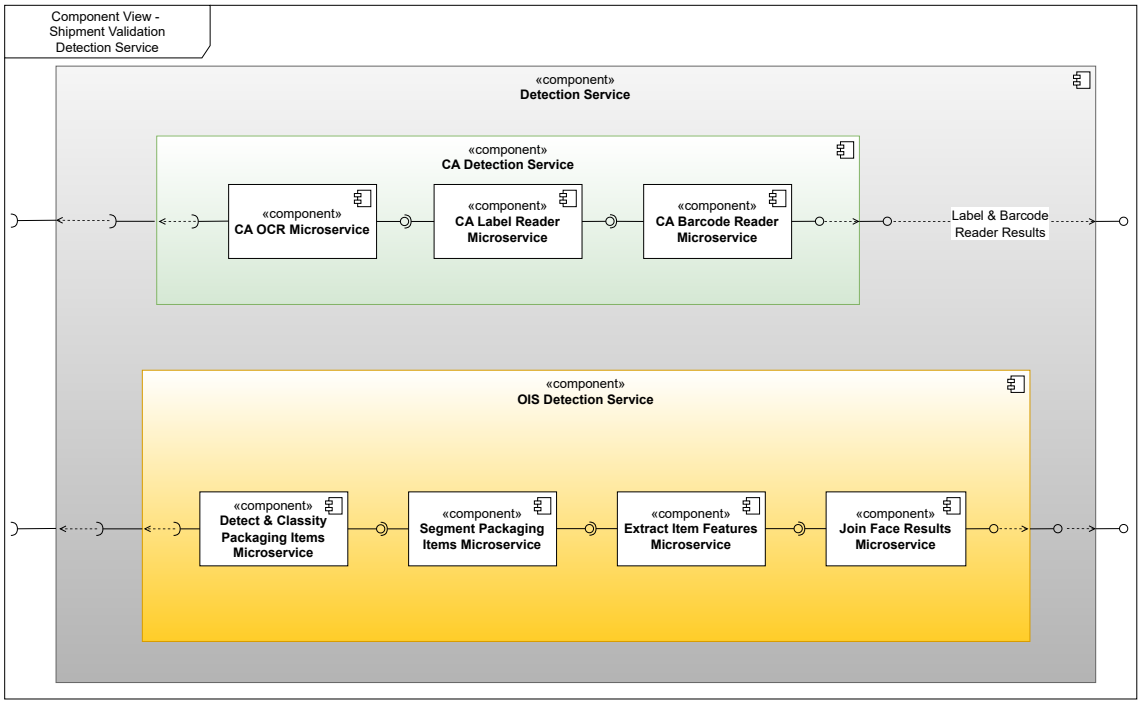


Figure 5.5: Component View - Shipment Validation - Detection Service

The validation service serves as the crucial and final component where data from the preceding modules is consolidated. Initially, the Handling Unit number is employed to retrieve the corresponding conformity rules from the customer requirements block. This retrieved data is then validated against the results generated by the AI-based microservices, which determine the item types, quantities, and positions. This particular component operates within an Apache Webserver, housed within a Docker container. This setup allows for easy access to the results by the Scanner UI and any potential future user interfaces. This architecture is shown in Figure 5.6

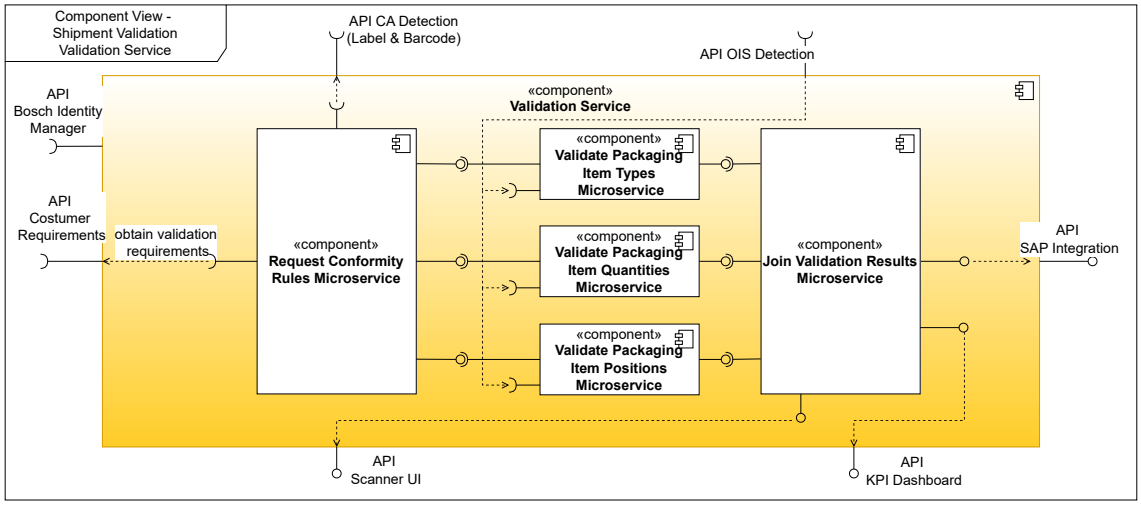


Figure 5.6: Component View - Shipment Validation - Validation Service

5.4 Runtime View

The runtime view of the shipment validation service is depicted in Figure 5.7. The procedure begins with the authentication of the forklift driver through the Bosch Identity Manager. Once authenticated, the camera gate can be triggered to initiate the process. The overall process is self-explanatory, but the preprocessing and detection stages warrant special mention.

During the preprocessing stage, microservices are applied to each picture captured by the four cameras at the Camera Gate. As there are four cameras, the microservice of CA and OIS are scaled accordingly and executed in parallel. Similarly, the OIS detection service follows a similar approach, with processes running for each lateral side of the pallet.

The forklift driver is able to see the results in the Scanner UI which is a web application developed inside OIS.

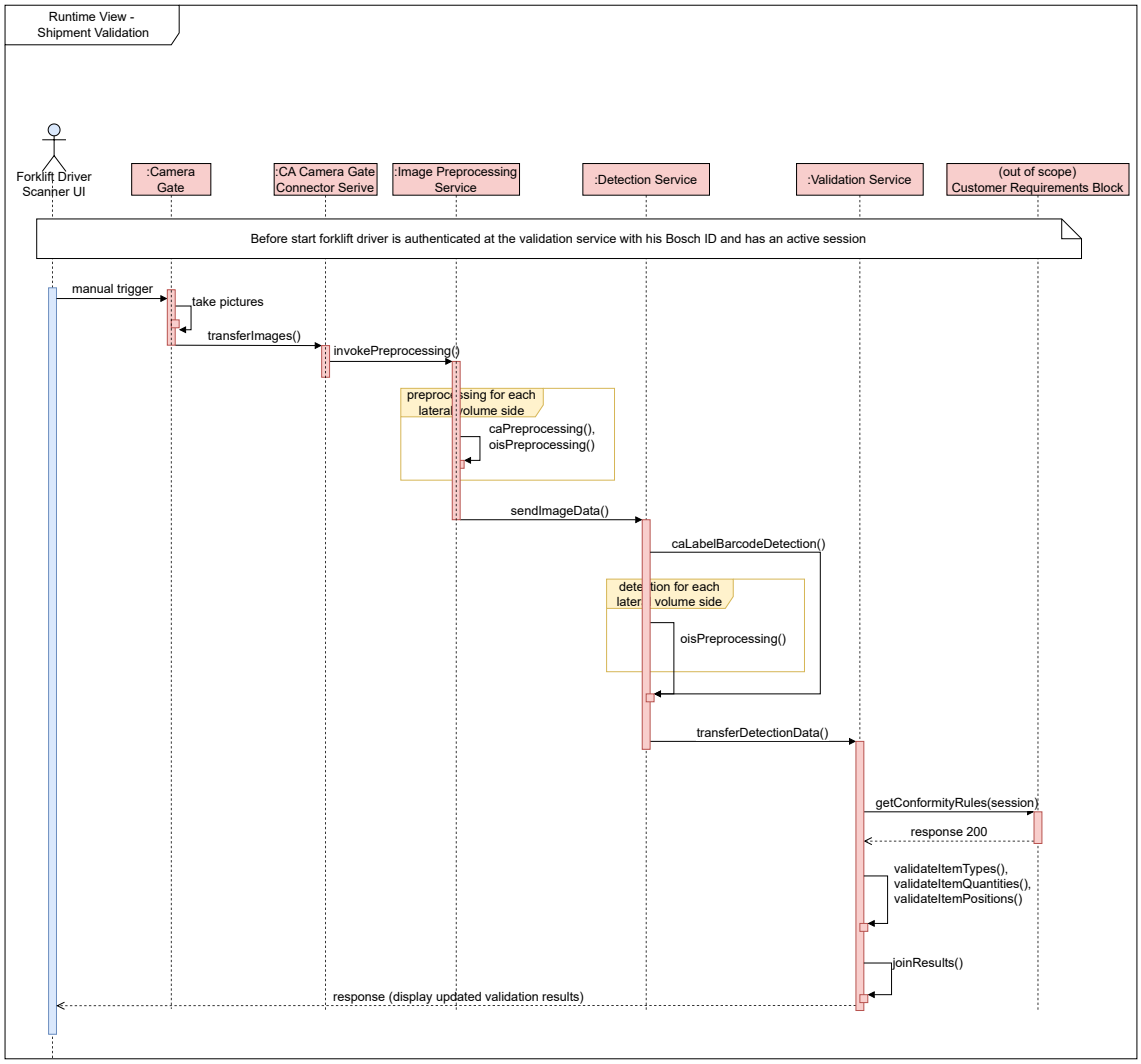


Figure 5.7: Runtime View - Shipment Validation

5.5 Deployment View

Figure 5.8 provides insights into the devices and execution environments where the components will operate, as well as the communication protocols employed for information exchange.

The camera gate will run on a camera controller, which is a pre-existing solution procured externally. The Operating System (OS) for this controller is Windows 10. The controller will be integrated into the same network as the host of the camera gate connector service and the image preprocessing service. Therefore images can be transferred easily through the File Transfer Protocol (FTP).

For the RFID system, a Linux-based operating system is commonly utilized, as it is a widely adopted platform for such applications. Since the gate connector and the image preprocessing will piggyback on this system's device, they also run on a Linux OS.

The camera gate connector service is developed using Kotlin by the CA team. Both preprocessing services are developed in Python and can be easily managed and initialized by the overarching "mother" component.

The detection and validation services are deployed on the Microsoft Azure cloud platform, as the CA team already operates within this environment.

To ensure logical separation and address security considerations, the detection service and validation service are deployed in separate Docker containers.

The Microsoft Azure Kubernetes Service (AKS) cluster is responsible for managing these two nodes. The deployment structure was recommended by the CA team, leveraging AKS for container management. Additionally, AKS offers various advantages, such as the integration of the Azure Kafka messaging backbone. This feature can be utilized to establish a health check and logging system for the microservices.

The validation service operates directly on an Apache Webserver, which possesses the necessary access to the customer requirements, Bosch Identity Manager, and SAP APIs. Communication with these interfaces is facilitated through the use of the HTTPS protocol. Although the specific details of these components are beyond the scope of this context, their involvement and functionality are acknowledged.

The Scanner UI and KPI dashboard are both web applications designed to run on their respective devices. For example, the Scanner UI is intended to be accessed and utilized on the tablet used by the forklift driver.

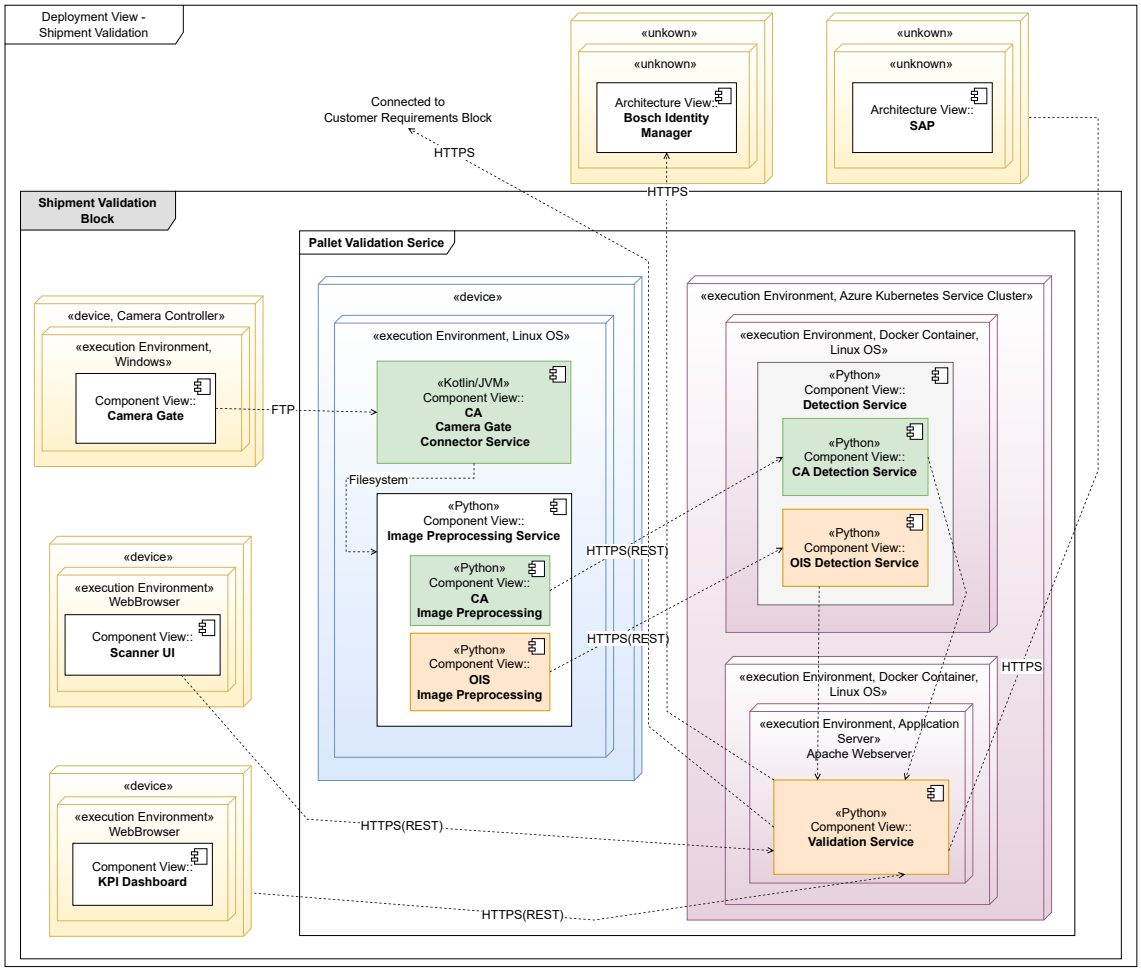


Figure 5.8: Deployment View - Shipment Validation

6 Conclusion

To answer the scientific question of integrating or combining preexisting solutions, the following steps were taken. A comprehensive literature review was conducted to gain knowledge on software architecture approaches, which was then used to decide on a software architecture approach. The review revealed comprehensive insights into various approaches, outlining the necessary phases required to design the corresponding architecture, along with their respective advantages and disadvantages. Additionally, the review evaluated these methodologies based on non-functional requirements, culminating in an architecture ranking matrix. This matrix can serve as a valuable tool for making informed decisions regarding the most suitable architectural approach.

The target system's roadmap included conducting multiple interviews with project teams, developing user stories and analyzing constraints, designing the system context, making ADs and ultimately formulating a target architecture.

The conducted interviews with the project teams provided valuable insights into their respective projects, including their architecture and team capabilities. Substantial amounts of data were gathered during these interviews, which were subsequently analyzed in-depth. Alongside the user stories, a thorough evaluation of the projects was carried out to achieve the best possible outcome.

It was evident that each project exhibited varying levels of maturity and fulfilled user requirements at different extents. Furthermore, the user stories played a crucial role in guiding the ADs, which in turn, were instrumental in constructing the desired target system.

As a consequence of incorporating the non-functional requirements of the users, the ranking matrix derived from the literature review was weighted accordingly to align with their specific needs. The resulting architecture that emerged from this process is the microservices architecture. By employing this method, the overall structure of the target system was systematically designed. As a result, the component, runtime and deployment views were derived from this process.

Utilizing the microservices architecture facilitated the seamless integration of the source projects into the system. Each component within the resulting target architecture is transformed into a service derived from the source projects. This approach ensures a straightforward adaptation of the overall synergy of components, accomplished by offering interfaces for each microservice.

The outcome of my thesis shows an application of the microservice architecture which is the result of the literature review. My work functions as a template or guideline for the VDT. It is explicitly not intended to serve as a comprehensive specification but rather as a concise report of the target architecture.

The incorporation of the ShipQ project in the future is deemed crucial to enhance customer interest. However, due to time constraints and the current engagement of the ShipQ team in transitioning from R3 to S/4HANA, integrating this project before the end of 2023 is unfeasible. Consequently, it is highly recommended to prepare the interface for ShipQ in advance to facilitate its integration at a later stage.

To the best of my research, there is presently a lack of guidelines for choosing a software architecture approach that is aligned with the non-functional requirements of a project. Non-functional requirements are critical aspects of a system's design that impact its overall quality and performance characteristics. Different architecture models have varying strengths and weaknesses when it comes to meeting specific non-functional requirements.

The objective of this work is to provide guidance to developers and software architects in the process of selecting an architecture approach that takes the non-functional requirements into consideration. However, as mentioned in Chapter 2, it is important to note that selecting an approach does not necessarily mean defining "the architecture" itself, but rather establishing a foundational structure for organizing the software. It is recommended to follow the best practices and patterns for example described in [Len12] [Ric15].

As I draw to the culmination of this thesis, I find myself reflecting on the journey of conducting this research within the Bosch enterprise. Throughout this process, I have gained firsthand experience and encountered unique challenges that have shaped my perspective on designing an architecture for application integration. When I commenced my thesis, I was initially assigned to design an integration software architecture and subsequently develop the proposed architecture with the team. Initially, the plan was to analyze four projects. However, it eventually expanded to analyzing seven integration solutions (only five of them were reported in this work), which were onboarded by the project lead. This resulted in a significant amount of time being dedicated to conducting interviews with different teams, analyzing compatibility and coordinating meeting schedules. Owing to multiple delays in the planning process, the team encountered difficulties in developing the integrated target solution as initially intended. This situation impeded the accomplishment of the primary scientific objective, which aimed to compare the old systems with the newly envisioned target system.

Bibliography

[09] *BM-MDA '09: Proceedings of the 1st Workshop on Behaviour Modelling in Model-Driven Architecture*. Enschede, The Netherlands: Association for Computing Machinery, 2009. ISBN: 9781605585031 (cit. on pp. 29, 43).

[AA15] R. Alexander, I. Alexei. "A reactive architecture for cloudbased system engineering". In: (2015) (cit. on pp. 30, 31, 44).

[AA16] S. Ajmal, S. Ali. "Agile-waterfall hybrid model for software development processes". In: *Science International* 28.6 (2016), pp. 5165–5170 (cit. on pp. 24, 25, 37).

[AAE16] N. Alshuqayran, N. Ali, R. Evans. "A Systematic Mapping Study in Microservice Architecture". In: *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*. 2016, pp. 44–51. DOI: 10.1109/SOCA.2016.15 (cit. on p. 22).

[Abe07] A. Abel. *Domain-driven design Quickly*. Lulu, 2007 (cit. on pp. 26, 39).

[AC22] M. Almeida, E. Canedo. "Authentication and Authorization in Microservices Architecture: A Systematic Literature Review". In: *Applied Sciences* 12 (Mar. 2022), p. 3023. DOI: 10.3390/app12063023 (cit. on pp. 22, 35).

[AJW03] G. Anneke, B. W. Jos, B. Wim. *MDA explained the model driven architecture practice and promise*. AddisonWesley Professional, 2003 (cit. on pp. 29, 43, 44).

[AP16] H. Abbas, J. Pooyan. "Microservices architecture enables DevOps: Migration to a cloud-native architecture". In: (2016) (cit. on p. 24).

[ARI15] D. Adjepon-Yamoah, A. Romanovsky, A. Iliasov. "A Reactive Architecture for Cloud-Based System Engineering". In: *Proceedings of the 2015 International Conference on Software and System Process*. ICSSP 2015. Tallinn, Estonia: Association for Computing Machinery, 2015, pp. 77–81. ISBN: 9781450333467. DOI: 10.1145/2785592.2785611. URL: https://doi.org/10.1145/2785592.2785611 (cit. on p. 17).

[Bas] Y. Bassil. "A simulation model for the waterfall software development life cycle". In: () (cit. on pp. 25, 38).

[BD14] R. Bruns, J. Dunkel. "Towards pattern-based architectures for event processing systems". In: *Software: Practice and Experience* 44 (Nov. 2014). DOI: 10.1002/spe.2204 (cit. on pp. 28, 41).

[BHJ16] A. Balalaie, A. Heydarnoori, P. Jamshidi. "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture". In: *IEEE Software* 33.3 (2016), pp. 42–52. DOI: 10.1109/MS.2016.64 (cit. on pp. 23, 24).

[BKO+19] J. Bishung, O. Koyejo, A. Okezie, B. Edosomwan, S. Ani, A. Ibrahim, A. Olushola, I. Odun-Ayo. "A Critical Analysis of Topics in Software Architecture and Design". In: *Advances in Science, Technology and Engineering Systems Journal* 4 (Mar. 2019). DOI: 10.25046/aj040228 (cit. on pp. 14, 21).

[Bos04] J. Bosch. “Architecture-Centric Software Engineering”. In: July 2004, pp. 22–24. ISBN: 978-3-540-22918-6. DOI: 10.1007/978-3-540-28630-1_27 (cit. on pp. 21, 33, 34).

[BQT22] L. Baresi, G. Quattrocchi, D. Tamburri. *Microservice Architecture Practices and Experience: a Focused Look on Docker Configuration Files*. Dec. 2022 (cit. on pp. 22, 34, 35).

[Bro04] A. W. Brown. “Model driven architecture Principles and practice”. In: (2004) (cit. on pp. 29, 42, 43).

[CB12] T. Clark, B. S. Barn. “A Common Basis for Modelling Service-Oriented and Event-Driven Architecture”. In: *Proceedings of the 5th India Software Engineering Conference*. ISEC ’12. Kanpur, India: Association for Computing Machinery, 2012, pp. 23–32. ISBN: 9781450311427. DOI: 10.1145/2134254.2134258. URL: https://doi.org/10.1145/2134254.2134258 (cit. on pp. 28, 41).

[Cha20] W. S. Chao. “Model-Based Systems Engineering Using Structure-Behavior Coalescence Modeling Language”. In: (Sept. 2020) (cit. on pp. 14, 21).

[Clo20] Cloud Native Computing Foundation. *Cloud Native Survey 2020: Containers in production jump 300% from our first survey*. 2020. URL: https://www.cncf.io/blog/2020/11/17/cloud-native-survey-2020-containers-in-production-jump-300-from-our-first-survey/ (cit. on p. 24).

[CSW21] N. Chondamrongkul, J. Sun, I. Warren. “Formal Security Analysis for Software Architecture Design: An Expressive Framework to Emerging Architectural Styles”. In: *Science of Computer Programming* 206 (Mar. 2021), p. 102631. DOI: 10.1016/j.scico.2021.102631 (cit. on p. 21).

[Dav21] C. Dave. “Microservices Software Architecture: A Review”. In: *International Journal for Research in Applied Science and Engineering Technology* 9 (Nov. 2021), pp. 1494–1496. DOI: 10.22214/ijraset.2021.39036 (cit. on pp. 34, 35).

[Dev17] N. M. Devadiga. “Tailoring architecture centric design method with rapid prototyping”. In: (2017) (cit. on pp. 21, 33).

[DL01] J. Daniel, B. Len. “Architecture-centric software project management: A practical guide”. In: (2001) (cit. on pp. 21, 33).

[DL22] K. Dürr, R. Lichtenthäler. “An Evaluation of Modeling Options for Cloud-native Application Architectures to Enable Quality Investigations”. In: *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*. 2022, pp. 297–304. DOI: 10.1109/UCC56403.2022.00053 (cit. on pp. 24, 36).

[DM18] A. Dima, M. A. Maassen. “From Waterfall to Agile software: Development models in the IT sector, 2006 to 2018. Impacts on company management”. In: *Journal of International Studies* 11 (June 2018), pp. 315–326. DOI: 10.14254/2071-8330.2018/11-2/21 (cit. on pp. 25, 37, 38).

[DSM+17] A. Debski, B. Szczepanik, M. Malawski, S. Spahr, D. Muthig. “A scalable, reactive architecture for cloud applications”. In: *IEEE Software* 35.2 (2017), pp. 62–71 (cit. on pp. 30, 31, 44, 45).

[DW05] V. Dheap, P. A. S. Ward. “Event-Driven Response Architecture for Event-Based Computing”. In: *Proceedings of the 2005 Conference of the Centre for Advanced Studies on Collaborative Research*. CASCON ’05. Toranto, Ontario, Canada: IBM Press, 2005, pp. 70–82 (cit. on pp. 28, 41).

[ELV+12] B. Engineer, A. Lombide Carreton, T. Van Cutsem, M. STIJN, W. De Meuter. “A Survey on Reactive Programming”. In: *ACM Computing Surveys* 45 (Jan. 2012). DOI: 10.1145/2501654.2501666 (cit. on p. 31).

[Erl] T. Erl. “Serviceoriented architecture”. In: () (cit. on pp. 27, 40).

[Eva04] E. Evans. *Domaindriven design tackling complexity in the heart of software*. AddisonWesley Professional, 2004 (cit. on pp. 25, 26, 39).

[Eva14] E. Evans. *DomainDriven Design Reference Definitions and Pattern Summaries*. Dog Ear Publishing, 2014 (cit. on pp. 26, 38, 39).

[FYX20] S. Fu, F. Yang, Y. Xiao. “AI Inspired Intelligent Resource Management in Future Wireless Network”. In: *IEEE Access* PP (Jan. 2020), pp. 1–1. DOI: 10.1109/ACCESS.2020.2968554 (cit. on p. 21).

[GPNV02] J. Gustafsson, J. Paakki, L. Nenonen, A. Verkamo. “Architecture-centric software evolution by software metrics and design patterns”. In: Feb. 2002, pp. 108–115. ISBN: 0-7695-1438-3. DOI: 10.1109/CSMR.2002.995795 (cit. on pp. 21, 33, 34).

[GYE22] S. Gottschalk, E. Yigitbas, G. Engels. “Model-driven Continuous Experimentation on Component-based Software Architectures”. In: *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*. 2022, pp. 20–24. DOI: 10.1109/ICSA-C54293.2022.00011 (cit. on pp. 29, 43).

[HH06] W. Harald, R. Harald. “Architectural improvement by use of strategic level domain-driven design”. In: *In Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems* (2006) (cit. on p. 26).

[HO20] E. Hustad, D. Olsen. *Creating a sustainable digital infrastructure: The role of service-oriented architecture*. Dec. 2020. DOI: 10.13140/RG.2.2.34362.00963 (cit. on p. 26).

[IE06] M. Ibrahim, O. Etzion. “Workshop on Event Driven Architecture”. In: *Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications*. OOPSLA ’06. Portland, Oregon, USA: Association for Computing Machinery, 2006, p. 624. ISBN: 159593491X. DOI: 10.1145/1176617.1176639. URL: https://doi.org/10.1145/1176617.1176639 (cit. on pp. 28, 41, 42).

[Jai19] M. Jaiswal. “Software Architecture and Software Design”. In: *SSRN Electronic Journal* (Jan. 2019). DOI: 10.2139/ssrn.3772387 (cit. on p. 21).

[Joh01] D. John. “Model-driven architecture: Vision and dards, and emerging technologies”. In: (2001) (cit. on pp. 29, 43).

[KEP18] A. Kamal, F. Erik, Z. Piyum. *Cloud Native Architectures Design highavailability and costeffective applications for the cloud*. Packt Publishing Ltd, 2018 (cit. on pp. 23, 24).

[KK09] B. Kathryn, L. Kenneth. “Service oriented architecture”. In: (2009) (cit. on pp. 27, 40).

[Kra15] L. Krause. *Microservices: Patterns and Applications: Designing fine-grained services by applying patterns*. 2015 (cit. on pp. 17, 23).

[Law04] W. Lawrence. "Understanding service-oriented architecture". In: *The Architecture Journal* (2004) (cit. on pp. 27, 40).

[Len12] R. K. Len Bass Paul Clements. *Software Architecture in Practice*. 3rd Edition. 2012 (cit. on pp. 17–19, 23, 24, 80).

[Lin17] D. S. Linthicum. "Cloud-Native Applications and Cloud Migration: The Good, the Bad, and the Points Between". In: *IEEE Cloud Computing* 4.5 (2017), pp. 12–14. DOI: 10.1109/MCC.2017.4250932 (cit. on pp. 23, 24, 35, 36).

[MAC+04] G. Miller, S. Ambler, S. Cook, S. Mellor, K. Frank, J. Kern. "Model Driven Architecture: The Realities, a Year Later". In: *Companion to the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications*. OOPSLA '04. Vancouver, BC, CANADA: Association for Computing Machinery, 2004, pp. 138–140. ISBN: 1581138334. DOI: 10.1145/1028664.1028719. URL: https://doi.org/10.1145/1028664.1028719 (cit. on pp. 29, 43).

[MBKN09] H. Mohammad, A. Bavar, N. M. Khashayar, D. Negin. "A brief survey of software architecture concepts and service oriented architecture". In: (2009) (cit. on pp. 27, 41).

[MMCF18] H. Matheus, K. Martin, S. Christina, M. Florian. "Supporting largescale agile development with domaindriven design". In: (2018) (cit. on p. 26).

[MMd18] J. D. Michael, T. " u. o. m. a. c. Michal, future directions. "ACM SIGAPP Applied Computing Review 17 no". In: (2018) (cit. on p. 22).

[MOTG17] M. Mora, R. O'Connor, F. Tsui, J. Gómez. "Design methods for software architectures in the service-oriented computing and cloud paradigms". In: *Software: Practice and Experience* 48 (Nov. 2017). DOI: 10.1002/spe.2547 (cit. on p. 21).

[MS21] S. Mishra, A. Sarkar. "Service-Oriented Architecture for Internet of Things: A Semantic Approach". In: *Journal of King Saud University - Computer and Information Sciences* 34 (Oct. 2021). DOI: 10.1016/j.jksuci.2021.09.024 (cit. on p. 27).

[MW07] P. Mike, V. D. H. Willem-Jan. "Service oriented architectures approaches technologies and research issues". In: (2007) (cit. on pp. 27, 40).

[Nic15] T. Nick. *Patterns principles and practices of domaindriven design*. John Wiley Sons, 2015 (cit. on pp. 25, 26, 38, 39).

[NIG+20] N. Niknejad, W. Ismail, I. Ghani, B. Nazari, M. Bahari, A. Hussin. "Understanding Service-Oriented Architecture (SOA): A systematic literature review and directions for further investigation". In: *Information Systems* 91 (July 2020), p. 101491. DOI: 10.1016/j.is.2020.101491 (cit. on pp. 27, 41).

[NPHK15] D. Nakhaeinia, P. Payeur, T. Hong, B. Karasfi. "A hybrid control architecture for autonomous mobile robot navigation in unknown dynamic environment". In: Aug. 2015. DOI: 10.1109/CoASE.2015.7294274 (cit. on p. 17).

[OHM+15] G. Oscar, C. Harold, V. Mauricio, S. Lorena, C. Rubby, G. Santiago. "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud". In: (2015) (cit. on p. 22).

[Pet17] Q. Peter-Christian. “Understanding cloudnative applications after 10 years of cloud computinga systematic mapping study”. In: (2017) (cit. on p. 24).

[PH07] M. Papazoglou, W.-J. Heuvel. “Service oriented architectures: Approaches, technologies and research issues”. In: *International Journal on very large data bases (VLDB)* 16 (Jan. 2007), pp. 389–415 (cit. on pp. 27, 40).

[PWB09] K. Petersen, C. Wohlin, D. Baca. “The Waterfall Model in Large-Scale Development”. In: *Product-Focused Software Process Improvement*. Ed. by F. Bomarius, M. Oivo, P. Jaring, P. Abrahamsson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 386–400. ISBN: 978-3-642-02152-7 (cit. on pp. 25, 38).

[R G22a] S. R Goniwada. *Cloud Native Architecture and Design: A Handbook for Modern Day Architecture and Design with Enterprise-Grade Examples*. Jan. 2022. ISBN: 978-1-4842-7225-1. DOI: 10.1007/978-1-4842-7226-8 (cit. on p. 35).

[R G22b] S. R Goniwada. “Cloud Native Architecture Principles”. In: Jan. 2022, pp. 55–125. ISBN: 978-1-4842-7225-1. DOI: 10.1007/978-1-4842-7226-8_3 (cit. on pp. 24, 36).

[R G22c] S. R Goniwada. “Introduction to Cloud Native Architecture”. In: Jan. 2022, pp. 3–26. ISBN: 978-1-4842-7225-1. DOI: 10.1007/978-1-4842-7226-8_1 (cit. on p. 36).

[RGKS20] M. Rumez, D. Grimm, R. Kriesten, E. Sax. “An Overview of Automotive Service-Oriented Architectures and Implications for Security Countermeasures”. In: *IEEE Access* 8 (2020), pp. 221852–221870. DOI: 10.1109/ACCESS.2020.3043070 (cit. on p. 27).

[Ric15] M. Richards. *Software Architecture Patterns*. O’Reilly Media, Inc., 2015. ISBN: 9781491925409 (cit. on pp. 18, 80).

[RJ06] L. Robert, E. T. James. “Software architecture-centric methods and agile development”. In: (2006) (cit. on pp. 21, 34).

[RMM16] M. Ronnie, M. Matt, A. Mike. *Microservice architecture: aligning principles*. OReilly Media Inc, 2016 (cit. on p. 22).

[SAAA23] M. Seedat, Q. Abbas, N. Ahmad, A. Amelio. *Systematic Mapping of Monolithic Applications to Microservices Architecture*. Apr. 2023. DOI: 10.22541/au.168110476.68608378/v1 (cit. on p. 22).

[SAM+19] C. Santana, L. Andrade, B. Mello, E. Batista, J. V. Sampaio, C. Prazeres. “A Reliable Architecture Based on Reactive Microservices for IoT Applications”. In: *Proceedings of the 25th Brazillian Symposium on Multimedia and the Web*. WebMedia ’19. Rio de Janeiro, Brazil: Association for Computing Machinery, 2019, pp. 15–19. ISBN: 9781450367639. DOI: 10.1145/3323503.3345027. URL: https://doi.org/10.1145/3323503.3345027 (cit. on pp. 31, 44, 45).

[Sam19] M. Samadi. “Waterative Model: an Integration of the Waterfall and Iterative Software Development Paradigms”. In: X (Aug. 2019), pp. 75–81 (cit. on pp. 25, 37).

[Sie11] J. G. Siek. “Special issue on library-centric software design (LCSD 2006)”. In: 76.4 (Apr. 2011), pp. 225–226. ISSN: 0167-6423 (print), 1872-7964 (electronic) (cit. on p. 21).

[Sob10] S. Sobers. “Reactive Architecture”. In: *ACM SIGGRAPH 2010 Posters*. SIGGRAPH ’10. Los Angeles, California: Association for Computing Machinery, 2010. ISBN: 9781450303934. DOI: 10.1145/1836845.1836889. URL: https://doi.org/10.1145/1836845.1836889 (cit. on pp. 30, 44).

[Sol00] R. Soley. “Model driven architecture”. In: (2000) (cit. on pp. 29, 43).

[SRB19] O. Sievi-Korte, I. Richardson, S. Beecham. “Software architecture design in global software development: An empirical study”. In: *Journal of Systems and Software* 158 (2019), p. 110400. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2019.110400. URL: https://www.sciencedirect.com/science/article/pii/S0164121219301748 (cit. on pp. 14, 21).

[Tel22] T. Telang. “Cloud-Native Application Development”. In: Dec. 2022, pp. 29–54. ISBN: 978-1-4842-8831-3. DOI: 10.1007/978-1-4842-8832-0_2 (cit. on p. 36).

[Tov19a] V. Tovarnitchi. “Designing Distributed, Scalable and Extensible System Using Reactive Architectures”. In: May 2019, pp. 484–488. DOI: 10.1109/CSCS.2019.00088 (cit. on p. 17).

[Tov19b] V. M. Tovarnitchi. “Designing Distributed, Scalable and Extensible System Using Reactive Architectures”. In: *2019 22nd International Conference on Control Systems and Computer Science (CSCS)*. 2019, pp. 484–488. DOI: 10.1109/CSCS.2019.00088 (cit. on pp. 30, 31, 44, 45).

[Ver13] V. Vernon. *Implementing domaindriven design*. AddisonWesley, 2013 (cit. on pp. 25, 26, 39).

[WLS22] M. Waseem, P. Liang, M. Shahin. “Software Architecture Design of Microservices Systems”. PhD thesis. July 2022. DOI: 10.13140/RG.2.2.19569.56168 (cit. on pp. 22, 34, 35).

[Woo21] M. Woodside. “Performance Models of Event-Driven Architectures”. In: *Companion of the ACM/SPEC International Conference on Performance Engineering*. ICPE ’21. Virtual Event, France: Association for Computing Machinery, 2021, pp. 145–149. ISBN: 9781450383318. DOI: 10.1145/3447545.3451203. URL: https://doi.org/10.1145/3447545.3451203 (cit. on pp. 28, 42).

[XYBZ19] F. Xu, F. Yang, S. Bao, C. Zhao. “DQN Inspired Joint Computing and Caching Resource Allocation Approach for Software Defined Information-Centric Internet of Things Network”. In: *IEEE Access* PP (May 2019), pp. 1–1. DOI: 10.1109/ACCESS.2019.2916178 (cit. on p. 21).

All links were last followed on July 21, 2023.

A Project Team Interviews

A.1 ShipSmart

How does live view into the system look like?

A camera mount for the pallet must be properly situated to guarantee adequate image. The LabView application operates directly on the on-site server for rapid data processing. Label validation is critical for ensuring correctness, including checking for precise placement, type, size, silent zone, and even sticky dots. Metadata may be utilized to forecast model setup as well as to assess printer quality. When it comes to barcode grading, it should be at an appropriate level according to International Organization for Standardization (ISO) criteria. If the barcode cannot be read by the reader, it is critical resolving the problem to avoid data processing problems.

Summary of jobs done by the system checks: The demands for a Proof of Concept (PoC) are generally to analyze the readability of the barcode, position of the customer label, quiet zone, number of sticky dots, and number of straps. Following the initial request, more functionalities and jobs were created, such as analyzing the damage to the customer label, label angle, number of customer labels, the radius of sticky dots, and packaging colour.

The PoC scope was effectively accomplished, and it aided in identifying areas for improvement in the packaging process. The team also evaluated the angle and positioning of the customer label, as well as the box colour, to ensure that the product stood out on the shelf.

To make sure that each shipment was properly tagged, the amount of customer labels was taken into consideration together with the sticky dots' radius to gauge how effectively they clung to the packing. How is the system used on a daily basis? The system is still a PoC. Currently, the tests in the operative environment are ongoing.

What are the strengths and weaknesses?

Because of the system's adaptability and flexibility, it is simple to change and adapt to new situations quickly. Regardless of the needs of the individual plants, it is adaptable and can be used. The system offers interface points with other systems, such as , allowing for smooth communication and process simplification. It can also be expanded to include OCR, which can increase the efficiency and accuracy of data gathering. To decrease the chance of errors, the system also has some capability to dynamically identify the pertinent ship-to-party number and apply the proper rule set. Last but not least, the system's potential for expansion via AI-based techniques, such as pattern recognition, may eventually boost its dependability and accuracy.

How is the system setup currently done?

The server is connected to the camera and lights, guaranteeing that the images are delivered and processed instantly. Although it is thought that it is possible to match the ShipSmart requirements with the AutoID common hardware, there may be limitations to take into account regarding plant alignment. The use case's specific requirements should be taken into consideration while defining the hardware, followed by the business case, software requirements, and hardware requirements, in that sequence. It is possible to make efficient use of resources by compiling LabView visual code on the development edition of LabView and using it on the customer site with a runtime-only license.

The system uses Queued Message Handler (QMH) as messaging and LabView libraries for image recognition and middleware. Pallet position is verified using proximity sensors, ensuring accuracy and lowering the possibility of mistakes. An OK or NOK result is obtained and delivered to the cloud within another 3 seconds after the image collection and processing take about 3 seconds. All data is currently kept on-premises, but on different Virtual Machines, and is currently persisted via a batch job in Microsoft SQL once images and capture results have been transferred to an Secure File Transfer Protocol (SFTP) server. All data and information exchanged are safe and secure because the system is situated in a secure area. All of the system's logic, including the vision-related libraries and modules, is housed in LabView. The system's licensed program can only be used once it has been compiled and costs 555 euros for an unlimited license. Other integrations, such as , database integration for client requirements, and database integration for outcomes, will be required at the front- and backends.

What are integration possibilities or options towards a combined target system?

The OIS database acts as a source and input for the validation process and can give the operator a good alternative for output display. The MS SQL database, which is currently being used in the PoC, has a design frontend that enables the specification of customer needs such as labels and positioning. The data structure might need to be expanded, but for the time being, it seems to be a decent fit. Although the system is currently designed for Android, a Windows version would be required for complete functioning. Without much automation for the shipping process itself, ShipQ is primarily a PDF viewer for client needs based on SAP information. Its main purpose is to digitize documents. The LabView technology stack can be used with other tech stacks, such as Python, to give the system flexibility and scalability. Further increasing effectiveness and speed is the QMH messaging system, which acts as an extension point and enables parallel and asynchronous processing.

What are the constraints?

There is a backup strategy ready in case there is a future system outage. This backup strategy entails consulting a paper book that lists customer specifications for how to label things. If an OIS is not accessible, the paper book would be the next best thing to using it for visualization. Currently, the fallback plan requires qualified personnel, but if the technology is implemented more widely, a more detailed manual could be required. The placement of cameras and lights will depend on the specific plant or project because the logistical procedure varies between plants. It might be necessary for cameras to be movable to accommodate various pallet and shipment heights.

The system should be able to operate offline and sync later when a connection is restored because network circumstances at different sites can be unpredictable. Only one side either the long side or both the long and one short side has the label. The Eisenach hardware is incompatible with the ShipSmart system.

The cost of the camera and its software must be established to meet the logistical procedure. It has to be established whether the LabView visual editor is difficult to test and prone to errors. To make the LabView code more reusable across various use cases and test kinds, it might be required to modularize it. Depending on the unique use case, the system's parameters must be adjusted.

What are the team capabilities?

No team, just a single developer.

How does live view into the system look like?

-

A.2 Optical Inspection System

How is the system used on a daily basis?

The RecorderApp can be configured to register inspection stations, including the number and kind of cameras and dock station ports, and it offers thorough inspection data. The Recorder App enables users to start scans from any device with a browser, while the RulesApp lets users create inspection criteria and provides a visual designer for positioning products on pallets, layers, and labels.

The VisionApp employs Django as a software framework, YOLO as an AIframework, and rules that are retrieved via the RulesApp. Dataset generation has been underway for a year, and training is done using the You Only Look Once (YOLO) framework, with a minimum average precision of 90% for package item detection.

The LabelingApp, based on Unity 3D, allows shopfloor operators to access regulations on a read-only basis via a scanner or by typing Handling Unit (HU) numbers, whereas the Braga plant has considerable variability and covers most plants. The program, which works with Windows, also enables users to scan a barcode to receive the appropriate rules and photos.

What are the strengths and weaknesses?

To convert pixels to millimetres, the system uses a height sensor instead of an expensive 3D camera, which lowers hardware costs. Four cameras are currently used by each inspection station, and the equipment is made to be suspended over the loading bay and take photographs diagonally from above the pallet. Modular image processing makes it possible to modify the hardware. The inspection stations' calibration and configuration are done using the Recorder app. Additional cameras might be required to read labels on package sides, which are currently being developed as part of the label reading component. Although spotting straps can be difficult, the system can use many cameras and zoom lenses for greater precision. The Center of Competence team is providing feedback as the project is being developed, and a multi-tenancy idea is anticipated for shared data and instances.

How is the system setup currently done?

The Recorder, Rules, and Vision apps are the three core apps that make up the project. jHipster was used to create the Recorder and Rules apps, which had an Angular front end and a SpringBoot backend. Each application has its own MySQL database.

The YOLO-based image recognition model is used by the Vision app, which was created using Python/Django. Despite being in the PoC stage right now, it was just deployed to a logistics service provider. The Vision app requires near deployment to gate stations (hardware) to reduce latency on image transfers to a cloud solution. The Recorder app displays the results because the Vision app lacks a front interface. Only metadata is immediately transferred to the Recorder app due to performance concerns, while photos are locally saved on the Vision app and periodically sent on a schedule to the Recorder app.

To manage roles and access, Identity Management (IdM) integrates Lightweight Directory Access Protocol (LDAP) with on-premise AD. The Recorder and Rules applications do not yet support single sign-on, which is a concern. SAP data that has been cloned in Dali. Internally, a caching service is utilized due to Dali's poor performance. This link only provides the barest of information: the Bill of Materials (BOM), which comprises packing, pallet, and lid part numbers, as well as ShipToParties for part numbers with 13 digits.

The services are hosted on OpenShift on the Bosch Private Cloud, which was selected because it has experience with other private cloud applications and has fewer security-related overheads. With a login associated with one or more plant IDs, multi-tenancy is accomplished in a single instance. Although IdM roles have been defined, they have not yet been put into practice. This will allow for more precise control over authorisation. The release procedure uses Bitbucket for source control. The test and master branches, respectively, are used to maintain and deploy the test and production environments. Test=0.6.1 and Prod=0.5.0, and Test=0.8.1 and Prod=0.8.1 are the current versions. Red Hat OpenShift serves as the foundation for the build pipeline, although there are certain limitations for full-fledged CI/CD (currently manual trigger), while Jenkins is planned for automation and PR validation. Images from OpenShift are used by the services.

What are integration possibilities or options towards a combined target system?

ShipQ wants to develop a rules app, and they digitize paper by presenting their data in a PDF format. An OIS-like solution for inspecting finished goods on pallets is presently unavailable from AutoID. While ShipSmart now only focuses on one portion of the pallet, its functionality overlaps with that of image recognition technology in the areas of classifying and locating objects. Speed is crucial for the integration of label recognition with CA, which depends on hardware performance. RPC or an SDK/library can be used as the integration technique.

What are the constraints?

Due to hardware overhead, two cameras per pallet side are needed to provide the required image quality to read the label content. The images are then combined. To achieve accurate image capture, operators must position the pallet in a specific spot with a tolerance of +/- 10 cm. Although this may alter with the CoC hardware, sites must be able to suspend hardware over the pallet area. The largest pallets used in the plants they spoke with had a tolerance of +/- 10 cm and measure 1.2 meters in length. The maximum pallet height allowed by heights under 2.4 meters is 1.2 meters, which is the typical height for pallets used in the plants they contacted. Two pallets are normally stacked while loading a truck, hence the maximum height is 2x1.2=2.4 meters.

What are the team capabilities?

It's crucial to assess each team member's skills and establish how they can contribute to the project most effectively to ensure the success of the combined, virtual team.

Three OIS developers and 1.5 developers from a university make up the team. However, as of right now, only 0.5 university-affiliated human resources are available, as one of the developers left the team in February 2023 after completing their MSc dissertation.

The team consists of developers as well as mechanical engineers with substantial experience in designing and constructing physical components. The team also has a CoC AutoID, which can help with asset identification and tracking as it travels through the supply chain. The team may complete a project that satisfies all of its goals and objectives by utilizing the special skills of each team member.

How does live view into the system look like?

-

How is the system used on a daily basis?

Around 1400 reservations are made each month at various high-volume plants in Feuerbach. The emphasis is on making sure that the receipt procedure is efficient, which entails going through a camera gate and having an employee of the office go over the reservation. OCR and barcode labels are both supported by the inbound process, while barcode labels are preferred. For reading labels and barcodes, several engines are employed, including OCR and neural network frameworks. Although the UIs have recently been redesigned in SAP/Landing Portal, not all functionalities have been deployed, and different plants use various UIs.

The booked/failed status, the number of pallets, and SAP information are all displayed in the status UI. The former CA UI, which offers access to imagery and archives, is still used by some websites. In general, the status can only be displayed for items that have arrived; an expected item receipt view is not available. In SAP Fiori, the forklift driver view is better developed and gives the driver access to the booking status of the pallets on their fork. The forklift operator's view displays the number of pallets, green for successful booking, and red for unsuccessful booking.

A.3 Camera Area

What are the strengths and weaknesses?

The system can easily communicate and integrate with other systems because it is a part of the larger InTrack ecosystem. Since it can read labels with text and is hence extendable for sending goods, it is simple to combine with other shipping systems. The system can readily identify commodities, track them, and indicate any problems or anomalies by making use of the label reading capabilities. As a result, it is a very efficient and effective system for managing shipping and logistics. The system's extensibility also allows for customization to fulfil certain business requirements, such as connecting with specialist shipping partners or integrating with proprietary labelling systems.

How is the system setup currently done?

Given that labels can already be read, including their text, the system can be expanded for delivering items. The InTrack team includes approximately 40 developers working on the InTrack platform, which enables more use cases including Dock Yard, Track Trace, etc. This system's frontend and backend are built on InTrack. The frontend will be converted to a single frontend and moved into

SAP Fiori, although it now lacks some functionality like photo archives and scanned photos. The CA front end was cross-platform, including mobile, as it was created in Flutter. Due to the new SAP UI incomplete feature set, there have been several issues.

The systems' hardware is currently specific to this use case, and automation is handled by C/SCL. Currently, only monochrome cameras are used for goods receipt because they are sufficient. For various use situations, InTrack employs multiple hardware configurations. The CA team has also tested colour cameras as part of a PoC for reading other information, such as the colour of straps and pallets, however, this PoC has been put on hold because of this larger project/discussion. One facility in Eisenach (outbound) has a special configuration where they just use the colour photos (similar to Budweis for ShipSmart).

The system features a status tracking service with de-duplication and business rules relating to receipt from recognition; these rules are the same for all plants (so far), according to our research. The system allows numerous plants to connect to SAP, and SAP is used to create multi-tenancy. A user may be given access to one or more plants, and a virtual machine is necessary to upload photographs to the cloud's camera connector.

The Python-based vision parts of this system use PyTorch, TensorFlow, and Azure ML. Kotlin and Spring Boot are utilized for status tracking, together with Grafana, Prometheus, Azure blob/tables, PostgreSQL, Key Vault, and Kafka. There are plans to replace Google Cloud Vision, which is currently utilized for OCR, with an internal alternative. The entire system is deployed on-premises using the Nuclio framework, though a hybrid deployment option is also possible.

This system's integration into InTrack ties some functions to the cloud, making it simpler. Including image upload time, the average duration for image processing is five seconds, and SAP is the bottleneck. Azure DevOps pipelines based on CI/CD and MLOps for training are used in the release process. There are numerous pipelines, however, it is unclear what their average maturity is. User management is dependent on InTrack, and regression of recognition is performed using Allure to learn more about accuracy. During automated testing, SAP is mocked, and SAP is in charge of UI implementation.

The overall environments are Dev, QA, Staging, and Production, and there is a staging environment with QA SAP. On Staging, manual testing is an option, but it's not apparent if a release will need this.

What are integration possibilities or options towards a combined target system?

There are currently no set client criteria for the label recognition system. Instead, the system makes an automatic attempt to interpret labels without any user input. OCR, barcode reading, and machine reading are just a few of the stages in the process, and any one of them could fail. Although outbound processes are not the system's primary focus, it is built to be easily extended, allowing for future upgrades and the addition of new capabilities. The system's business model is to provide a range of services that sites can select from, making it a "plug-and-play" solution that can be tailored to each client's unique requirements.

Four cameras make up the label recognition system, and each of them can recognize up to four different labels. The technology is currently limited by the requirement that labels be applied to the long side of pallets for it to work effectively. Despite this drawback, the label recognition system is a strong and adaptable solution for companies wishing to streamline their supply chain management and logistical procedures.

What are the constraints?

-

What are the team capabilities?

The united virtual team is made up of people with a wide range of abilities and skills. These skills must be taken into consideration to make sure that each team member can properly contribute to the project.

Two data scientists are on the team, and they may contribute to the project by using their knowledge of data analysis and machine learning. A QA professional is also available to guarantee that the finished product upholds high-quality standards. The team also has a frontend developer who can concentrate on the user experience and a DevOps expert who can handle infrastructure and deployment.

Despite being a smaller team, the group still consists of the entire development team, so they have access to a wide range of knowledge and abilities. With this team, businesses can be sure that they have the skills necessary to accomplish their objectives and provide a high-quality product.

A.4 AutoID

The AutoID project is concerned with the hardware of the in-place RFID system. It is not concerned with vision technology, but there are existing smart camera technologies in the open market tackling the same issue. Therefore the AutoID project is also taken into account.

What is the AutoID competence about?

They began using RFID technology and have since diversified their offerings. They worked on code reading with a camera for one to two years. They have about 700 different hardware devices in their lab and a wide range of optical solutions. They routinely evaluate the performance of gear from various vendors in their lab. They especially check devices for network security, compliance and performance in a Bosch environment. Before making purchase selections, they also conduct hardware testing to assess possible manufacturing partners.

What are you currently working on?

To identify pallets while they are in motion, the business uses a 3D, stereo camera. The camera can calculate and recognize a pallet's size, the number of boxes, colours, and pallet type. Future iterations of the device will incorporate damage detection, according to the idea. To extract information from photos, they employ Cognex OCR technology. The camera doesn't take photos; instead, it builds a data structure that represents a point cloud and can be used to extract the required data. Processing is handled by the camera directly, so a backend or middleware system is not required. This hardware-based method is more effective than sending photos via a robust network. The development effort for programming the camera is handled by outside teams.

Updates are required for each installation separately as the project is still in its early phases. Even though updates can be performed remotely, a person must still be present at the device to complete them. For remote bulk updates to be compliant with Bosch Corporate Directives, some work and research are needed.

What do you suggest regarding the hardware setup for the target solution? The business could use both image-based or movement-based pallet recognition systems. The pallet must be still and in a precise location with some tolerance when using the image-based method, which entails transmitting images of the pallet to the backend. Pallets can move close to the scanner for detection and processing with the movement-based solution. Although the precise price difference is not stated, the movement-based solution is more expensive than the image-based one. It is assumed that the more expensive Crosstalk solution—rather than the more recent, unfinished solution—is now in use. As other Cognex technology was not yet available, a previous project for Goods Receipt utilized a straightforward image-based hardware solution.

A.5 ShipQ

ShipQ is an application for sales and warehouse packer teams. It is already in use in several Bosch plants.

How is the system used on a daily basis?

Before, all packing instructions were printed out at the working station on paper. However, a digitalized version that displays packaging instructions is now accessible. With this modification, there is no longer a need for paper-based instructions because accessing and using package instructions is more effective.

The plant then translates and streamlines the customer needs after the sales team receives them, which might be rather large (60–70 pages long). There is a standard template with visualizations accessible, but not all customers use it.

To conduct a search based on criteria, the packer can enter or scan the handling unit, delivery note, or ShipQ. The packer also has the option of searching in any plant, which was created to allow plants to communicate content. A secondary ship location is occasionally connected to the primary facility, which then displays documents from the secondary location.

Scanners that are not special to ShipQ are already in use by the facility, and the information they read is temporary rather than persistent. There are two types of searches: generic and specific. The specific search applies all search parameters, whereas the generic search displays all documents for a plant regardless of other search criteria. For the same client, cross-searching of documents across plants is possible.

The date is emphasized in yellow if a document has recently changed, less than or equal to two weeks ago, to make sure the operator is aware of recent modifications. Although it is the plant's responsibility, it is not always practicable to promptly inform operators or other plants of changes. Additional data that SAP keeps can be viewed by internal or plant management. Document usage metadata, such as the date and how frequently a document was opened, is kept track of. This gives a general idea of whether documents are being used when they ought to be. Theoretically, these data may be used to generate monitoring rules for alerting.

A history or audit of document changes is now provided. This makes it possible for plant managers to monitor any document changes and make sure that packers are using the most recent information.

What are the strengths and the weaknesses?

This product's integration with SAP has given it a huge presence. However, it has already been implemented in a few plants, and by the end of 2023, it is intended to be in use at close to 50 plants. This product's primary use case is for understanding and recording customer requirements. It can also deliver PDF or JPG-based documents for manual review, but again, this is only applicable in the context of its use case.

How is the system setup currently done?

The SAP-integrated Fiori app was created especially for POE=BBM Outbound Shipping SAP. The app's front (Fiori) and backend (ERP) are now both managed by the CI team, and they can be installed on different SAP systems. Despite this effort, the app's distribution continues because it is an already-existing system that is now in use. If the system is down, each plant must have a unique emergency plan, which the ShipQ team is unaware of. There is no user segregation, however, the tenant identity is the SAP Plant code. Data sharing between plants is done to double-check for the same clients.

The software can be set up to eliminate the common prefix in the UI for the operator's convenience, and a JPG file instead of a PDF can be uploaded. Approximately 6,300 documents are currently available in the app. Overall, SAP's Fiori app is a crucial tool for controlling POE and outward shipping, and its features and functionalities are constantly growing.

What are integration possibilities or options towards a combined target system?

The UX survey was completed, and OIS was used to deliver the findings to the sales teams. Although the OIS does provide support for the majority of client use cases, many edge situations from numerous customers in various countries are not handled. Additionally, there are criteria for plant levels that are not covered by the OIS. The frequency of the edge cases is still unclear, and this is a subject that has to be addressed. To ensure that an OIS fulfils the demands of all clients, not just the most frequent ones, it is crucial to take into account the numerous use cases and requirements before adopting one.

B Architecture Decisions

B.1 Camera Hardware

Camera Hardware	
Problem Description	Validation of shipments require optical inspection using the 4-eye principle or image capture through cameras and image recognition technologies.
Option A: New hardware with integrated image processing	
Option Description	• Images are processed internally by the system itself • Validation result is delivered directly by the camera • Cameras are updated with improved recognition models
Option Implications	**Flexibility/Maintainability/Operations:** • No infrastructure required for image recognition software or persistence • Deployments are manual and performed individually (Remotely possible, requires onsite assistance) • Full remote updates require investigations to ensure compliance with Bosch • Uniform combination of new software releases and camera models might be hard to maintain (new camera generation in latest plant vs. old camera model at first PoC plant) → variance expected. • Development of new features and bug fixes are performed by an external company **Features/Process/Cost:** • Pallets will not need to stop in a specific location (processed in motion) • Timeline unclear (currently in development independent of this project) • Higher price • Will cover: dimensions, number of boxes, colours, pallet type, damage • Images are not captured, rather a point cloud data structure (efficient processing, transfer and storage) • Persistence of images and integration with “bigger“ system still requires for claims and archival in the long term
Option B: (Existing) Hardware with CrossTalk support	
Option Description	• CrossTalk is the middleware between RFID readers and the booking system • Packaging is scanned in multiple ways • Additional processing takes place in the CrossTalk middleware which is already installed in 80 locations

Continued on next page

Table B.1 – *continued from previous page*

Option Implications	**Flexibility/Maintainability/Operations:** • Pallets will not need to stop in a specific location (processed in motion) • Lock-in into vendor solution for recognition might limit integration & extensibility options • Limits hardware choices to CrossTalk support • Development of new features and bug fixes are performed by an external company **Features/Process/Cost:** • Integration into SAP already exists • Only metadata is sent to the backend (performance) • Some processing already possible in CrossTalk: barcode reading, OCR, filtering • History of CrossTalk with RFID focus leads to doubt in the applicability of middleware for heavy-duty processing and transport • Around 11 thousand existing installations • Not capable of sending images to the backend • Only basic processing supported: barcode reading, OCR, RFID) • Higher price
Option C: Plain cameras	
Option Description	The camera remains dumb, as it only takes pictures
Option Implications	**Flexibility/Maintainability/Operations:** • Requires a local piece of hardware and software (i.e. gateway) to collect and transfer images • Requires pallet to be still and in a specific location • Different hardware could be leveraged as long as the output image conforms to a pre-defined set of criteria (resolution, angle, sharpness, colour, ...) • New features and bug fixes would be controlled by the core team and much quicker to roll out • No lock-in to a particular vendor **Features/Process/Cost:** • Camera hardware is cheaper and more hardware options • Existing hardware could potentially be reused • Feature set can be tailored and increased as seen fit • Requires common alignment on minimum hardware requirements • Backend has to do all the heavy lifting
Decision	**Option C: Plain cameras**

Continued on next page

Table B.1 – *continued from previous page*

Justification	Option A is not yet available and has a high risk of limiting the system to the vendors' speed of implementation and features. The flexibility is hampered and the high cost of the camera equipment might be a hurdle. Also, the expected variance and difficulties in managing the firmware on the individual cameras can be a very tough challenge in the foreseeable future. Option B is well established but the middleware's history in RFID data which is very small does not bode well to extend it easily for heavy-duty image processing. It faces similar challenges to Option A in terms of flexibility and control of image processing. Thus Option C seems to be the best fit to have a cheap, simple hardware setup for the plants and shift the logic of image processing into a backend that can be adapted more easily without vendor lock-in or tough camera device management. The local gateway is a downside but would likely also be needed in case of options a and b for advanced use cases.

Table B.1: Camera Hardware

B.2 Customer Requirements Definition

Customer Requirements Definition	
Problem Description	Pallet packaging for shipping goods varies based on plant, shipping party and customer requirements. Finalizing these requirements takes several iterations. Currently, plants maintain their own set of requirements as PDF or offline handbooks, which are not machine-readable and have limitations in re-use, comparison and searchability. They are also not suitable for automated validation. Thus it is crucial to have customer requirements for packaging: • in machine-readable and standardized format • that can be used as a validation target for images • that must be flexible to cater to all kinds of edge cases • that must reflect the different dimensions of an instruction (plant, product, ship to party, etc.) • that can be defined with a visual editor
Option A: ShipQ	
Option Description	• currently display PDFs created outside of the system • ShipQ would be extended to include requirements definition • Future ready solution also for S/4HANA
Option Implications	• ShipQ is a pure viewer based on SAP R3 technology and not extensible for customer requirements definition via UI, thus high effort needed • Planned phase-out migration due to S/4HANA requires to recode • Lock-in into SAP world expected and intended
Option B: SAP Co-Innovation: ShipQ successor (Buy + Feature Input)	

Continued on next page

Table B.2 – *continued from previous page*

Option Description	• Portfolio idea by SAP that allows requirements definition in an S4 Fiori app • Under investigation by SAP as generic SAP offering
Option Implications	• Requires close collaboration and commitment to SAP • High risk that feature will not come at all or way too late • Flexibility is very likely limited to generic logistic outbound process that applies to most customers • Bosch specific might be hard to include (edge cases hard to support)
Option C: OIS	
Option Description	Custom development with UI to define customer requirements
Option Implications	• Requires a high maturity of OIS and extension of the UI • By far the most versatile offering with Bosch specifics in place • Integration into third-party systems is possible since the system is fully under Bosch control • Already comes with UI and data format • Current usage is complex to properly define all rules and there is no logical check in place (consistency, correctness and logical sense of combined ruleset) • Development, maintenance and operation of service has to be covered by Bosch or contracted developers (product ownership stays at Bosch)
Option D: New custom development	
Option Description	Specification and development of a Bosch specific customer requirements tool (similar to OIS)
Option Implications	Not further evaluated since it is basically replication Option C which would be a waste.
Option E: Vendor analysis	
Option Description	• Investigation in tools available on the market and buy of service • Assumption: There is a tool available that fits sufficiently
Option Implications	Not further evaluated since it was done prior to OIS development There is currently no tool available on the market that fulfils the Bosch requirements and could be easily integrated to support the intended use cases.
Decision	**Option C: OIS**

Continued on next page

Table B.2 – *continued from previous page*

Justification	Option C (OIS) is the most mature tool that was specifically created to exactly solve the stated problem description. It has a UI and creates a machine-readable format. Options D and E have been discarded due to obvious reasons (see above). Option A is very limited in its scope and requires an upgrade to the new SAP stack. The intended successor by SAP is not confirmed and might not cover Bosch requirements in full without the flexibility for adjustments and an unknown price tag. Thus, Option C is by far the best approach even for the long-term and could theoretically also be extended with ShipQ functionalities.

Table B.2: Customer Requirements Definition

B.3 Customer Requirements Visualization (Packer UI)

Customer Requirements Visualisation (Packer UI)	
Problem Description	The packaging for shipping goods via pallets varies based on plant, shipping party and other factors. Those requirements by the customers must be available during the pallet packaging to the packer to correctly label the package, add straps, sticky dots etc. Currently, every plant maintains its own set of requirements often as PDF or offline handbooks which are used. These are not always following the exact pattern and make it harder for the planner. Thus the requirement is to have customer requirements visualized for packaging: • usable on different end-devices (android tablet, computer, packaging stations) that might not have full internet access • web-based for interoperability • with no room for interpretation and very clearly guides the packer • automatic visualization based on initial scan of product via barcode or manual input
Option A: ShipQ	
Option Description	• Generate instructions via OIS • OIS is extended to generate PDF • Generation and upload of PDF to ShipQ (can also be automated in OIS)
Option Implications	• No changes to ShipQ needed • Ready to go visualization for PDFs • Limited to PDF only and no further improvements possible • Updates always have to come via reupload • No future proof solution due to necessary phaseout of ShipQ • No features such as interactively rotating pallet • Requires implementation of PDF rendering in OIS including a visualization

Continued on next page

Table B.3 – *continued from previous page*

Option B: OIS Labeling App	
Option Description	• OIS current labelling app view is extended to visualize the defined customer requirements (packer view) • Visualization of correct requirements based on: – Scanning of barcode Handind Unit number – manual input inserting part number and ShipQ party
Option Implications	• Role concept and tenancy have to support visualization of image • No PDF rendering necessary (can be done ad-hoc based on data format) • More versatile for different devices and how it is visualized • OIS is required to support many more users and plants (setups) → tenancy must be more sophisticated • Very easy to keep changes in data format and user interface aligned • Functionality to render rules very likely required for specification as well to improve UX • On-premise hosting could make it easier for network connectivity
Option C: Third-Party Packaging Tools	
Option Description	• OIS provides a defined API that can be queried to obtain data • Third-party tools such as Nexeed Package and Control (even though focusing on the individual boxes) could integrate and obtain the shipment requirements • Rendering of the requirements based on the common format
Option Implications	• Each tool has to implement rendering of common data format (customer requirements) • Can be always an additional integration alternative in case “out-of-the-box“ visualization is not needed or sufficient • Currently for the packaging of pallets there are generally no user interfaces in place (just for box packaging) • Off-the-shelf solution would miss a crucial part of the use case and would require additional integration if nothing is in place
Option D: Custom User Interface	
Option Description	• A new frontend is developed specifically for this purpose • The data is directly obtained from OIS via the interface • OIS is persisting the data to service a multi-tenant setup
Option Implications	• Additional effort and overhead needed to provide UI decoupled from any other system • Separate team seems to be overhead and not very efficient • Increase in service landscape does not bring a benefit • Closely overlaps with Option C (third-party) and Option B or E
Option E: Part of validation software (CA, ShipSmart)	

Continued on next page

Table B.3 – *continued from previous page*

Option Description	• The label validation software that shows the result is also used during the packing • UI is implemented e.g. in CA or ShipSmart
Option Implications	• Validation software that displays OK / NOK for forklift drivers can also implement an additional view for packer • Integration and obtaining customer requirements is required for validation in any case so data for rendering is already in place • In case error should be visualized in the long-term rendering of instruction might be beneficial • Decoupling from the customer requirements definition tool can lead to dependencies regarding data format (versioning is important) • Users and tenancy is already required in labelling validation software.
Option F: SAP Fiori Frontend	
Option Description	• Implementation of a packer UI that is available on the shopfloor based on sap Fiori • Requires business function in the background (i.e. OIS) • Similar to goods receiving approach
Option Implications	• Rollout can be done via standard SAP launchpad and hosted on SAP environment that the plant has access to • Does allow for sap operations and hardware for the frontend • Known to the shopfloor and preferred app delivery which would avoid additional tech stack • Implementation speed and capabilities for Fiori apps are limited and generally not as good as custom development • Very fancy implementations such as on-demand rendering might be harder to implement • Unclear if Fiori can support scanners without a full monitor or tablet on a simple scanner.
Option G: Print-Out based on OIS	
Option Description	Not further evaluated since Option B would already be available and presumably higher acceptance
Decision	**Option B: OIS Labelling App**
Justification	Option B seems like a natural fit since a visualization rendering is already implemented. Option A would allow the immediate rollout of OIS generated instruction to users based on PDF but with the downside of manual input to display the correct instructions. Option F would be the logical successor of Option A but might take some time but could be actually the way forward for the ShipQ team.

Table B.3: Customer Requirements Visualization (Packer UI)

B.4 Customer Requirements Fallback

Customer Requirements Fallback	
Problem Description	Shipping of goods is a critical process in the supply chain to ensure delivery to other Bosch plants but to also avoid claims from customers that are waiting for parts. A technical incident with the visualization system of customer requirements (network, software bug, hosting provider issue, latency) should thus not impact the shipping of the goods in a way that it comes to a standstill. The fallback should work: • without an IT system • be easy to use • not rely on another technical system that might face the same problems (network & hosting outage)
Option A: Digital Files: ShipQ (PDF Viewer)	
Option Description	• Customer requirements are always exported as PDFs and uploaded to ShipQ • In case the system is down a fallback to ShipQ can be used to obtain the PDFs
Option Implications	• Only works in case system outage does not also affect ShipQ and network is still intact • Requires constant duplication of data in two systems • Requires ShipQ in the long term as a fallback • Costly since ShipQ infrastructure has to be kept
Option B: Digital Files: Local PC and remote storage	
Option Description	• A local computer has access to remote storage and can view the requirements • Storage can be a file share, network drive, cloud storage etc.
Option Implications	• The fallback only works in case the network and cloud storage is unaffected as well • Remote storage could be more easily filled automatically and centrally, e.g., by OIS in case of an update • Central storage does cost and requires maintenance
Option C: Digital Files: Local PC and storage	
Option Description	• The files are stored within the plants' local infrastructure (disk of a local computer) • A local computer can be used to visualize the files • On-demand a print out is still possible for multiple lines

Continued on next page

Table B.4 – *continued from previous page*

Option Implications	• The fallback works for a large range of outages and is independent of the network • Single computer and storage can lead to bottleneck but prints can be done • Unclear who and when exports of the PDFs happen to the local computer • In case no automatism implemented instruction could be outdated → requires a process • Sharing of files via a single computer is not best practice (which user account) → shared user account also not ideal • Cost-efficient
Option D: Analog File: Printout on paper	
Option Description	• After every customer requirement change a printout is triggered and the handbook is updated • The printout is kept on-site at the plant in case of a system outage • Printout can be duplicated if needed
Option Implications	• Update in OIS would for example trigger a notification to update the handbook • No reliance on any technical system and thus very resilient. • No additional infrastructure, system or tooling is required. • Cost-efficient but neither digital nor eco-friendly
Decision	**Option B: Digital Files: Local PC and remote storage**
Justification	Option B is the best trade-off between providing the fallback in a simple manner yet allowing for central provisioning of the files by the customer requirements service. The downside of a needed network connection from the plant to the data center is accepted since a failure would also impact other more critical systems such as SAP. Option C is not chosen since it is much harder to centrally provide the PDFs automatically and relies on the shopfloor users to download the PDFs. Option D with pre-printing everything every time might lead to a mess or outdated versions and searching the manual documents is not user-friendly. Option A requires a more elaborate integration between OIS and ShipQ to upload the documents, otherwise, a manual upload would have the same processual burden as Option D. However, this option can still be chosen by the shopfloor itself in combination with Option B (at the time of fallback) in case ShipQ is available by just uploading the needed files on-demand.

Table B.4: Customer Requirements Fallback

B.5 Packaging Validation Hardware Setup

Packaging Validation Hardware Setup

Continued on next page

Table B.5 – *continued from previous page*

Problem Description	The finished pallets have to be validated in a timely manner based on image recognition. There are multiple implementations available that are based on images. Usually, multiple images from different angles are analyzed to deduct if the package conforms to a defined rule set such as: • label placement • label existence • number of sticky dots and their placement • number of straps The requirement is to quickly indicate to the forklift driver if the pallet is ready for shipment. Thus the recognition must be: • reliable in different conditions (dust, humidity, temperature, lighting) • fast in terms of validation results (<3 seconds total) • easy to use by means of pallet positioning (margin for error or enforced accurate placement) • fast in terms of usage (driving through the camera gate or quickly positioning and picking it up again) • should support future validation options (e.g. correct boxes, colour, damage etc.)
Option A: Gate (Moving Through)	
Option Description	• Cameras are mounted on a gate and the pallet is pushed through • Images are taken during movement
Option Implications	Not further evaluated since it would require a more sophisticated camera setup. • Requires more lighting and more complex setup • High costs (20k more per gate than any other option) • Requires mobile validation UI / light on forklift
Option B: Suspended Gate (Stopping)	
Option Description	• Cameras are hanging from a suspended setup either from the ceiling or a gate and the pallet has to be positioned underneath within certain limits • Images are taken while the pallet stays still
Option Implications	• Movement is not inhibited and approx. 10 cm margin is possible for the forklift driver • Slower throughput of pallets since the pallet has to be positioned • Feedback can be based on hardware (light)
Option C: Fixed on Ground (Stopping)	
Option Description	• Pallet is positioned at a specific place and cameras are mounted around on the frame around area • Images are taken while the pallet stays still

Continued on next page

Table B.5 – *continued from previous page*

Option Implications	• Movement is inhibited and harder to navigate • Might need more space • Slower throughput of pallets since the pallet has to be positioned
Decision	**Option B: Suspended / Gate (Stopping)**
Justification	Option A was evaluated but the hefty price increase per setup is not worth the added value and increase in speed. Option B is the preferred option since it allows easy movement of the pallet without the risk of damaging hardware or manoeuvring. It is cheaper and a local validation result can be directly fed back to the forklift driver without having a mobile UI.

Table B.5: Packaging Validation Hardware Setup

B.6 Packaging Validation Software

Packaging Validation Software	
Problem Description	The finished pallets have to be validated in a timely manner based on image recognition. There are multiple implementations available that are based on images. The requirement is to quickly indicate to the forklift driver if the pallet is ready for shipment. Thus the recognition must be: • fast in terms of validation results (<3 seconds total) until feedback • easy to use in terms of indicating what is wrong • should support future validation options (e.g. correct boxes, colour, wear and tear) • should have means to integrate towards a customer requirements specification system and SAP • must support proper long-term operations and maintenance of the software stack • must allow for a joint collaboration effort to bundle know-how and resources
Option A: Camera Area	

Continued on next page

Table B.6 – *continued from previous page*

Option Description	• CA is extended with the pallet validation use case since it has the entire chain for image validation already in place. – The extension is built into the CA technical stack and reuses most components if applicable – Reuse of code/knowledge/components from OIS (vision app, results app) and ShipSmart possible • VDT is contributing, developing and integrating the customer "pallet validation module" into the CA backend based on the CA standards • ShipSmart is not further developed and replaced by CA → contribute into development of CA • OIS customer requirements service is further developed (rules app, packer UI, planner UI) and used by CA (API) – Assumption: The customer requirements part of OIS would be independent • OIS pallet validation (vision app, result app) is not further developed and replaced in CA → contribute into the development of CA including reuse of code or components
Option Implications	Interoperability/Flexibility • No pallet validation so far just a more sophisticated environment (infrastructure) for goods receipt • Extension of CA backend with additional services possible • SAP stack limits possibilities and full integration into CA backend might come with processual limitations (enforced CI/CD, technology, integrations etc.) • VDT has to adhere to CA standards in terms of development, contribution and tool stack Operations/Maintenance / Rollout • Operations and rollout could piggyback on SAP rollouts for other use cases • Operations Team of CA could do technical and application infrastructure operations (split) while VDT support the application logic • VDT has to onboard themselves into InTrack processes Cost/Vendor • Cost separation for use cases will be extremely difficult but not needed since in the same teams portfolio (InTrack) Performance/Reliability/Security • Depending on setup reliability and security are in shared responsibility between InTrack Team and VDT with the governance at InTrack • Problems of existing InTrack services can impact label validation (shared services/communication/infrastructure) • Currently, there is no edge validation in place thus round-trip to send images, validate them would be likely too slow → Likely requires additional at the plant
Option B: OIS (Vision App + Result App)	

Continued on next page

Table B.6 – *continued from previous page*

Option Description	• OIS software is hosted as-is in the bosch private cloud including an edge for the vision app to collect and process images • The VDT operates and develops OIS further • ShipSmart is not further developed and replaced by OIS → contribute to development of OIS • CA focus is remaining on the goods received use case and does not develop pallet validation/goods sent use case capabilities. → contribute to development of OIS
Option Implications	Interoperability/Flexibility • Tech-stack allows for further extension and integration of other services • Custom development is flexible in change implementation Operations/Maintenance/Rollout • VDT would have to do the entire operation and maintenance of application software – If not desired, an operations team has to be found that takes over those tasks • Rollout in plants would be a separate process and could not be bundled with the goods received use case • Full flexibility in how to do operations, tooling and development workflows • Plant is more cumbersome to rollout Cost/Vendor • No license costs but infrastructure and manpower • Cost separation and charging can be implemented more easily than in other options if needed Performance/Reliability/Security • Validation is performant due to setup with edge processing of the images and does not require a change in the design • No dependencies to other use cases
Option C: Combination of OIS and CA	
Option Description	• The OIS software stack is combined with the CA stack • This is similar to Option A, however, the OIS services are fairly independent and the apps are used as-is and are not tightly coupled into the CA ecosystem (e.g. Nuclio, Kafka or similar). – Assumption: Also the customer requirements part of OIS would be merged to CA

Continued on next page

Table B.6 – *continued from previous page*

Option Implications	Interoperability/Flexibility • The OIS backend could use CA services where needed to extend functionality (e.g. SAP integration) • Flexibility of hosting environment allows for freedom in technology choice • No coupling between CA backend and OIS on application level allows for decoupled development • Long-term more and more integration could be done on a case-by-case basis Operations/Maintenance/Rollout • Technical operations could be covered by CA (e.g. joint runtime) • Application infrastructure and application itself operation would be in responsibility of VDT • Operations and rollout could piggyback on SAP rollouts for other use cases (InTrack) • Reuse of on-premise footprint within a plant () for ships received use case possible (network connectivity is solved) – Local plant s will be a pain for operations (follow-up decision) Cost/Vendor • Cost split might be tough to separate between CA and OIS components • Lock-in into Azure provider in contrast to OIS Bosch Private Cloud that is a significant business risk for the plants Performance/Reliability/Security • Depending on setup reliability and security are in shared responsibility between InTrack Team and VDT • Problems of existing InTrack services can impact label validation (shared services/communication/infrastructure)
Option D: ShipSmart	
Option Description	• On-Site installation with central binary with the latest release for self-installation • VDT would develop and operate the validation suite of ShipSmart integrating e.g. the customer requirements definition and rolling out the software • ShipSmart would be deployed in the plant on a • Label validation happens on the edge

Continued on next page

Table B.6 – *continued from previous page*

Option Implications	Interoperability/Flexibility • The LabView technology hampers the interoperability in terms of clearly defined interfaces, versatile runtime environment or modular contributions • Lock-in into proprietary graphical programming language (extensibility possible across languages but difficult to maintain) • Hard to centralize individual parts such as the validation connecting many tenants • Tough to implement advanced features such as user management, SSO via OAuth etc. without the use of custom web services Operations/Maintenance/Rollout • The maintenance of the solution across many installations is likely very cumbersome • CI/CD tooling is underdeveloped and not comparable with other programming frameworks • Rollout in many plants would likely be done with individual installations • Operations/Maintenance in full responsibility of VDT which would require long-term ops team • Limited skill availability for LabView within the team Cost/Vendor • License fee required for every installation
Decision	**Option C: Combination of OIS and CA**
Justification	Option C provides a very fast way forward by re-using the already implemented pallet validation by OIS and its interfaces while bundling resources in terms of operations, environment setup, development and potential mid- to long-term benefits of integrating more CA services in OIS and vice versa. The bundling of the closely related use cases is beneficial for the customer since rollout, camera gate and infrastructure could support goods received and sent use case together with the same infrastructure and provider. It also allows for better collaboration within development since the technical onboarding is reduced. Option A would require OIS and ShipSmart developers to learn a new framework and rework more things from scratch costing time and wasting resources. Option B would be also a valid option but the separate setup and sole usage of OIS comes with downsides: The image processing infrastructure would be duplicated for both use cases, the maintenance and operations would have to be done by VDT or a operations providers have to be found instead of relying on InTrack. The customer would require a separate rollout for goods sent and goods received.

Table B.6: Packaging Validation Software

B.7 Packaging Validation Fallback

Packaging Validation Fallback	
Problem Description	In case the intended system for validation fails different options for a fallback exist. We can assume there are multiple reasons for a validation outage such as: • customer requirements for validation input are not available (e.g. OIS is down) • camera or other hardware is damaged • the network connectivity is impaired • the validation software has a bug or is not available • there is a major outage at a provider or hosting environment
Option A: Fallback system in different environment	
Option Description	• The backend is unavailable in its primary location • The fallback means a redundant environment is activated that is for example hosted in a different cloud provider region/Bosch data center or provider
Option Implications	• Would only be beneficial if the problem has its origin in backend itself or the hosting provider • Does not cover any plant-specific outage or connectivity issues • Requires a lot of processual and technical overhead and cost based on how mature the fallback implementation is • Any manual fallback (e.g. reinstalling the system) is likely similar in time in just restoring the primary environment back to a working state
Option B: Local validation on gateway	
Option Description	• The validation can be executed in a degraded state directly in the plant without the main system being available • Edge hardware does continue to process and once the full system is online synch the results and images
Option Implications	• Requires logic and processing power on the edge (directly within the plant) • Architecture design has to be built upfront to support this use case (e.g. buffering results on the edge and uploading them later → resiliency) • Covers most outage scenarios except plant-specific hosting environment • In case camera hardware or local setup is affected an outage still occurs • Would require a more powerful edge environment that is not relying on Bosch data centers (otherwise similar outages could occur)
Option C: Manual 4-eye principle	

Continued on next page

Table B.7 – *continued from previous page*

Option Description	• The shipment is validated manually in a 4-eye principle approach (two sequential shopfloor workers) as currently done • The validation input comes either from the customer requirements fallback option or from the customer requirements visualization system if still available
Option Implications	• Does not require any technical implementation • Very cost efficient and no overhead to implement since already done • Switch to manual process is more error-prone especially if no dedicated workers are usually doing it (no experience anymore and short staffed) • Can cover any kind of failure scenario
Decision	**Option C: Manual 4-eye principle**
Justification	The manual 4-eye principle is well established and the current approach. It does cover any kind of outage and is the most cost-efficient and easiest approach to implement. Options A and B come immediately with an intense technical complexity while still not covering all possible outages (data center, plant network, etc.). Option B could be potentially a feature in the mid- to long-term depending on the resulting integration architecture but the benefit vs. the required implementation effort is questionable and would require a revisit of this decision.

Table B.7: Packaging Validation Fallback

B.8 Delivery Model Backend

Delivery Model Backend	
Problem Description	This decision will evaluate different delivery options for the planned solution, meaning how will the solution be sold and delivered to the customer. This is a fundamental decision when developing a new solution since it affects a lot of aspects of the solution and has a lot of implications. Those implications will be compared in the following. An important requirement to fully evaluate the impact is the planned business model to monetize the solution.
Option A: Fully managed services offering Software-as-a-Service (SaaS)	

Continued on next page

Table B.8 – *continued from previous page*

Option Description	• product team will host the solution for the customers in owned infrastructure/cloud account • hosting for all customers by the product team required • operations required by the product team • Customers do not get any specific developments or features and can only use built-in configuration options • Integration into SAP systems is the responsibility of the product team
Option Implications	Quality and Performance of service • full control of product quality and performance • immediate customer feedback and response Operations effort • easier to maintain as in control of the project team • no effort in adapting solutions to different platforms • flexibility and control for updates and changes Cost • infrastructure and operations costs charged to product team → covered by customer payments • synergies in costs as multiple customers might share a single infrastructure and an operations team • pay-per-use models are easier to offer (in contrast to licensing when self-hosted at customer site) Security • shared responsibility of security is heavily tilted towards the product team • Single security approach sufficient
Option B: Delivery to customer for self operation	
Option Description	• Customer operates the solution by himself on an infrastructure of his choice • No access of product team to customer hosting • Handover just in terms of software artefact and installation instructions • Update is the responsibility of the customer • On-premise gateway for image transfer and camera setup is the responsibility of the customer • Integration into SAP systems is the responsibility of the customer

Continued on next page

Table B.8 – *continued from previous page*

Option Implications	Quality and performance of service • negative effect on quality and performance of product possible, as no control over the environment • limited feedback from customers and slow response Operations effort • separate hosting for each customer - maintenance effort for each deployment at customer • no daily IT operations effort for the product team • additional effort in documentation and customer support • updates and changes in responsibility of the customer - multiple versions might be required to be supported → enforcement of updates via contracting • troubleshooting much more difficult due to diverging platforms • no control over quality and performance of the product • customer can define availability, maintenance window and update schedule by himself Cost • infrastructure and operations costs charged to customer directly • higher flexibility for customers in terms of hosting and costs • pay-per-use difficult to realize (delivery of metrics to product team). Different pricing models required (e.g. licensing per hosting,...) • Pay for support possible Security responsibility of security is tilted towards the customer
Option C: Provided Customer Deployment With Self-Operation	
Option Description	• Product team could bootstrap cloud deployment for a single customer • Customer owns and has access to the cloud deployment and is responsible for the operation • product team could have operations and troubleshooting access • on-premise gateway for image transfer and camera setup is the responsibility of the customer • Integration into SAP systems is the responsibility of the customer with support • similar tech stack between customers

Continued on next page

Table B.8 – *continued from previous page*

Option Implications	Quality and Performance of service • negative effect on quality and performance of product possible, as not in control of daily operations • limited feedback from customers and slow response Operations effort • deviations in customer environments (configuration) → more difficult troubleshooting • no daily IT operations effort for the product team • additional effort in the documentation and customer support • updates and changes in responsibility of the customer - multiple versions might be required to be supported • customer can define availability, maintenance window, update schedule by himself • easier to troubleshoot for product team (infrastructure equal for all hostings, operation on behalf of the customer easier to realize) • customer can benefit from deployment templates Cost • Infrastructure and operations costs are the responsibility of the customer • no costs for infrastructure charged to product tea • pay-per-use easy to realize, different licensing models possible Security is a shared responsibility of security between the customer and the product team (product team might deliver infrastructure definitions and blueprint)
Decision	**Option A: Fully managed services offering SaaS**
Justification	Option A facilitates CI/CD on the cloud premises. The whole development, stage and production environment are easily managed. The maintenance and adaptability to other platforms are beneficial for operations. On-premise gateway for image transfer, image preprocessing and camera setup could be in responsibility of the plant/customer. This necessitates on-premise installation and processing of the software.

Table B.8: Delivery Model Backend

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that 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 copies.

---

place, date, signature