TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ANTTI PARKKONEN

ADOPTION OF MICROSERVICES IN INDUSTRIAL INFORMATION SYSTEMS: A SYSTEMATIC LITERATURE REVIEW

Master’s thesis

Examiner: Assisstant professor David Hästbacka

The examiner and topic of the thesis were approved on 29 August 2018

ABSTRACT

ANTTI PARKKONEN: Adoption of microservices in industrial information systems: a systematic literature review
Tampere University of Technology
Master of Science Thesis, 50 pages
November 2018
Master's degree program in Automation engineering
Major: Information systems in automation
Examiner: Assisstant professor David Hästbacka

Keywords: microservices, industrial information systems, systematic literature review, SOA

The internet, digitalization and globalization have transformed customer expectations and the way business is done. Product life cycles have shortened, products need to be customizable, and the production needs to be scalable. These changes reflect also to the industrial operations. Quick technological advancements have increased the role of software in industrial facilities. The software in use has to enable untraditional flexibility, interoperability and scalability.

Microservices based architecture has been seen as the state of the art way for developing flexible, interoperable and scalable software. Microservices have been applied to cloud native applications for consumers with enormous success. The goal of this thesis is to analyze how to adopt microservices to indstrial information systems. General information and characteristics of microservices are provided as background information and a systematic literature review is conducted to answer the research problem. Material for the systematic literature review was found from multiple digital libraries and 17 scientific papers matched the set inclusion cirteria. The material was then analyzed with an extensively documentated method.

The thesis brought together the available publications on the topic. Guidelines for adopting microservices to industrial information systems were derived based on the analysis. Real time applications need special attention when using microservices architecture, the developers need to use proper tools for the tasks, and the developers and users need to be properly introduced to service-oriented systems. Based on this thesis microservices seems like a suitable approach for developing flexible industrial information systems, which satisfy the new business requirements.

TIIVISTELMÄ

ANTTI PARKKONEN: Mikropalvelupohjaisen arkkitehtuurin soveltaminen teollisuuden tietojärjestelmissä: systemaattinen kirjallisuuskatsaus
Tampereen teknillinen yliopisto
Diplomityö, 50 sivua
Marraskuu 2018
Automaatiotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Automaation tietotekniikka
Tarkastaja: Assistant Professor David Hästbacka

Avainsanat: mikropalveluarkkitehtuuri, teolliset tietojärjestelmät, systemaattinen kirjallisuuskatsaus, SOA

Internet, digitalisaatio ja globalisaatio ovat mullistaneet kuluttajien odotuksia ja muokanneet yritysten liiketoimintamalleja. Nämä muutokset heijastuvat myös teollisuuden toimintatapoihin. Teollisuuden on kyettävä toimittamaan erilaisia tuotteita lyhyemmällä syklillä, tuotteita on voitava mukauttaa ja tuotannon koon on oltava säädeltävissä. Teknologian nopea kehittyminen on korostanut ohjelmistojen roolia teollisuusjärjestelmissä. Liiketoiminnan muuttuneet vaatimukset vaikuttavat myös teollisuudessa käytettävien ohjelmistojen kehittämiseen. Ohjelmiston on mahdollistettava joustavuus, yhteentoimivuus ja skaalautuvuus.

Mikropalveluarkkitehtuuri on mahdollistanut joustavien, yhteentoimivien ja skaalautuvien ohjelmistojen kehittämisen. Mikropalveluarkkitehtuuria on sovellettu menestyksekkäästi erityisesti kuluttajille suunnatuissa pilvipalveluissa. Tässä diplomityössä perehdytään mikropalveluarkkitehtuurin yleisiin ominaisuuksiin ja suoritetaan systemaattinen kirjallisuuskatsaus. Tehdyn kirjallisuuskatsauksen tarkoituksena on analysoida, miten mikropalvelupohjaista arkkitehtuuria voidaan soveltaa teollisuuden tietojärjestelmissä. Kirjallisuuskatsauksen aineistoksi valittiin kaikki 17 asetetut hakuehdot täyttänyttä tietellistä julkaisua. Kirjallisuuskatsauksen aineisto haettiin digitaalisista kirjastoista. Aineisto käsiteltiin systemaattisesti tarkkaan dokumentoidulla menetelmällä.

Tutkimus kokosi yhteen aihetta käsittelevät tieteelliset julkaisut. Aineiston analyysin perusteella muodostettiin suositukset mikropalveluarkkitehtuurin soveltamiseen teollisuuden tietojärjestelmissä. Mikropalveluarkkitehtuurin soveltamisessa tulee huomioida erityisesti reaaliaikaisten sovellusten toteuttaminen, oikeiden työkalujen käyttö sekä kehittäjien ja palvelujen käyttäjien perehdyttäminen mikropalvelupohjaisiin järjestelmiin. Tutkimuksen perusteella mikropalvelut vaikuttavat sopivalta tavalta toteuttaa joustavampia ohjelmistoja, jotka vastaavat teollisuuden muuttuneisiin vaatimuksiin.

CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

RAMI4.0	Reference architecture model for Industry 4.0
IIRA	Industrial IOT Reference Architecture
ICPS	Industrial Cyber-Physical Systems
SOA	Service-oriented architecture
W3C	World wide web consortium
REST	Representational state transfer
DevOps	Development operations
JSON	Javascript object notation
SLR	Systematic literature review

1. INTRODUCTION

1.1 Background

Business organizations are under a lot of pressure to respond agilely to quickly changing requirements and demand. Globalization and the Internet have made competition more fierce. Enterprises must respond to competitors' actions, and align their business processes and products with the current overall situation ever faster. Due to this fierce competition product life cycles have shortened. This dynamic environment has driven enterprises from a traditional vertical business divisions to horizontal business process oriented structures and towards an ecosystem paradigm, where different business functions work seamlessly together and provide services for each other. [17] Companies need a complete integrated environment that covers the whole product life-cycle including design, testing, manufacturing, services and related business activities.

As well as other business functions, the industrial production systems are facing the shift in paradigm. Devices on the production floor level are getting smarter and smarter. More information is being collected and this has spurred multiple different suggestions for new paradigms and architectures. Since its introduction the ISA-95 enterprise control systems integration standard has lead the way in the vertical integration of production floor level devices to manufacturing execution systems and the enterprise resource planning level [59]. However the ISA-95 standard does not consider horizontal or diagonal integration and its development started already two decades ago in a time in which the processing capabilities of field devices was a fraction of the current capabilities.

1.2 New paradigms in industrial information systems

New movements like Industry 4.0 [32] are trying to tackle the emerging challenges and predict the future of industrial systems apriori. The goal of Industry 4.0 is to leverage digitalization and ever developing software capabilities in the integration of different parts of the industrial systems, and in the development of new digital services and business models[72]. The movement has come up with a reference model for industry 4.0 (RAMI 4.0) [31], which maps the technological and economical landscape in to a three dimensional cube with 6 layers. The same principles, which the Industry 4.0 movement highlights, have been mentioned with slightly differing emphasis under multiple different names. While Industry 4.0 is an European movement, the American equivalent is the Industrial Internet movement developed by the Industrial internet consortium. Its goals are to transform the industry and add business and social value through advanced data analysis and intelligent operations with internet capable connected devices. The Industrial internet consortium has also come up with their own industrial internet reference architecture (IIRA). [43] Both of these movements fall under the concept of industrial cyber-physical systems (iCPS). ICPS has been used as a hypernym to cover both Industry 4.0 and Industrial internet, as well as other similar movements, in multiple different publications such as [71], [13], [12], [14], and [42].

In the center of these movements is the idea that information gathering, processing and networking capabilities can be fitted to almost every device in the production environment, and that all physical entities in the process have a virtual twin which reflects the physical state of the entity. High computing and storage capabilities are essential to gain advantage of large volumes of processed information and the virtualized objects and some information just requires fast and real-time computation on a small data set. This has lead to dividing computation in 3 different layers, the edge layer, fog layer and cloud layer. These 3 layers are depicted in figure 1.1. [33]

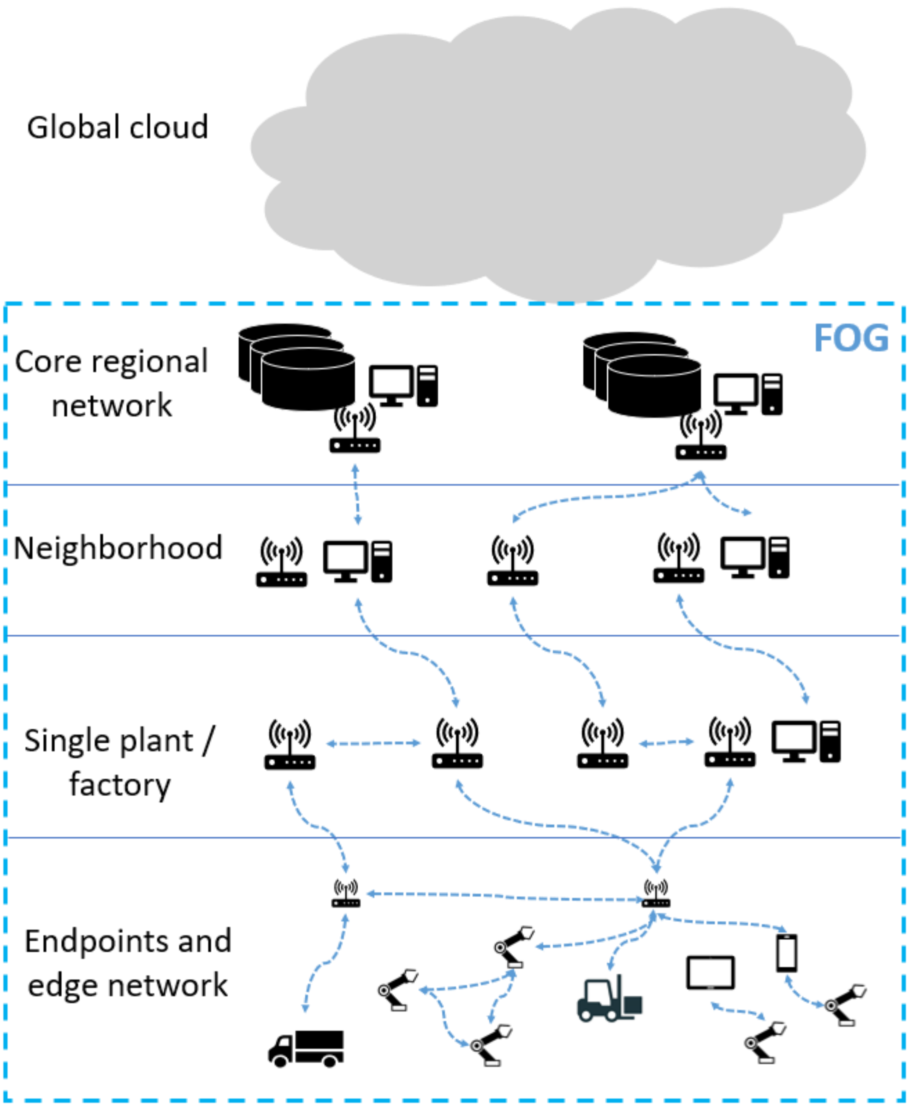

Figure 1.1. *Different computing domains in the future industrial information systems [33].*

Each computation layer comes with specific properties. The edge layer's responsibility is to provide application intelligence to the field devices. It literally exists on the edge devices i.e. the PLCs, terminals and mobile devices in the shop floor level. On the edge layer the edge devices perform computational tasks that require real-time capabilities and low latencies, but the devices have limited storage and computing capabilities. The edge temporarily stores information, preprocesses it and forwards it through the core network to persistent data storages for further analysis and statistics. On the edge all processing and information is in close proximity to its users and distributed along the edge devices. [61][23]

Cloud is the exact opposite of the edge layer. Even though the cloud is often seen as ubiquitous computing it is a centralized computing model, where all processing happens in huge data centers. These data centers have huge storages and high computing capabilities. The cloud also offers redundancy as the datacenters might be fully backed up and geographically distributed across the globe to minimize the effect of local failures. Due to cloud's huge computing capabilities it has been seen as a good way to process large datasets. However all information needs to be transmitted through internet to the centralized cloud for computations. This limits the real-time capabilities that cloud computing offers and puts sensitive information at risk when transporting it off-site [23].

Fog computing extends the concepts of edge and cloud computing by pushing the capabilities of cloud computing closer to the edge of the network. Fog computing is a distributed computing model, where heterogeneous devices at the edge or close to the edge provide elastic computation, storage and networking services in collaboration with each other. The concept is specifically designed to address the challenges of relying solely on edge computing or cloud computing. The vision is that fog would add a resource rich layer of distributed nodes that provide reliable, secure and high performance computing with low latencies. [70] The OpenFog consortium has even envisioned fog computing as a way of creating system level computing, where resources are shared with all layers from cloud to edge and across multiple protocols. The resources would be consumed on demand and taking in to account the quality of service requirements set for the specific task. [28]

This is particularly challenging from the industrial information systems point of view. Traditionally industrial software has been immutable. Software has been designed for a specific purpose and configured to fit the needs of the particular production line or process. As long as the real-time constraints, dependability requirements and security requirements were met the software was considered good enough. In addition to this many of the devices and components connected to the information system have used proprietary or non-standard communication protocols and drivers. These fitted technologies do not fit well to the vision of future industrial systems as they cannot be easily scaled horizontally or deployed automatically to more suitable computing domains.

1.3 Software development

To achieve the flexibility and interoperability required in a modern industrial information system, software development for industrial information systems as well as connectivity has mimicked approaches that have been proven successful in other areas. Especially the extraordinary success of the internet has proven its underlying technologies to be extremely scalable and interoperable in a heterogeneous environment. The basis of connectivity has shifted from proprietary industry specific standards towards ICT standards applied everywhere. Ethernet, wireless local area networking and mobile networks are developed by researchers and engineers to enable time-sensitive networking using the TCP/IP protocol family. [69] This would enable the same high level protocols to be used all the way from the enterprise management level to the production floor level.

Engineers, developers and researchers have put a great deal of effort in increasing software development speed and to easing up the configuration efforts. Software has been broken down to small reusable, stateless and composable modules called software components. Software components are units of composition, who foster reuse and prepare systems for changing individual parts. The use of components enables the division between developers and software architects and thus lowers the overall complexity of the development task. [40]

According to James Bach [55] software testability defines, how easily a program can be tested. According to his definition testability consists of operability, observability, controllability, decomposability, simplicity, stability and understandability. Componentization fits well to each lot already by definition. Component developers have a limited and well defined scope with clear requirements for each component. Architects can limit their testing efforts to the composition logic instead of proper functioning of each component. Traditionally components rely on a component model or a framework in which the components run. These frameworks include for example Microsoft's .NET framework, and JavaBeans.

To take componentization even further services have emerged to free the developers from component models or platform and programming language specific implementations. Perrey and Lycett defined services as something that adds value to the user and as a boundary between the technical and business perspectives as well as between consumer and a provider. [53] Using services as the building blocks of applications led to a paradigm called service-oriented architecture (SOA). In his seminal book Erl [18] published a series of design principles for SOA which include loose-coupling, abstraction, reusability, standardized service contracts, autonomy, statelessness, discoverability, composability and service-orientation and interoperability. These principles, even though over a decade old, line up perfectly to business requirements, and interoperability and flexibility needs set for the new industrial information and production systems.

For a decade web services using either World wide web consortium's (W3C) standardized technologies or representational state transfer (REST) ruled the research and industrial implementations of SOA. However recently new technologies have risen and a new paradigm called microservices has been derived from SOA. Microservices share the design principles introduced with SOA but introduce new technologies such as containerization and new concepts such as continuous refactoring and continuous delivery and deployment [9].

Researchers have put a lot of effort in making their cases for software componentization and SOA both in the ICT domain and in the industrial informatics domain. Microservices have been seen as the state of the art way of implementing SOA, especially in the cloud native environments. Large successful companies, including Netflix [47], Amazon [49], EBay and Google [62], and Microsoft in Azure [48] have taken an aggressive approach to adopting microservices architecture to their systems. This has been noticed also in the scientific community. For example, the IEEE had hosted an International workshop on Architecting with MicroServices in the year 2017 [46].

1.4 Research scope, method and goals

This thesis continues on the footsteps set by the scientific and industrial counterparts from an industrial informatics point of view. Microservices architecture in its today's form is a relatively new concept that has sprung out and quickly gained popularity in the application domain instead of research domain. Because of that it does not yet have an established position in the industrial informatics research.

Research in software engineering and computer science domain has concentrated on the organizational aspects and enabling software scalability and maintainability already from the start of development. However industrial software has often special quality of services requirements, which don't come up in typical consumer applications. These quality of service requirements include but are not excluded to security, reliability, predictable performance under load, hard real-time capability and graceful degradation. Satisfying the requirements is crucial for carrying out production orders safely and adequately. These requirements typically get stricter the closer the software is to the physical operational technology.

The goal of this thesis is to map the initial research efforts and to answer, how microservices based architecture can be adopted to the industrial information systems development. To aid in the research process the following questions were also formulated.

- What type of research is conducted on microservices in industrial information systems?
- What have been the motives behind the research?
- How have microservices been applied to industrial information systems? What are the emerging standards and tools for applying microservices in industrial information systems?
- What has been challenging in adopting microservices to industrial informatics?

In addition to answering the questions above this thesis analyzes and synthesizes the experiences and results published in research literature to gain further understanding in applying microservices in industrial informatics domain. As a result this thesis also aims in identifying gaps in existing research and possible opportunities for further research. The scope of this research is limited to only microservices and industrial information systems but literature and technical reports from the general software architecture domain are used in explaining microservices architecture and concepts related to it.

To answer the questions a research method called systematic literature review is applied. Systematic literature review was chosen as it fits the goals and is appropriate for answering the set research questions. Systematic literature review as a method has the capability to provide new information and insights from existing literature [63]. There is a clear demand in creating more agile and flexible industrial systems as mentioned in the earlier section. Systematic literature review helps the researchers and application developers to gather information especially as no other similar reviews exist. The research method is explained in detail in chapter 3 and the review protocol is documented in chapter 4.

2. MICROSERVICES

2.1 Concept of microservices

According to Vural et al. [66] microservices were first mentioned in 2010 by Fernandez-Villamor et al. in [19]. Fernandez-Villamor et al. described as ”a lightweight service classification framework for REST architectural style” that takes advantage of semantic service descriptions. More recent literature however maps microservices either as a new architectural style or as a specialized implementation of SOA with new technologies instead of a framework for REST [52].

One of the most cited definitions is written in Lewis and Fowler’s web article[22]. Their definition is inspired by the success stories of cloud native companies. In the article Lewis and Fowler describe microservices as an architectural style and define it through 9 common characteristics. The characteristics and Lewis’s and Fowler’s main motivations are listed below.

- Componentization via services. Services are independently deployable and thus a change in a service does not require the redeployment of the whole application. Explicitly published remote call interfaces and mechanisms make it impossible to break a component’s encapsulation and lead to looser coupling.
- Organized around business capabilities. Organize teams around services and business capabilities so that all required skills needed in development and project management are available in each teams. This way no one has to work around team barriers to create the experience desired.
- Products not projects. Teams should own the software product over its whole life cycle instead of handing it over to some other organ after project deadline has been met. The developers should be concerned of how their product could help the users optimally and add value over the whole life cycle.
- Smart endpoints and dumb pipes. Instead of applying logic and routing in the communication structures between services the services should implement all the logic and conversions needed and communication should be just simple requests and responses as in the web.
- Decentralized governance. Teams can choose the way they work and choose the most appropriate tools for their tasks. This is often difficult in centralized governance, which drives for standardized tools and platforms.
- Decentralized data management. Centralized data management helps in keeping data consistent but pushes different business capabilities to handle data in a uniform way,

which might not be suitable in fulfilling their task. Decentralized data management enables each business organ to handle their data in the most suited way.

- Infrastructure automation. Automating testing and deployment process is an important step in managing microservices architecture as deploying lots of different services manually is a gruesome task.
- Design for failure. Services running independently can fail at any time. Thus their state and operation is monitored and possible faults and exceptions are handled automatically.
- Evolutionary design. Services are developed in a manner, which makes it easy to control change. Some services might be scrapped altogether in production and this should not affect collaboration. [21]

Many other blog articles discuss similar characteristics or of a subset of the characteristics presented by Lewis and Fowler. For example, Giamas [24] writes about Zalando's migration to microservices architecture and describes characteristics similar to design for failure, infrastructure automation and componentization via services, Loftis [44] mentions componentization via services, infrastructure automation and organizing around business capabilities, and Richardson [56] writes about how componentization via services is done and mentions infrastructure automation as a requirement. In addition to blog articles Newman has published a list of seven characteristics. The list includes hiding implementation details into services, modeling around business services, decentralization, culture of automation, deploying services independently, isolating failures and observability. [50] Even though Newman's list is two items shorter than Lewis and Fowler's the content is almost identical and the items map really close to each other.

According to these discussions and definitions, microservices as an architectural style is an approach to developing an application as a set of small services. The services are built around business capabilities, all services are running in their own processes and they communicate with some lightweight mechanism. Each service may be deployed independently in a fully automated fashion. Governance and data management are distributed to the services and centralized management is reduced to bare minimum.

The researchers and developers, who argue microservices as a new incarnation of SOA, highlight the fact that many of the above mentioned characteristics are also achievable and desirable in SOA. [52] Zimmermann goes as far as stating that design for failure and high observability are key characteristics in any distributed system. Zimmermann also notes that Lewis and Fowler's, and Newman's characteristics are a mix of process, architecture and deployment properties. Architectural styles are usually defined using technical constraints or intent, principles and patterns. Mixing process and deployment concerns to the architectural definition blurs the actual architecture and the concerns would be easier to consume and to apply if they appeared in a separate and dedicated place.[74]

2.2 Advantages and drawbacks

Whether or not microservices is seen as an architectural style or as an implementation of SOA, it has clearly gained traction in professional application development. The process and development related characteristics in microservices have been an enabler and a success factor, in a time when scalability, latency and security requirements have blown up [74]. This section lists down common benefits and drawbacks related to application development using a microservices approach. The benefits and drawback were found from published surveys, migration reports or blogs. The advantages and issues, which appeared in more than a single source are collected to tables 2.1 and 2.2 respectively.

Table 2.1. *Advantages of choosing microservices approach*

Advantage	Description	Mentioned in
Clear boundaries	Each developer team is responsible only for the service they are developing and teams do not share responsibilities.	[64] [21][27][37]
Testability	Clearer scope and ownership gives developers an incentive for higher test coverage.	[21][37]
Scalability	System can be easily scaled by running more instances of a busy service behind a load balancer.	[66][21][73][64]
Quick feedback loops	Independent deployment supports faster releases and thus the developers are able to receive feedback from consumers quicker.	[64][27]
Seamless integration	Services developed in different programming languages or based on different data storages may be composed together seamlessly	[73][21]
Fault tolerance	Services are designed to work independently and to handle error cases gracefully. Multiple design patterns have been developed to handle cases where requested services are unavailable or where services need self healing.	[73][37][21][66]
Automation	Deploying multiple services encourages in automating the whole deployment pipeline and to reduce manual configuration efforts.	[66][21][68]
On demand performance	During high demand more services can be deployed to achieve desired response times	[73][27]

Vural et al. [66] list agility, autonomy, scalability, resilience, and easy continuous deployment as benefits of choosing microservices architecture but they discuss very little about the drawbacks. 3 out of these 6 benefits have been fitted to table 2.1. Resilience has been interpreted as fault tolerance, easy continuous deployment as automation and scalability is mentioned also in other papers as such. Zhu and Bayley [73] mention seamless integration, continuous evolution, optimal runtime performance, scalability and fault tolerance. 4 of the listed benefits were included in table 2.1 as such. Also Gouigoux and Tamzalit [27]

***Table 2.2**. Issues and drawback in microservices approach*

Issue	Description	Mentioned in
Increased brainload for developers	Architecture based heavily on services increases the total complexity of the system itself.	[64][4][68][73]
Requirement for automation	Complex deployment and testing scenarios are hard to set up and need senior and skilled developers. Manual deployment and composition is not sufficient and takes up a lot of effort.	[64][4][73]

mention performance and that microservices based application in their experience showed sometimes even 10 times faster response times than the previous monolithic version of the application. In addition to performance Gouigoux and Tamzalit list increased reuse, easier and faster replacement, and lower customer support requirements. Faster replacement was interpreted as quick feedback loops in the table 2.1 as it was described in a similar way. Increased reuse was not mentioned by other papers probably as it should be achievable with any component based development method.

Killalea [37] writes about the hidden dividends of microservices. Killalea includes 3 benefits, which are mentioned also in other articles, in the descriptions of the hidden dividends. These 3 benefits are clear boundaries, fault tolerance, and testability. Taibi et al. [64] have listed scalability, clear boundaries, independent deployment, and quick feedback loops as benefits. All except independent deployment were mentioned as benefits in at least one other article. Independent deployment has been mentioned in multiple articles but instead of a benefit it has been mentioned as a property, requirement or a characteristic. Finally in their migration report Wingelhofer et al. [68] compare their experiences from a small development oraganization's point of view against the common characteristics listed by Lewis and Fowler. They experienced that the main advantages were automation and enabling continuous evolution. Out of the 9 listed characteristics the development organization could fill 8. Only decentralized governance was not suitable in the organization in question.

Issues regarding microservices approach were all similar. Taibi et al. [64], Wingelhofer et al. [68], Zhu and Bayley [73], and Balalaie et al. [4] all state that microservices approach introduces a lot of complexities in the system. Learning the new tools and techniques required is difficult and takes time. Handling all the tools and understanding, how the system functions increases the developers' brainload. Taibi et al. also list that as the architecture is expected to evolve quickly over time the developers need to be particularly aware. Both Taibi et al and Balalaie et al. go as far as stating that developing applications using microservices approach requires more skilled or senior developers than the monolithic counterpart. Taibi et al. and Balalaie et al. also mention that deploying the services becomes an issue and requires automation. Again building the automated pipeline requires new type of skills and experience. Zhu and Bayley cite Daya et al., who state that there is no point in implementing microservices without automation. Even though both of the issues in table 2.2 handle increasing brainload, requirement for automation was introduced as separate issue as it was not mentioned by Wingelhofer et al.

The advantages listed above are well in line with the characteristics discussed in 2.1. The interesting thing in the mentioned issues and advantages is that both of the listed issues were also seen as advantages in some other publication. For example automation or an advantage closely related to automation was mentioned 3 of the chosen articles and still it made its way also to the issues. Taibi et al. also saw the continuously evolving architecture as a disadvantage even though Wingelhofer et al. listed it as the main benefit in their publication. These contradictions may be a sign that developers in some organizations are more prepared for the migration. The developers might have prior experience in developing distributed systems and better understanding of continuous development methods. Also the organizations' products and solutions could be inherently a better fit for a microservices based approach.

2.3 Core technologies and concepts

Some of the central concepts are already mentioned earlier in this chapter but this section provides more insight and details about key concepts and some central technologies associated with them. When discussing about microservices three concepts are mentioned in almost all publications about microservices. These concepts are development operations (DevOps), containerization, and messaging.

2.3.1 Development operations

According to Jabbari et al. [36] individual studies do not consistently use a single definition of DevOps. Instead of a single definition some common components can be observed in these studies. From these common components Jabbari et al. derieved that "DevOps is a development methodology aimed at bridging the gap between Development (Dev) and Operations (Ops), emphasizing communication and collaboration, continuous integration, quality assurance and delivery with automated deployment utilizing a set of development practices." as a short definition for DevOps after data extraction from 44 DevOps related research articles. The definition wraps the cultural and technological aspects nicely together.

From a culture point of view DevOps aims at breaking the walls between the maintanance / support and developers. Developers should take part in the maintanance and support and vice versa. [34] Close collaboration and teamwork between development and operations is also critical, because not all information can be effectively shared over team boundaries [45].

Even though cultural aspects are strongly present in the defintion, in practice DevOps is often realized only by utilizing tools to form an automated pipeline from development to deployment [34]. Establishing a the pipeline also works as an enabler in the cultural shift as the developers and operations personnel have to together form an understanding of the products whole life cycle already in the beginning of the development process. A typical DevOps pipeline from development to production environment is depicted in figure 2.1.

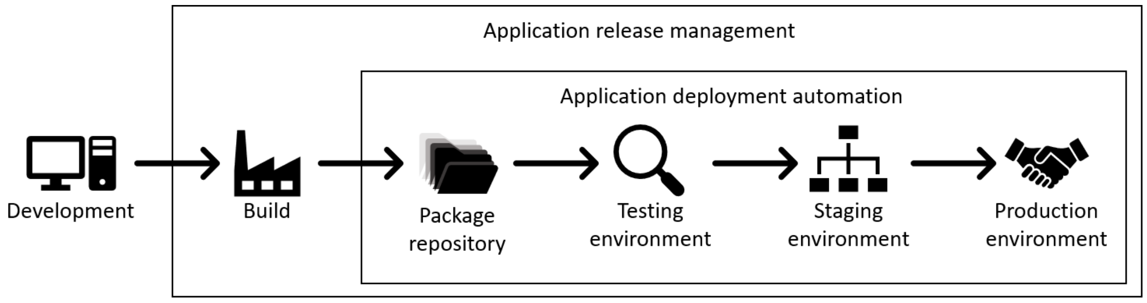

***Figure 2.1**. Typical development operations pipeline. Adopted from [73].*

The DevOps pipeline forms an automated path from development to deployment. Developers write their own unit and regression tests for the features they provide. These tests are run after the build along with other regression and integration tests. Features that pass these tests are pushed to package repository. From the repository new features are published to a testing environment, in which other developers or a select set of end users get to test the features. If problems occur the changes can be easily reverted back to the previous version from the repository. If no problems arise the feature can be pushed on to the staging environment, where it waits for release to production environment. The feature is pushed to production when it has passed the tests and the feature release is scheduled.

The cultural aspects of DevOps are really similar to the microservices characteristics organizing around business capabilities and products not projects. Both concentrate on ripping specialized one dimensional teams apart and promote collaboration and product ownership. Also the reliance on tools and automation is present in both DevOps and microservices. The most prominent defining factors have been embedded in the microservices approach. Because of this it's no wonder that DevOps is often referenced in the literature regarding microservices or that microservices approach shares a lot of the concepts with DevOps.

2.3.2 Containerization

Containerization, sometimes referred to as operation system level virtualization, is a virtualization technique similar to virtual machines. Virtual machines have been the go to virtualization technology for a decade. A full virtualized system gets a set of isolated resources allocated for it and requires a full guest operating system installed on its allocated resources. In contrast to virtual machines, containers allow sharing the underlying platform and infrastructure in a secure but also portable way. Containers share the kernel with the host operating system and only run in separate processes on the container engine in the host operating system. This makes containers require less file space, random access memory and a lot faster at startup than an equivalent virtual machine. Containers contain packaged ready-to-deploy parts of applications and if needed the necessary business logic and / or middleware. [51] The difference in architecture stack is portrayed in figure 2.2

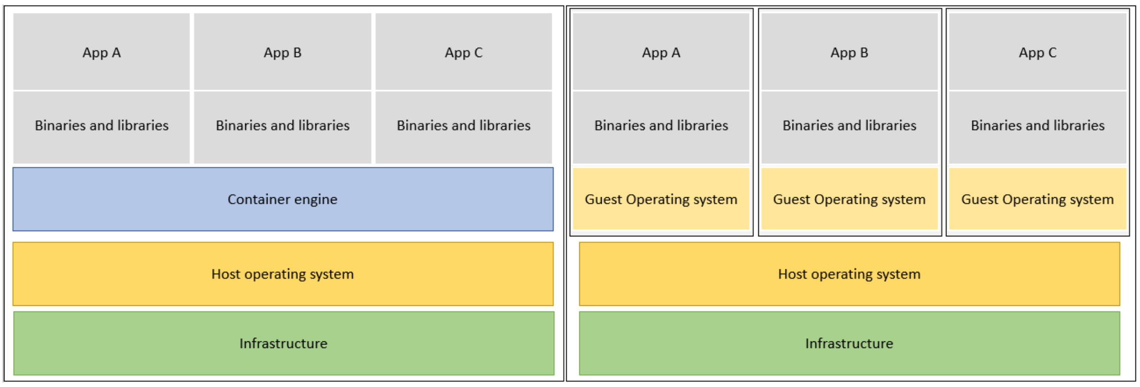

Figure 2.2*. Comparison between virtualization architecture achieved by using virtual machines and container. Virtualization using containers on the left and virtualization with virtual machines on the right. Adopted from [51].*

The properties mentioned above make containers suitable for software delivery. Each remote machine can run several containers all sharing the same underlying kernel. Containers take up only a negligible amount of drive space compared to a virtual machine and they can be sent through network and started up almost instantly. In addition containers contain all required dependencies so they can be deployed on any machine that runs on the same operating system. The same properties also make containerization an excellent tool in microservices architecture as it enables fast and automated deployment in an isolated environment with all dependencies but access to shared resources.

Docker is the de facto industry standard for containerization to such an extent that in some publications it is viewed as synonymous to containerization. In their review Vural et al. [66] mentioned Docker as the most often occurring tool in proposed or implemented solutions. Docker provides its own image and runtime specifications, has built in support in Amazon's, Microsoft's and Google's cloud environments, and the deployment of containerized applications can be automated using Docker's own orchestration engine Swarm or Google's Kubernetes. Docker also provides metrics for monitoring the execution of containers. [73]

2.3.3 Lightweight messaging

Messaging and data interchange play a huge role in distributed applications such as those designed and developed with microservices. In the traditional SOA approach W3C standardized WS* web services often rise as the proposed or implemented solution. However in microservices approach according to Vural et al. [66] these standards were not mentioned even once in the literature regarding microservices. Instead Vural et al. see Representational state transfer (REST) as the emerging standard in messaging. Even the W3C markup standards WSDL and WADL were completely absent in the papers included in the review and only Swagger (the tools for implementing Open API specification) appeared as a service description markup language.

REST

Representational state transfer is an architectural style developed for distributed hypermedia systems. As an architectural style, REST doesn't introduce any implementation details but instead the architectural style just a set of constraints for the components and their interfaces. [20] Using SOAP as an interaction protocol requires several specifications and formal documents even for a simple web service. REST architectural style provides a simpler alternative.

REST is derived using six principles: client-server model, statelessness, cache, uniform interface, layered system and code-on-demand. The principles are applied in the order they are mentioned starting from a null point in which there are no constraints whatsoever. The key behind client-server model is the separation of concerns. The user interface is stored and handled on the client side and data is stored on the server. Statelessness states that session state is stored in the client side and all information needed to fulfill a request is passed to the server in the request. Since sending all relevant session state over the network each time a request is made, caching is introduced as the following constraint to improve network efficiency. Information labeled as cacheable may be stored in the client cache and reused for later equivalent requests. [20]

According to Fielding uniform interface between components is the central feature, which distinguishes REST from other architectural patterns. Uniform interface simplifies the architecture, increases visibility and decouples the implementations from the services they provide. Layered system constraint enables the service to hide full complexity of the services. The client only interacts with the utmost layer of the service and he does not need to be aware of the intermediaries such as load balancing or security policies in place. Finally the code-on-demand is an optional constraint. It allows the client to download applets or scripts from the server to extend its functionality. [20]

JSON

JavaScript Object Notation (JSON) is an alternative data interchange format for XML. The goal of JSON has been to make it lightweight, text-based and highly interoperable. Like the name JSON suggests JSON objects can be made from JavaScript objects. However, this behavior is not in any way exclusive to JavaScript and most programming languages offer ways to pack objects into JSON format effectively. JSON can represent four different primitive types, *strings*, *numbers*, *booleans*, and *null*, and two structured types, *objects*, and *arrays*. [11]

The idea of an object varies widely between different programming languages. Languages are continuously evolving and the concepts of objects are often divergent. That's why JSON instead describes objects as collections of name and value pairs. Most programming languages have some way of managing this kind of collections, often with a name like *struct*, *record*, *dict*, *map*, *hash*, or *object*. [16] Objects in the JSON format are zero or more

```
ExampleObject = {
    "name":"John",
    "age":30,
    "cars": [
        { "name":"Ford", "models":[ "Fiesta", "Focus", "Mustang" ] },
        { "name":"BMW", "models":[ "320", "X3", "X5" ] },
        { "name":"Fiat", "models":[ "500", "Panda" ] }
    ]
 }
```

Program 2.1*. ExampleObject with properties name, age, and cars in JSON notation.*

name and value pairs wrapped inside curly brackets. Names are strings. A single colon after the name separates the value from the associated name. A single comma separates the value from the following name. All names in an object should be unique. [11] An example of a JSON object is shown in program 2.1.

In program 2.1 the object ExampleObject includes name and value pairs with names *name*, *age*, and *cars*. The last mentioned name is associated with a value representing another object. This kind of object nesting is possible and allows the creation and representation of complex graphlike structures in JSON format. JSON doesn't directly support cyclic graphs [16].

Internal representations of a number varies between programming languages. Most languages make a distinction between integers, floating, fixed or binary numbers. The differences in internal representations make interchange between languages difficult. JSON format doesn't make a difference between integers or decimal numbers. All numbers in JSON are just a sequence of digits. All programming languages have a way of making sense of these sequences even if the internal representations would differ. [16]

Strings in JSON are a sequence of Unicode points wrapped in quotation marks. Special characters like backslashes or other quotation marks wrapped inside the sequence can escaped by inserting a backslash in front of the special character. Alternatively, if a character is in the basic multilingual plane it may be represented as a six character sequence beginning with a backslash. This type of string representation is common in the C family of programming languages. [11]

Array structures are a common concept in almost every programming language. They often go by names like *array*, *vector*, or *list*. In JSON array is described with a list of values separated by single commas and wrapped inside square brackets. Values can be any of the types JSON supports and nesting objects or arrays inside arrays is possible. [16] Example of arrays are shown in program 2.1. The name *cars* is associated with an array of objects. Each of the objects has a name *models*, which refers to an array of strings.

2.4 Aligning microservices to the future visions

The properties and concepts mentioned in this chapter and the visions for more flexible and interoperable industrial informatics match together nicely. Particularly the technical advantages scalability, seamless integration, on demand performance, automation and fault tolerance mentioned in the table 2.1 fit perfectly with the vision of seamless computing where business critical workloads are mobile and can be moved between computing domains according to their requirements. For example Aazam et al. [1] describe the capability for fast response times and computation offloading as ”inevitable requirements” for realizing industrial internet of things.

Operating system level virtualization makes it possible to wrap applications and all their dependencies to an easily and quickly deployable package. Different tools and services already allow monitoring service loads and automatic horizontal scaling to tackle peak loads, and orchestration tools enable complex service compositions. Operating system level virtualization also tackles interoperability issues in the heterogeneous environment as all devices running some supported operating system can also run the containerized applications with minimal configuration efforts. In addition to this interoperability issues are tackled by using standard technologies in communication interfaces and hiding implementation specific details such as data formats or programming languages inside the service implementation.

Developers and engineers who use and maintain the industrial information systems are more than often not the same people who have developed the services. Contractors, consultants and in-house experts develop, design and often introduce the systems before handing them out to operators and maintenance. However, on the shop floor of a production facility it is common to have equipment and software from various vendors. Most of the development work has already been done by these vendors and they carry the responsibilities of their products and software. This habit matches nicely with the clear boundaries organizational advantage of microservices. Especially if the development on the vendors side is organized around business capabilities.

The major disadvantage of the microservices based architecture is increased complexity and brainload for developers. The main reason for the increased complexity comes from the distributed nature of microservices. This distributed nature is an inherent feature of industrial information systems in general and thus the developers should be familiar with defining interfaces and complex composition situations. Still the range of possible hardware and software components is huge and the software development processes may vary between organizations. Transition to an even more complex system of systems might still cause headache for some developers.

3. SYSTEMATIC LITERATURE REVIEW

Huge amount of research is produced world wide each year. Especially in the technology oriented fields of study the amount of published material is overwhelming. New technologies are adopted on multiple domains simultaneously and simple keyword searches might be insufficient to find relevant information. Also independent sources might reveal conflicting results due to sample variation, study differences, flaws or bias. These situations blur the overall picture. What has been studied? Which results are reliable? Should the results be used as a basis for policy decisions or industrial implementations?

Systematic literature review (SLR) is a clearly defined research method, which addresses the above mentioned problems. SLRs aim at being methodological, repeatable, and thorough. They adopt a systematic approach to minimize any bias, which might be prevalent in ad-hoc research. [3] Baumeister & Leary [5] give five main goals for systematic literature reviews. The main goals are

- theory development,
- theory evaluation,
- mapping,
- problem identification,and
- providing historical account on a topic.

Each goal has implications for the structure and the place of the article. Already from the ambitious goals, it is easy to see that a SLR is a piece of research on its own. By its nature, SLRs are able to address much broader questions than single empirical studies ever can. SLRs are a popular research method in multiple fields of study. Adaption to software engineering has happened quite recently through *evidence-based software engineering* [38]. SLRs are called secondary studies. Secondary studies form new information by combining parts or elements from results from primary studies in a quantitative or qualitative manner.

SLRs are an appropriate research method for novices. There are clearly defined protocols and guidelines on conducting SLRs.[3] This thesis follows the guidelines set by Kitchenham et al. [39]. In the guidelines the SLR process is split into 3 phases: planning, conducting, and reporting. Each of the phases is covered in more detail in the subsections. All subsections are adopted from the guidelines written by Kitchenhamm et al. The few side notes taken from other sources are cited accordingly.

3.1 Planning phase

The final goal of the planning phase is to set up a research plan and to formulate research questions. The research plan defines the progression of the research. The research questions are an integral part of the research. Everything in the research should contribute in answering the set questions objectively and according to the research protocol. Kitchenhamm et al. [39] divide the planning phase to five steps.

1. Identifying the need for the research.
2. (Commissioning the review.)
3. Specifying the research questions.
4. Developing a review protocol.
5. (Evaluating the review protocol.)

The first step in the planning process is identifying the need for the research. This includes going through existing reviews on the topic and getting an overall image of the field of study by reading existing research on the topics. When the need for a research is identified, research questions may be formulated. One could also commission the research, if he is not capable of conducting the research on his own.

The most important step in the planning phase is specifying the research questions. Research questions drive the whole process of planning and conducting a SLR. Research questions decide, what is researched, which studies are brought into inclusion consideration, and by which criteria the studies are searched for. The research questions play also a large role in the methods used in data extraction and data analysis stages. Research questions have to explicitly describe the researched topics and they have to be relevant for the researchers and for those, who plan on taking advantage of the research results. Answers to the research questions should provide new information or verify existing information.

Review protocol determines the methods used in conducting the SLR. Review protocol includes the strategies for searching, including and excluding studies, the assessment criteria for included studies, strategies for data extraction and data analysis, and finally the review schedule. All parts of the review protocol should be described in a systematic and explicit manner.

Data analysis is often referred to as data synthesis. In data synthesis the reviewers collate and summarize the results of the included primary studies. The form of synthesis to be used in conducting the SLR should be defined in the review protocol. As the actual data is not available in the planning phase, some of the issues can not be resolved in the planning phase. For example if there is no evidence of heterogeneity in the data, subset analysis investigating heterogeneity is not required in the conducting phase.

According to Kitchenhamm et al [39] there are three forms of synthesis, narrative, quantitative and qualitative. Quantitative synthesis is appropriate, when the results are in a numeric form or the results may be converted to a numeric form. Qualitative synthesis is used when results are in natural language. Narrative synthesis is used in highlighting similarities or differences between studies. In a narrative synthesis the information should be tabulated in a manner consistent with the research questions. Applying statistical techniques to obtain quantitative synthesis from descriptive (non-quantitative) results is called a meta-analysis.

The planning phase is often an iterative process. The results in later phases of the reviewing process might affect the earlier phases. For example, identifying the need for research and specifying the research questions are strongly coupled and study inclusion and exclusion criteria defined in the review protocol might need altering during the conducting phase. As the planning phase is a critical element in the credibility and success of the review, the protocol should be peer-reviewed. Detecting the deficiencies in the review protocol as early as possible saves both reviewers' time and effort. [39]

3.2 Conducting and reporting phases

In systematic literature reviews, the conducting phase is a step-wise and iterative process as the planning phase. Conducting phase should be tested already during the planning phase in order to continuously improve the research plan using the arising problems. Conducting phase progresses as defined in the research plan. First step is to search the selected resources and select the studies to include in the review. These studies are then further analyzed and their results are assessed and synthesized. [39]

Searching the resources aims at finding all relevant studies handling the selected topic. Preliminary searches can be conducted using digital libraries, but final searching needs to be done using multiple resources and as described in the review protocol. Different keywords and conditions should be trialled to find the most suitable ones. Used resources, search dates and search conditions need to be documented explicitly so that the readers can assess the quality and extent of the searches.

After searching the reviewer has to select, which studies will be included in the review. According to Kitchenhamm et al's guidelines the study selection is done according to the inclusion and exclusion criteria described in the review protocol. This is a sensitive part in the SLR. The reviewer must exclude the studies, which don't answer any set research questions. Still all relevant studies that might affect the results of the SLR have to be included in the SLR. Siddaway [63] recommends keeping track of borderline cases. If many cases require subjective judgement in the selection, the inclusion and exclusion criteria should be revisited.

The following step in the conducting phase is assessing the quality of the selected studies. Assessing the quality of an individual study is challenging. In fact Babar and Zhang [3] list it as one of the major challenges in conducting SLRs. Babar and Zhang argue that because of the lack of clear guidelines in assessing quality, only a few of the researchers actually assess the quality of selected studies in SLRs. The guidelines provided by Kitchenham et al. also mention the difficulty of assessing the quality of an individual study but state that bias and method validity problems can be noticeable. The results of the quality assessment can be used to investigate whether the quality differences provide explanation for different results or to guide recommendations for further research.

The reviewer designs data extraction forms to make conducting data extraction systematic and objective. The reviewer has to design the extraction forms so that all information needed to answer the research questions and to assess the study quality are captured in the forms. If quality conditions are set in the criteria used in study selection, the reviewer should design a separate quality assessment form. This is because the information must be collected already before the main data extraction phase. If quality assessment is done only as a part of the data analysis, quality criteria and the review data can be included in a single form. According to Kitchenham et al. at least two reviewers should perform data extraction independently on the selected papers whenever possible. If this is not possible, data extraction should be made as consistent and systematic as possible. After the data extraction the results are analyzed and gathered for synthesis. The synthesis should be performed as described in the review protocol.

The final phase of the SLR process is reporting. Main goals in the reporting phase are presenting the findings and the documentating SLR process including the review protocol in a readable form. Kitchenham et al. include the submission to appropriate scientific publications and peer reviews as a part of the reporting phase.

4. REVIEW PROTOCOL

Review protocol used in conducting this thesis study is presented in this chapter. SLR is a method for objectively analyzing existing information and combining it to form new information. SLR as a research method and the guidelines used in conducting this thesis study is described in the previous chapter.

After getting a picture of the current state of research, research questions for this thesis were formulated. Then the search conditions and keywords for finding all studies relevant for answering the research questions were composed. From the found studies the ones matching the inclusion and not exclusion criteria were selected. The selected studies went through the quality assessment and data analysis phases. 17 studies were finally selected for the data extraction and analysis.

4.1 Research questions

Initial mapping showed that microservices research has not yet matured. The studies that came up during examination had various definitions of microservices but most implementations had similarities between them. Most articles had a positive tone and the implementations and experience reports stated multiple benefits when migrating to microservices. Despite the positive tone only a few articles in the initial mapping had any references to industrial information systems. Thus the final research questions remained open-ended. Final research questions are listed below.

R1 What type of research is conducted on microservices in industrial information systems?

R2 What have been the motives behind the research?

R3 How have microservices been applied to industrial information systems? What are the emerging standards and tools for applying microservices in industrial information systems?

R4 What has been challenging in adopting microservices to industrial informatics?

4.2 Searching process

Conducting a SLR starts with defining the searching process and searching the studies from selected resources and with selected conditions. The searching process in this thesis is split to 3 phases. The studies included in the data analysis are the results of the whole searching process. The first phase of the searching process is defining the rules for searching and selecting resources to be searched. The second phase is defining the inclusion and exclusion criteria to be used in selecting the included studies from the list of all found studies. After defining the rules the actual search is conducted.

4.2.1 Search criteria

Defining search criteria for the SLR was a challenging task. To eliminate irrelevant search results as early as possible, search criteria have to be well thought. However, too strict search criteria might exclude also relevant studies from the search results. In this thesis this problem was approached iteratively in a top-down fashion.

The first iterations for finding suitable search criteria was done with extremely loose criteria. First searches were done using just keywords *microservices*. These searches in various digital libraries included thousands of results. Just by adding the keyword *industrial* after *AND* boolean operator to the end of the keywords limited the amount of results to a fraction of the first search. In fact the results diminished so much that it was highly probable that with this search criterion some plausible articles were not included in the search results. By changing the keyword after the *AND* operator it became clear that multiple different keywords were needed to find all relevant articles. After trial and error the final search query formed to:

(microservices AND industrial) OR (microservices AND industry) OR (microservices AND factory) OR (microservices AND manufacturing) OR (microservices AND cyber-physical) OR (microservices AND cyberphysical).

With this query large majority of the found studies seemed relevant according to their titles. In some digital libraries such as ACM Digital Library the number of boolean operators in a single query was limited so the query had to be split up to several queries. The final search criteria left some seemingly irrelevant results, but a broad scope enables as many articles as possible to be brought into consideration.

4.2.2 Selection criteria

To be included in the SLR every study had to pass at least one of the inclusion criteria. Besides passing at least one inclusion criterion the study was not allowed to match any exclusion criterion. When defining criteria, the main target was to make the selection process fast and simple.

The exclusion criteria were kept simple. Following exclusion criteria were set for the found articles.

E1. Study does not address microservices as a means of developing future industrial information systems.

E2. Study is not a scientific publication. (Conference article or a journal article)

E3. Study is a review.

E4. Study is published before the year 2014.

These criteria were selected as they appeared sufficient in eliminating studies, which were irrelevant in answering the set research questions, already in an early stage of the whole SLR process. Books about microservices were also ignored as the thesis study is aimed more at scientific peer-reviewed publications. Studies published before 2014 were eliminated in the selection process, as there have been huge advances in technology during recent years. Also the shift in architectural models and movement to cloud has made a lot of of the publications prior to 2014 irrelevant to this study.

Also 4 inclusion criteria were set. To be included in the review a study had to fill at least one inclusion criterion and none of the exclusion criteria.

I1. Study includes a vision of using microservices in industrial information systems.

I2. Study mentions one or more technologies that are used in microservices architecture.

I3. Study lists challenges or benefits of choosing microservices architecture.

I4. Study states motivations for using microservices in industrial information systems.

These inclusion criteria made sure each included study made remarks on the topic which helped in answering the research questions set for the review. To check a study against the criteria the abstracts of each study were read. In borderline cases even the whole study was read to ensure the study met requirements.

4.2.3 Search

Search was conducted using search engines IEEE Xplore, ACM Digital Library, ScienceDirect, and SpringerLink. To make sure no relevant studies were missed collective search engines Google Scholar, Scopus, and Web of Science were also used. The results of the latter search engines included large amount of duplicates. Multiple resources and search engines helped in reaching as large coverage as possible while using only digital libraries.

The total number of results found and the number of studies brought into consideration from each digital library using the query mentioned in section 4.2.1 is 25. Many of the articles were rejected as they matched the exclusion criterion E1. Especially the keywords industry and industrial brought a lot of misses to the results, as they could be interpreted to any industry and industrial adoption in software engineering. Further details of the search and results are in chapter 5.

4.3 Study assessment

As mentioned in the previous chapter, the quality of included studies needs to be assessed. Also the quality of literature concerning the topic gets simultaneously evaluated while assessing individual studies. No studies were excluded after the assessment phase. Study assessment focuses mainly on the quality of reporting. One can not make assumptions on the reliability of the results based on the quality assessment. Results of the study assessment are presented in section 5.2.

A scoring method was developed for the quality assessment. Scoring was done by answering predetermined questions. All questions were composed so that they could be answered *yes*, *no*, or *partly*. Assessed study received 1 point from each yes answer and 0.5 points from each partly answer. Study did not receive any points from a no answer. The overall quality score was the sum of points received from all questions.

Assessment questions are taken from the long list of study quality assessment questions provided by Kitchenham et al. [39]. Selected questions were most appropriate for this thesis's research questions. Selected questions mainly address the clarity of reporting, bias, and repeatability of the studies. Assessment questions and their explanations are written below.

A1. Are the aims of the study clearly stated?

A2. Are the measures used in the study clearly defined?

A3. Are the results clearly stated?

A4. Are all study questions answered?

Goal of the first assessment question is to find out, have the writer's identified the need for their research and stated the ambitions of the study. The motivation, provided related works and background, and research questions of the study were examined to answer this question. In practice this assessment question yielded only yes or no answers.

The second question evaluates the research methodology used in the study. Especially, how well was the used method reported in the study. Explicitly and clearly reported methodology earned 1 point, a general explanation earned 0.5 points and ignoring methodology did not earn points.

The third and fourth assessment questions deal with the results of the study. The third question focuses on the reporting. Clearly reported results earned 1 point, implicitly reported results earned 0.5 points and ambiguously reported results or findings did not receive any points. The fourth question is more focused on the quality of the reported results. Do the presented results answer to the research questions set in the study? If the study answered most set research questions, it received 1 point. Otherwise the study received 0.5 points, if it answered the most important research questions and 0 points, if the study did not clearly answer any set research questions. A perfect quality assessment score is thus 4.

4.4 Data extraction

Data extraction follows study assessment. The goal in data extraction is to find all relevant information for answering the SLR's research questions from the included studies. In this thesis a data extraction form was used in data extraction. Due to the nature of the thesis and the set research questions most of the fields in the data extraction form are natural language fields. Also only a few of the included studies had clearly stated numerical values in them.

Table 4.1. *Data extraction form.*

Field	Explanation	Data type	Reason
F1.	Researchers	Text	Reference
F2.	Publication	Text	Reference
F3.	Title	Text	Reference
F4.	Year published	Integer	Reference
F5.	Cited by	Integer	Reference
F6.	Assessment score	Decimal	Quality assessment
F7.	Research method	Text	RQ1.
F8.	Results	Text	RQ1.
F9.	Motives of applying microservices	Text	RQ2.
F10.	Challenges and benefits	Text	RQ4.
F11.	Technologies used	Text	RQ3.
F12.	Other relevant information	Text	Other

Studies were carefully read through while simultaneously filling the data extraction forms for each study. All fields in the data extraction form and their explanations are shown in table 4.1. In addition to the information needed in answering the review's research questions, the data extraction form includes fields, which capture information about the study itself. Fields F1 — F5 capture the essential referencing information of the study. Field F6 is the overall quality assessment score for the study, which is calculated as explained in section 4.3. Field F7 is documented to map the research methods used in the included studies but also answers to RQ1 as such. Field F7 is also used in analyzing the field of study itself.

Rest of the fields in table 4.1 are included in the data extraction form to enable answering in the research questions. As the results in the included studies are not quantitative, data captured in the fields from F8 to F12 is in textual form.

4.5 Data analysis

Finally the extracted information is analyzed and put together to form new information. In this thesis the method for conducting data analysis is qualitative meta-synthesis. In the guidelines by Kitchenham et al [39] the qualitative meta-synthesis is referred to as mere reciprocal translation. However Walsh & Downe [67] split the qualitative metasynthesis into two phases.

According to Walsh & Downe first part of qualitative meta-synthesis is to compare and group concepts and technologies presented in the included studies. The result of the first phase is a grid of concepts, in which the concepts closely related to each other are grouped together. Reciprocal translation is done after the completion of the grid of concepts. In the reciprocal translation the synthesis is build from bottom-up by comparing the data extracted from one study individually to other included studies. Concepts, which appear synonymous, are translated to a single concept and similarities found in the studies are bundled together.

In this study the reciprocal translation is done as a first step in the qualitative meta synthesis and it is reported already in chapter 5 where the data extraction results are reported. Raw data extraction results were so comprehensive that they would have taken too much space in the scope of this thesis. After the data extraction results the qualitative analysis is reported in the form of discussion, and comparison between the extracted results and the IT domain in chapter 6. This analysis method is slightly different to what was proposed by the guidelines or by Walsh & Downe but fits the set aims and research questions better.

5. OVERVIEW OF RESULTS

5.1 Search results and general information

The search was conducted on August 2nd 2018. The results from each search engine are collected to table 5.1. The total number of search results was quite high, 1175. Fortunately a large fraction of the studies were not valid for consideration based on their title alone and this eased up the selection work. Also over half of the results (751) were from Google Scholar. These results included lots of books, book chapters and articles from non peer reviewed sources like arXiv.org and thus the results were faster to go through than results from other libraries.

***Table 5.1**. Total number of studies found and the number of studies brought into consideration.*

Resource	**Found studies**	**Articles considered**
IEEE Xplore	51	8
ACM Digital Library	20	3
ScienceDirect	109	9
SpringerLink	113	0
Google Scholar	751	5
Scopus	96	0
Web of Science	33	0
TOTAL	1175	25

Finally only 25 articles were brought into inclusion consideration based on their titles and abstracts. Even though multiple test searches were done in the IEEE, ACM, ScienceDirect and SpringerLink search engines when fine tuning the search query Google's world-renowned algorithms were able to find four articles that matched the search query but did not show up in their own search engines. Also one article published in Wiley's digital library was brought into consideration from the Google Scholar search.

Out of the articles brought into consideration 23 were fully available online. These articles were read through carefully and assessed against the exclusion and inclusion criteria. Out of the 1175 search results 17 were included into the review. These articles are listed in table 5.2.

The number of articles shrank even more when considered articles were read through since two articles were published with slightly different titles but essentially the same input in different journals, also four of the articles brought into consideration after reading the title and abstracts did not match any of the set inclusion criteria. The final number of articles included in the review is low. This goes to show that research that fits in to the scope of the

***Table 5.2**. Studies included to the review.*

Ref	Authors	Title
[57]	Rufino J., Alam M., Ferreira J., Rehman A., Kim, F.T.	Orchestration of containerized microservices for IIoT using Docker
[35]	Innerbichler J., Gonul S., Damjanovic-Behrendt V., Mandler B., Strohmeier F.	NIMBLE collaborative platform: Microservice architectural approach to federated IoT
[58]	Schäffer E., Leibinger H., Stamm A., Brossog M., Franke J.	Configuration based process and knowledge management by structuring the software landscape of global operating industrial enterprises with Microservices
[41]	Lee C.K.M., Zhang S.Z., Ng K.K.H	Development of an industrial Internet of things suite for smart factory towards re-industrialization
[15]	Santos De Brito M., Saiful H., Steinke R., Willner A.	Towards Programmable Fog Nodes in Smart Factories
[25]	Gogouvitis S.V., Mueller H., Premnadh S., Seitz A., Bruegge B.	Seamless computing in industrial systems using container orchestration
[7]	Bloch H., Hansel S., Hoernicke M., Stark K., Menschner A., Fay A., Urbas L., Knohl T., Bernshausen J.	State-based control of process services within modular process plants
[65]	Thramboulidis K., Vactsevanou D., Solanos A.	Cyber-physical microservices: An IoT-based framework for manufacturing systems
[6]	Bloch H., Hansel S., Hoernicke M., Stark K., Menschner A., Fay A., Urbas L., Knohl T., Bernshausen J.	A microservice-based architecture approach for the automation of modular process plants
[26]	Götz B., Schel D., Bauer D., Henkel C., Einberger P., Bauernhansl T.	Challenges of Production Microservices
[2]	Afanasev M., Fedosov Y.V., Krylova A.A., Shorokhov S.A.	An application of microservices architecture pattern to create a modular computer numerical control system
[10]	Ciavotta M., Alge M., Menato S., Rovere D., Pedrazzoli P.	A Microservice-based Middleware for the Digital Factory
[29]	Ha J, Kim J., Park H., Lee J., Jo H., Kim H., Jang J.	A web-based service deployment method to edge devices in smart factory exploiting Docker
[8]	Borodulin K., Radchenko G., Shestakov A., Sokolinsky L., Tchernykh A., Prodan R.	Towards Digital Twins Cloud Platform: Microservices and Computational Workflows to Rule a Smart Factory
[60]	Seningtona R., Patakib B., Wang X.V.	Using docker for factory system software management: Experience report
[30]	Habl A., Kipouridis O., Fottner J.	Deploying microservices for a cloud-based design of system-of-systems in intralogistics
[54]	Porrmann T., Essmann R., Colombo A.W.	Development of an event-oriented, cloud-based SCADA system using a microservice architecture under the RAMI4.0 specification: Lessons learned

thesis has not been conducted much despite of the popularity of microservices in the IT domain.

All of the included articles have been published either year 2017 or year 2018. There were

77 unique author names listed in the included articles and only a group of seven authors had been listed as authors in two papers. Both the recent publication dates and the broad range of authors indicate that even though microservices have been a hot topic in IT domain for already multiple years their potential has been noticed in the industrial information systems domain just now.

5.2 Quality assessment

The quality assessment using the protocol shown in section 4.3 focused mainly on how well the reporting was conducted in each study. Quality assessment scores are visually in figure 5.1 and in table 5.3.

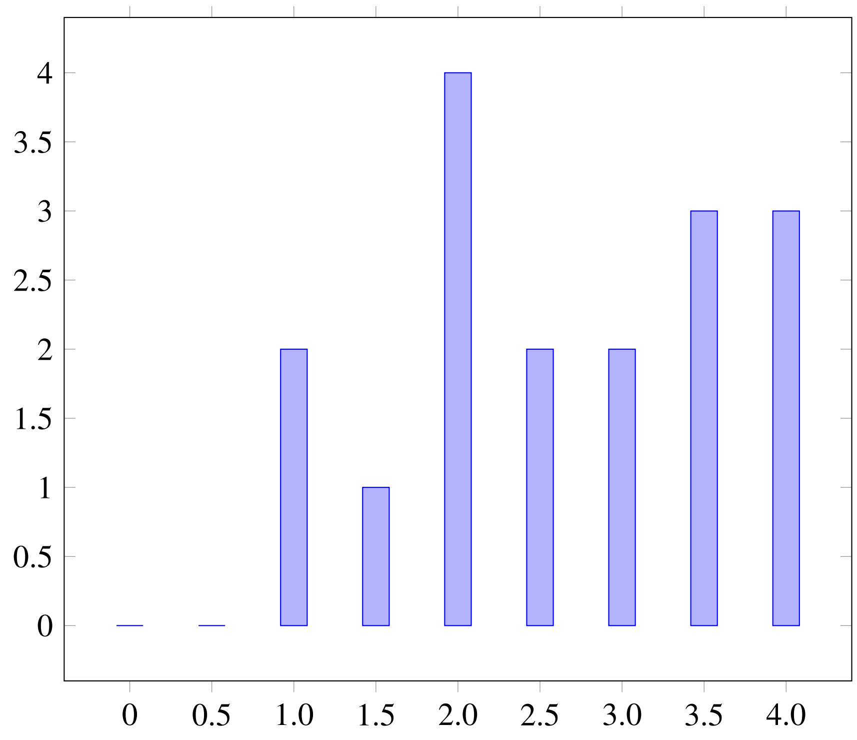

***Figure 5.1**. The distribution of overall quality assessment scores.*

***Table 5.3**. Average and deviation of assessment scores by assessment question*

Assessment question	Score 0	Score 0.5	Score 1.0	Average	Deviation
A1.	0	8	9	0.76	0.26
A2.	7	4	6	0.47	0.44
A3.	0	7	10	0.79	0.25
A4.	4	5	8	0.62	0.42
Overall				2.65	1.02

As figure 5.1 shows, the overall assessment scores are almost uniformly distributed from 1 to 4, and none of the studies received an overall assessment score below 1.0. However, the assessment criteria were set so that score 2.0 was possible even with poor / implicit reporting that required reading between the lines. 7 out of the 17 included studies received an assessment score lower than or equal to 2.0, which is alarming.

Table 5.3 shows the number of studies that received a certain score, average scores, and the deviation of the scores grouped by the assessment question. Stating the aims or goals of the study (A1) or documenting results (A3) were not a problem. All studies included had stated their aims some way and 9 out of 17 studies stated the goals of the study well. All studies also documented the results and conclusions and 10 out of 17 studies documented results and findings clearly and concisely. Problematic parts in many studies were documenting the measures used (A2) in the study and answering the research questions or meeting the aims stated in the study (A4).

Only 6 studies defined the used measures clearly and 7 studies did not define any of the used measures or their relationship to the research. Some studies could not define their measures, because of the nature of the study such as [8], in which the authors define the concept of a cloud platform that provides Digital twins as a service. There were also studies, which received a 0 from assessment question A2 even though there could have been measures. A good example is [29], where the authors' goal is to provide an easier way to deploy microservices to edge devices. This ease of use was not tested or at least the measures used were not reported in the study.

Score of assessment question A4 was often proportional to the assessment question A2. In most of the studies that received a bad score in A4 it seems like somewhere in the process of writing the authors' had lost track of their goals and changed the view point ever so slightly that the documented results might have been well in line with the used measures but not really with the aims or problems stated in the beginning. Again here the studies that were proposals or visionary make an exception to the proportionality.

5.3 Data extraction results

5.3.1 Research types

The articles could be split in to three separate main categories based on their research types. Six of the studies included an implementation and its analysis, two studies were experience reports and the rest were narrative studies, which defined architecture proposals, concepts or paradigms.

Results reported in the included articles were mostly qualitative and visionary. This is obvious for the concept definitions and some what assumed for architecture proposals, but even the articles that included demonstrations or proper implementations lacked quantifiable results. The exceptions to this were [2] and [57], in which the authors provided clear quantified measurement results of their implementations' performance.

5.3.2 Motivations for choosing microservices

Motivations for using microservices architecture as an approach for developing future industrial information systems were stated in every included article. The motivations for choosing microservices architecture could be split to six different categories: increased flexibility, modularity, better interoperability between components, faster and easier deployment, technology scalability and cost reduction.

The difference between flexibility and modularity in the scope of this thesis is that flexibility refers to the possibility of reconfiguring existing software and hardware combinations for different tasks whereas modularity refers to being able to build, deploy or change parts of the software without affecting the rest of the software. Interoperability means the possibility to make heterogeneous devices communicate with each other or even run the same software. Faster and easier deployment means that software could be distributed centrally and automatically to their final runtime environment.

Most included articles listed multiple reasons for choosing microservices and a few had mentioned only a single reason. The motivations mentioned in each article are in table 5.4. In most cases the motivations were stated with the exact words that are the column headers in table 5.4. In articles [8] and [58] the motivations were derived from the content in articles' introductory chapters.

***Table 5.4**. Motivations for choosing microservices architecture*

Ref	**Flexibility**	**Modularity**	**Interoperability**	**Deployment**	**Scalability**	**Costs**
[57]	X			X		
[35]			X			X
[58]						X
[41]		X	X			X
[15]			X	X	X	
[25]			X		X	
[7]				X		
[65]	X	X				
[6]	X	X				
[26]	X		X			
[2]	X			X		
[10]			X		X	
[29]				X		
[8]					X	
[60]				X	X	
[30]			X			
[54]			X		X	X

5.3.3 Perceived challenges and benefits

The included articles mentioned multiple challenges and benefits in implementing production level software for industrial information systems using microservices. 5 articles out of the 17 did not mention any challenges or problems for choosing microservices. Only [29] did not mention any benefits in choosing the microservices approach.

All perceived challenges could be classified to five main problems, which are either related to the technologies in use or to the software development using microservices based architecture. These five main problems were meeting real time requirements (latency), increased network load, defining interfaces, service encapsulation and increased overall complexity. Latency and increased network load are related to the used technologies and the three other challenges are development related. The challenges and the articles that mentioned them are summarized in table 5.5.

Table 5.5. *Main challenges and the articles in which they were mentioned.*

Challenge	Number of occurences	Mentioned in
Latency	4	[65][2][25][60]
Increased network load	3	[26][2][41]
Defining interfaces	3	[65] [54][10]
Encapsulating functionality	3	[65], [54][7]
Increased complexity	6	[29][54][30][10][65][6]

Out of the 12 articles that mentioned any perceived challenges four listed latency as a problem in the microservices architecture. All four of the studies that mentioned latency as a problem used some container technology for virtualisation and concluded that the container technologies used in the IT domain are not suitable for real-time operations per se. The increased network load was mentioned by three articles. As the information is distributed in to more granular components in different network locations, the information has to be shared over network and thus network load increases.

Thramboullidis et al. [65] highlight that the integration of microservices running on containers is costly and that the integration delay is magnitudes larger than a function call even when using operating system's own inter-process communication methods. Thus the use of fine-grained microservices in real-time systems can have unwanted side effects. Götz et al. [26] also note that the network load is proportional to the service granularity and thus service granularity is an optimization problem with flexibility, robustness and resources as the criteria. Gogouvitis et al. [25] present a mapping of container technologies to non-functional requirements. According to their mapping none of the included container or container orchestration technologies have real-time capabilities.

Afanasev et al. [2] reason that in a classic industrial management system it is hard to produce use cases where multiple operators would need to work on one device. According to them this situation changes with cyber-physical production systems. Instead of a single person working with a device, the device can share its information to and receive orders from multiple different systems or operators. This puts an extra emphasis on the system's networking capabilities. In their study Afanasev et al. provide network measurements from a system mimicking a CNC machine implemented using microservices architecture. Afanasev et al. conclude that the system implemented using microservices makes modules as autonomous as possible, extremely interoperable and simplifies scaling the application. They also conclude that networking was not a major issue in their implementation. The notable thing in their implementation is that it did not use any layer of virtualization in the components.

Finally Senington et al. [60] point out that even though container technologies were a key component in their experience report it still remains unclear, whether or not the container technologies are stable enough and do version changes break existing implementations. They were also concerned that the performance impact of containers could make them unsuitable for use in smart factories due to lack of responsiveness in physical tools.

Defining interfaces, encapsulating functionality to services and increased complexity were all classified as challenges related to software development. Both the difficulty to define appropriate interfaces to microservices and the difficulty to decide how to split the functionality to independent services were found in three articles. The increased complexity was mentioned as a challenge in 6 articles. This makes increased overall complexity of the systems the most immanent challenge of the five main challenges. Increased complexity was perceived in multiple different parts of the process from IT infrastructure to integration.

Porrmann et al. [54] claim that using microservices requires providing multiple servers to host the microservices and that this server infrastructure must be configured in a certain manner to handle the composition of these microservices. Their claim fails to include the possibility of deploying microservices to any system capable of running the service, even the edge devices. The deployment and composition should be done using the tools designed particularly for this purpose. However, Habl et al. [30] take this possibility into account but still conclude that using microservices increases the overall complexity as the developers and operators have to have a good understanding of the software tools and system components. Habl et al. also highlight the increased need for communication and information sharing.

Ciavotta et al. [10] link the complexity of the systems with the amount of services. As more and more small services are created the relationships between the services get easily tangled. They reason that experienced developers have a higher chance to keep the system functional and to deal with the emerging problems in the best possible way. Ha et al. [29] argue that deployment of applications developed using microservices is too difficult for a typical factory operator. To solve these challenges they provide an idea of an external web application that provides an UI for automatic deployment.

Bloch et al. [7] propose a generic workflow to help in encapsulating process functionalities to services. In their opinion it is difficult to determine which way the encapsulation and combination of services is reasonable and thus the workflow helps in identifying the suitable approach.

In all of the above mentioned cases the difficulties boil down to not being familiar with microservices architecture and its characteristics. Thinking of software as independent services that interact with each other and are distributed automatically to the devices in the network is a drastic change to what the people are traditionally used to. The author argues that most of these difficulties are easy to overcome with a proper introduction to the development and the software tools. Naturally this introduction has its own opportunity cost attached.

The perceived benefits of microservices were a lot more scattered than the perceived challenges and could not be grouped in a similar fashion. In addition to the disjointedness, if an article mentioned any benefits, they mentioned multiple. Often the perceived benefits were fitting to the motivations, but this was not always the case. Also many of the benefits listed as benefits of using microservices are inherent to all component and services based approaches. That is why the list of benefits presented in this section is a cherry picked version which includes only the clear benefits that are caused by choosing microservices approach. These selected benefits caused from choosing microservices architecture were horizontal scalability, release from integration focused systems engineering and flexibility in the technology stack. These benefits and the articles that mentioned them are presented in table 5.6

Table 5.6*. Perceived benefits caused by choosing microservices*

Benefit	**Number of occurences**	**Mentioned in**
Horizontal scalability	6	[26][41][10][35][54][57]
Easier integration	4	[6][2][26][60]
Flexibility in technology stack	7	[65][2][54][10][26][35][57]

Horizontal scalability was mentioned as a benefit in six of the included articles. Horizontal scalability was achieved by spawning new instances of the services under load and balancing the load between these services. In addition to increased information throughput the horizontal scalability also introduced fault tolerance as faulted services could be replaced with new freshly spawned service instances on the fly. Horizontal scalability was the most technical perceived benefit as the other benefits were more development related.

Four articles mentioned easier integration as a benefit. Instead of focusing on integrating different systems by coordinating service calls, routing and transforming data, the development could be focused on autonomous and independent services with communication capabilities and clearly defined open interfaces. This means that different parts of the systems can be easily changed without massive reconfiguration as long as physical and virtual interfaces are unified.

Finally flexibility in the technology stack was mentioned in seven articles, which makes it the most often mentioned benefit even when factoring in the benefits that were left out of reporting. Flexibility was perceived as a possibility to reuse existing services easily, to change implementations in the existing services, to choose the appropriate programming language and data storage for the tasks and to let software evolve regardless of underlying hardware. Surprisingly, this wide definition of flexibility was largely shared among all seven articles that mentioned it.

5.3.4 Technologies used

Overall the technology landscape was wide and flat. 33 different technologies were used in the proposals or implementations in the articles and many more were mentioned. 27 of the used technologies were unique and mentioned only once in all the included articles. The selected and named technologies addressed various parts of the overall systems such as operating systems, virtualization, data management, messaging, data interchange formats and cloud platforms.

Protocols related to messaging and data exchange between services were most frequently mentioned and every implementation used at least one protocol. 6 unique protocols were used or proposed. The most often mentioned method for communication was HTTP, which was used in 6 cases. In 5 of the 6 cases the communication was described as ”using REST” and the author took the liberty to translate this to HTTP. After HTTP, OPC UA was used in 3 cases. Those two were the most used protocols. MQTT, COAP, LwM2M and WebSockets were all named in one article each. Even though the messaging protocols were mentioned in every article the data interchange formats were mentioned in only 2 articles. One of these articles used JSON and the other XML and RDF.

Database technologies played an integral part in multiple included articles. NoSQL technologies dominated this category as Redis was used in 3 articles, Cassandra in 1 article and 1 article just named that a NoSQL database was used. MySQL was the only traditional SQL database used in the included articles.

Only operating system level virtualization technologies were used in virtualization. Docker was the only container technology that was used in included articles and it was named 5 times. The orchestration engine provided with Docker, Docker Swarm, was used in container orchestration in 1 article. Kubernetes was the other software used in container orchestration and it was also used in 1 article. 3 unique operating systems were named, Hypriot, FreeRTOS and ArchLinux. Each of them was named only once.

Python was the only programming language named as the development language and it was mentioned in two articles. Various other implementation technologies were also mentioned once in the included articles. Web authorization standards OAuth2.0, OpenID Connect and SCIM were all mentioned once, Microsoft Azure and Amazon web services IAAS platforms were used in 2 separate articles, Apache Spark was mentioned as a proposal for big data handling platform and finally Apache Kafka data streaming technology was mentioned as a way to stream information in an architecture proposal.

In addition to the implementation technologies, a couple of markup and description languages were used in the design and development phase. UML, SysML and Automation ML were all used once in the included articles to help in the design and modeling phases. WSDL and RESTdesc were the only service description languages mentioned in the included articles and both of the description languages were used once. 3 articles brought up the concept of development operations.

6. DISCUSSION

6.1 Discussion on data extraction results

In the included studies the general atmosphere around microservices development and industrial information systems architectures based on microservices was positive. Multiple studies stated that microservices are suitable for developing future industrial information systems. This pattern repeated despite the research type. Architecture proposals compared the technology capabilities to the requirements set by different production environments while test implementations took a more practical approach and analyzed the success of their implementations, but all articles concluded that microservices based architecture is a good step forward in realizing future visions mentioned in the introduction.

In the included articles microservices were applied to all computing domains from the edge to cloud. All parts of the industrial information systems were covered in the implementations and in the architecture proposals. Modularization of a single shop floor device as well as high performance computations and simulations in the cloud were covered. However the definitions of what is considered as microservices was not clear. This same problem was also present in the IT domain and discussed in section 2.1.

There were slight differences in the perceived techonology readiness levels in the included articles. This is mostly visible in the difference between studies that provided their results in the form of an architecture proposal and did not have any practical test case. The discrepancy in the perceived technology readiness is also visible in the quality of results. The results from the studies that built on existing research from IT domain and had practical implementations were by a large margin more significant in assessing the adoption of microservices in this thesis's context. Using existing and tested approaches in a different domain gave much more information on the applicability than visionary architecture proposals. This difference in the significance is not visible in the quality assessment as results might have been well reported regardless of the results' form.

The flexibility in the technology stack is also visible in the used technologies. Multiple different technologies were used in all parts of the system. The only visible emerging standard was operating level virtualization, where Docker was the only technology used. In messaging hypertext transfer protocol (HTTP) and OPC UA were the two most used protocols and could be seen as the two most suitable messaging protocols for industrial informatics. HTTP based messaging is the de facto standard in IT domain and there is an immense amount of information and existing tools available for it. These available resources make it easier to adopt HTTP and leverage existing cloud architecture compared to OPC UA. The lack of emerging technology standards is not a problem. In microservices architecture the developers should be free to choose any tools which are suitable for the task at hand and the developers are familiar with. Included articles stated this also as almost half of the articles listed this flexibility in the technology stack as a benefit caused by microservices.

The use of operating level virtualization is often associated with microservices architecture in both IT domain and the included articles. In industrial informatics this association brings up two big problems. Extensive use of container technologies is based on an expectation that all devices in the system will be capable of running a full fledged computer operating system to begin with. This is not yet the case as most small and medium manufacturing facilities the author has witnessed rely heavily on programmable logics and simple sensors and it is hard to predict the speed in which these devices will be replaced with small computers. The second problem is the overhead that using Docker brings. Two of the most often mentioned challenges in adopting microservices architecture were latency and network load. Adding a level of virtualization won't make these problems easier to overcome. In the included articles it was noted that integration of containers causes delays and that the performance of container technologies has not been properly measured and verified. The development of a real time container technology could be a reasonable way to tackle this problem. Another solution for this could be developing technologies that allow low latency communication within locally hosted services.

The researchers approached microservices with grand motivations and often they were able to achieve the vision or prove their concept at least partially. The motivations behind the research were well in line with the future visions and the new business requirements. Microservices was seen as a way to enable the workload mobility and more flexible and configurable production systems. Despite the problems mentioned regarding container technologies they are seen as an enabling technology more than any other technology. The tools, configurability and automatic deployment they provide fit the visions for future industrial information systems well.

Remarkably, interoperability between different parts of the systems was a main motivation in multiple included articles. This is surprising as the communication interfaces or data interchange formats have not been in any way standardized in microservices while in W3C standardized web services these are fixed and the methods are by standard discoverable.

Perhaps the W3C has gone too far with the standardized architecture and superfluous specifications. According to the included articles it is easier to agree on in house formats or to publish an application programming interface with a documented data format and access endpoint than to conform to a series of standards even when developing a simple service. In the included articles, only 2 used any service description markup. No matter how human and machine readable an XML file is, it does bring an extra layer of complexity to the integration. This is also visible in the perceived benefits as easier integration was one of the most often mentioned benefits for microservices.

Challenges in the adoption of microservices were either software development or technology related. The biggest technology related challenges, latency and increased network load were already discussed above in this chapter. The development related challenges are similar to the challenges that have come up also in the IT domain despite the author's presumption that industrial information system developers are more familiar with developing distributed software. There were no suggestions or tips for alleviating the development related challenges in the articles. Service oriented software development has been around for over a decade but software developers and researchers still find the paradigm challenging. This is probably due to lack of experience and education in the paradigm, and might change in recent years as the paradigm gains more and more traction due to the hype and success around service orientation.

The increased complexity was seen as a drawback also in the IT domain. In the IT domain these complexities have been alleviated with the use of tools and processes particularly designed to handle them. A great way to simplify the development process also in the industrial information systems development is to unify all processes to follow the same chain. Even though an identical continuous deployment model as in DevOps might not be possible in all the industrial information system cases, most of the steps in the development can be automated and made invisible in the daily activities. Another way to facilitate the complexity is to stick to the tools that have been designed for the tasks at hand since these problems have been partially solved already in the IT domain.

The organizational characteristics related to microservices did not come up as much in the industrial informatics research in comparison with the IT domain. None of the organizational aspects mentioned in section 2.1 were mentioned in the included articles. Software architecture and development practices that enable and encourage longevity and commitment to the software is important in any modern software business. In the author's opinion this was not stressed enough in the articles included in the thesis. Business is the driving force behind adaption and the organizational characteristics related to microservices are a key part of creating more business value as a part of software development. Including developers in to the deployment, maintenance and bringing the willing developers to the customer interface gives variation to the core development and more accurate support for customers.

In general the research in IT domain brought up more soft aspects in software architecture and the industrial information systems research focused on the technical aspects. In the author's opinion this is due to the fact that microservices are more widely known and mature in the formerly mentioned field of research and they are just getting noticed in the latter field field of research. Industrial systems tend to evolve slower and also the business models and practices are slower to adopt new approaches. The tardiness in industry is due to the higher dependability and security requirements for the systems, and the economic entry barrier. Original equipment manufacturers dominate with their scale and use their own proprietary protocols and technologies. This makes it harder and more expensive for newcomers to enter the market with new business models.

6.2 Research conclusions

According to the results microservices are suitable architectural choice for industrial software development. Microservices show potential for solving some of the subproblems in achieving more flexible, scalable and interoperable industrial information systems. Microservices as a software architecture needs a clear definition and guidelines in both IT and industrial information systems. This thesis does not provide a clear definition but it can at least bring the researchers and developers in both domains a bit closer together and bridge the gap between the two domains. In addition to this 3 rules of thumb for adopting microservices to industrial information systems development were derived from the results and analysis.

Real time applications should be deployed without a level of virtualization. The inter process communication inside a real time application should be minimized and the real time communication should be done in a dedicated network or in dedicated field buses. In microservices architecture this means that real time services are coarse grained and the service interfaces they expose are meant exclusively for starting or stopping sequences and monitoring the service state. These steps should minimize the latency problems that came up in the included studies.

Software with softer real time requirements should be deployed as containerized services. Development, deployment, monitoring and maintenance processes should be similar for these services. The path of least resistance is to adhere the de-facto standards and tools set in the IT domain as there is plenty of resources, experience reports and documentation available for these technologies. Leveraging virtualization increases flexibility and is a cornerstone for the workload mobility needed in the future industrial information systems.

Migrating existing software to microservices requires investments and experienced developers. Companies who consider migrating to microservices should start of with building an environment and infrastructure around microservices and educating staff to mitigate the development related issues. When done properly the migration to microservices architecture should reduce development efforts and enable building better software. Including education to the process as early as possible should also help with the perceived complexity of distributed software development.

6.3 Future opportunities

The author identified three clear future research opportunities from the data extraction and analysis. First branch is the development of operating system level virtualization technologies that allow for low latency near real time communication. This development could consist of totally new virtualization concepts, optimizations in container networking or optimizations in inter-machine communication. Containers with real time capabilities could enable using the same tooling and automation stack from enterprise software level to the shop floor device level, and thus render the first rule of thumb obsolete. This future research direction would benefit also the IT domain and consumer internet of things applications in addition to the industrial informatics domain.

The other two future research directions are related to proving the conjecture that using microservices with the above mentioned guidelines is an appropriate way of developing software for future industrial information systems. Full scale implementation either in a demo production facility or in a simulated environment would be a great test for the microservices architecture. Small and simple demonstrations have their own place in research and they enable building something quickly to prove a point or measure something. These small scale demonstrations can not however replicate the true complexities of a full blown production system.

Finally the third future research opportunity would be implementing a system with true dynamic workload mobility. Such systems were mentioned in included studies and dynamic workload mobility is a key part of realizing the future visions for industrial information systems. In fact, fog computing relies on the concept of dynamic workload mobility. Workload should be intelligently moved during runtime according to their performance and resource requirements, while simultaneously taking care of the services' states.

6.4 Research evaluation

Systematic literature review proved to be a valid approach to tackle the research problem of this thesis. Another way to tackle the set research problem would have been through experimentation and reporting. The experimentation would have made the research more susceptible for bias as the tight schedule would have forced the author to choose technologies and approaches he is familiar with. Systematic literature review as a method enabled a wider scope, the inclusion of multiple different approaches and thus more reliable results.

The research results are generalizable and the statements provided in the research conclusions are logically sound and in line with the information provided in the included literature. Qualitative analysis on the extracted data was also a valid approach. The data presented in the included articles was not quantitative or easily quantifiable. Information extracted from the articles helped in answering the set research questions. The theory regarding particularly microservices has not stabilized but the theory builds on extensive research on service oriented architecture.

The number of articles that came up in the search process was enormous, 1175. The final sample size of the literature review was low, 17. Either the inclusion process was too strict or more suitable articles were not available. In any case the low sample size lowers the significance and reliability of the research results. The search process is clearly documented in chapters 4 and 5 and anyone should be able to repeat the process.

Finally, in addition to the low sample size, almost half of the samples received low quality assessment scores. More than half of the samples failed to properly report the measures that had been used or to answer the research questions they had set. The poor sample quality also reduces the reliability of this thesis.

7. SUMMARY

The goal of this thesis was to map the research efforts put forth in adoption of microservices to industrial information systems and to answer how to adopt microservices architecture to industrial information systems. Microservices was chosen as the main phenomenon to research due to its recent popularity in the IT domain. In the IT domain microservices has been seen as a software architecture that encourages organizing software development more around business needs and makes it easier to build more flexible and scalable software by design. These characteristics are needed for the future industrial information systems as they are expected become even more software-centric in the recent years.

Four research questions were formulated in order to guide the research. The research was conducted as a systematic literature review. The method is documented and guidelines appropriate for software engineering reviews were used. Review protocol was documented and defined well before the actual research phase. In the research phase multiple digital libraries were searched for articles that would help in answering the set research problem. Found articles were screened against set inclusion and exclusion criteria and 17 articles were included in the data extraction and analysis.

Data extraction results were synthesized in the discussion chapter. According to the included studies microservices architecture fulfills the requirements for modern industrial information systems. As a result of the synthesis the author suggests that 3 rules of thumb are followed when adopting microservices architecture to industrial information systems.

- Real-time applications need special attention. They should be deployed on real hardware and operating system and the services should be granular. Real-time communication should be done in dedicated networks.
- Development, deployment, monitoring and maintenance processes should be similar for all services. Build on top of the de-facto standards and tools set in the IT domain.
- Start migration with building an environment and infrastructure around microservices, and educate staff members.

Finally the conducted research was evaluated. In conclusion the research was successful in answering the set questions. Results are valid and generalizable. However, due to the low sample size and low sample quality the research reliability is not gold standard.

REFERENCES

[1] M. Aazam, S. Zeadally, K.A. Harras, Deploying fog computing in industrial internet of things and industry 4.0, IEEE Transactions on Industrial Informatics, Vol. 14, Iss. 10, Oct, 2018, pp. 4674–4682.

[2] M.Y. Afanasev, Y.V. Fedosov, A.A. Krylova, S.A. Shorokhov, An application of microservices architecture pattern to create a modular computer numerical control system, in: Open Innovations Association (FRUCT), 2017 20th Conference of, IEEE, 2017, pp. 10–19.

[3] M.A. Babar, H. Zhang, Systematic literature reviews in software engineering: Preliminary results from interviews with researchers, in: Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on, IEEE, 2009, pp. 346–355.

[4] A. Balalaie, A. Heydarnoori, P. Jamshidi, Migrating to cloud-native architectures using microservices: an experience report, in: European Conference on Service-Oriented and Cloud Computing, Springer, 2015, pp. 201–215.

[5] R.F. Baumeister, M.R. Leary, Writing narrative literature reviews., Review of general psychology, Vol. 1, Iss. 3, 1997, p. 311.

[6] H. Bloch, A. Fay, T. Knohl, M. Hoernicke, J. Bernshausen, S. Hensel, A. Hahn, L. Urbas, A microservice-based architecture approach for the automation of modular process plants, in: Emerging Technologies and Factory Automation (ETFA), 2017 22nd IEEE International Conference on, IEEE, 2017, pp. 1–8.

[7] H. Bloch, S. Hensel, M. Hoernicke, K. Stark, A. Menschner, A. Fay, L. Urbas, T. Knohl, J. Bernshausen, State-based control of process services within modular process plants, Procedia CIRP, Vol. 72, 2018, pp. 1088–1093.

[8] K. Borodulin, G. Radchenko, A. Shestakov, L. Sokolinsky, A. Tchernykh, R. Prodan, Towards digital twins cloud platform: Microservices and computational workflows to rule a smart factory, in: Proceedings of the10th International Conference on Utility and Cloud Computing, ACM, 2017, pp. 209–210.

[9] L. Chen, Microservices: Architecting for continuous delivery and devops, in: Proceedings of the 2018 IEEE International Conference on Software Architecture, Mar. 2018, Seattle, USA.

[10] M. Ciavotta, M. Alge, S. Menato, D. Rovere, P. Pedrazzoli, A microservice-based middleware for the digital factory, Procedia Manufacturing, Vol. 11, 2017, pp. 931–938.

[11] D. Crockford, The application/json media type for javascript object notation (json), 2006. Available (accessed on 3.30.2018): http://www.ietf.org/rfc/rfc4627.txt

[12] W. Dai, V.N. Dubinin, J.H. Christensen, V. Vyatkin, X. Guan, Toward self-manageable and adaptive industrial cyber-physical systems with knowledge-driven autonomic service management, IEEE Transactions on Industrial Informatics, Vol. 13, Iss. 2, April, 2017, pp. 725–736.

[13] W. Dai, W. Huang, V. Vyatkin, Knowledge-driven service orchestration engine for flexible information acquisition in industrial cyber-physical systems, in: 2016 IEEE 25th International Symposium on Industrial Electronics (ISIE), June, 2016, pp. 1055–1060.

[14] W. Dai, Z. Zhang, P. Wang, V. Vyatkin, J.H. Christensen, Service-oriented data acquisition and management for industrial cyber-physical systems, in: 2017 IEEE 15th International Conference on Industrial Informatics (INDIN), July, 2017, pp. 759–764.

[15] M.S. De Brito, S. Hoque, R. Steinke, A. Willner, Towards programmable fog nodes in smart factories, in: Foundations and Applications of Self* Systems, IEEE International Workshops on, IEEE, 2016, pp. 236–241.

[16] The json data interchange syntax, ECMA International, web site, Dec. 2017. Available (accessed on 3.24.2018): http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

[17] M. Endrei, J. Ang, A. Arsanjani, S. Chua, P. Comte, P. Krogdahl, M. Luo, T. Newling, Patterns: service-oriented architecture and web services, IBM Corporation, International Technical Support Organization, 2004.

[18] T. Erl, Service-oriented architecture (soa): concepts, technology, and design, 2005.

[19] J.I. Fernández-Villamor, C.A. Iglesias, M. Garijo, Microservices - lightweight service descriptions for rest architectural style, in: ICAART, 2010.

[20] R.T. Fielding, R.N. Taylor, Architectural styles and the design of network-based software architectures, Vol. 7, University of California, Irvine Doctoral dissertation, 2000. Available (accessed on 4.12.2018): https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[21] L.J. Fowler Martin, Microservices a definition of this new architectural term, Martin Fowler's blog, 2014. Available (accessed on 27.6.2018): https://martinfowler.com/articles/microservices.html

[22] P.D. Francesco, I. Malavolta, P. Lago, Research on architecting microservices: Trends, focus, and potential for industrial adoption, in: 2017 IEEE International Conference on Software Architecture (ICSA), April, 2017, pp. 21–30.

[23] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, E. Riviere, Edge-centric computing: Vision and challenges, SIGCOMM Comput. Commun. Rev., Vol. 45, Iss. 5, Sept. 2015, pp. 37–42.

[24] A. Giannis, From monolith to microservices, zalando's journey, InfoQ blog, 2016. Available (accessed on 27.6.2018): https://www.infoq.com/news/2016/02/Monolith-Microservices-Zalando

[25] S.V. Gogouvitis, H. Mueller, S. Premnadh, A. Seitz, B. Bruegge, Seamless computing in industrial systems using container orchestration, Future Generation Computer Systems, 2018.

[26] B. Götz, D. Schel, D. Bauer, C. Henkel, P. Einberger, T. Bauernhansl, Challenges of production microservices, Procedia CIRP, Vol. 67, Iss. 1, 2018, pp. 167–172.

[27] J.P. Gouigoux, D. Tamzalit, From monolith to microservices: Lessons learned on an industrial migration to a web oriented architecture, in: 2017 IEEE International Conference on Software Architecture Workshops (ICSAW), April, 2017, pp. 62–65.

[28] O. Group *et al.*, Openfog reference architecture for fog computing, Fremont, CA, USA, Tech. Rep. OPFRA001, Vol. 20817, 2017.

[29] J. Ha, J. Kim, H. Park, J. Lee, H. Jo, H. Kim, J. Jang, A web-based service deployment method to edge devices in smart factory exploiting docker, in: Information and Communication Technology Convergence (ICTC), 2017 International Conference on, IEEE, 2017, pp. 708–710.

[30] A. Habl, O. Kipouridis, J. Fottner, Deploying microservices for a cloud-based design of system-of-systems in intralogistics, in: Industrial Informatics (INDIN), 2017 IEEE 15th International Conference on, IEEE, 2017, pp. 861–866.

[31] M. Hankel, B. Rexroth, The reference architectural model industrie 4.0 (rami 4.0), ZVEI, April, 2015.

[32] M. Hermann, T. Pentek, B. Otto, Design principles for industrie 4.0 scenarios, in: System Sciences (HICSS), 2016 49th Hawaii International Conference on, IEEE, 2016, pp. 3928–3937.

[33] P. Hu, S. Dhelim, H. Ning, T. Qiu, Survey on fog computing: architecture, key technologies, applications and open issues, Journal of Network and Computer Applications, Vol. 98, 2017, pp. 27 – 42.

[34] J. Humble, J. Molesky, Why enterprises must adopt devops to enable continuous delivery, Cutter IT Journal, Vol. 24, Iss. 8, 2011, p. 6.

[35] J. Innerbichler, S. Gonul, V. Damjanovic-Behrendt, B. Mandler, F. Strohmeier, Nimble collaborative platform: Microservice architectural approach to federated iot, in: Global Internet of Things Summit (GIoTS), 2017, IEEE, 2017, pp. 1–6.

[36] R. Jabbari, N. bin Ali, K. Petersen, B. Tanveer, What is devops?: A systematic mapping study on definitions and practices, in: Proceedings of the Scientific Workshop Proceedings of XP2016, New York, NY, USA, 2016, ACM, XP '16 Workshops, Edinburgh, Scotland, UK, pp. 12:1–12:11.

[37] T. Killalea, The hidden dividends of microservices, Commun. ACM, Vol. 59, Iss. 8, July 2016, pp. 42–45.

[38] B. Kitchenham, O.P. Brereton, D. Budgen, M. Turner, J. Bailey, S. Linkman, Systematic literature reviews in software engineering–a systematic literature review, Information and software technology, Vol. 51, Iss. 1, 2009, pp. 7–15.

[39] B. Kitchenham, S. Charters, Guidelines for performing systematic literature reviews in software engineering, 2007. Available (accessed on 12.2.2018): https://userpages.uni-koblenz.de/~laemmel/esecourse/slides/slr.pdf

[40] H. Koziolek, Performance evaluation of component-based software systems: A survey, Performance Evaluation, Vol. 67, Iss. 8, Special Issue on Software and Performance, 2010, pp. 634 – 658.

[41] C. Lee, S. Zhang, K. Ng, Development of an industrial internet of things suite for smart factory towards re-industrialization, Advances in Manufacturing, Vol. 5, Iss. 4, 2017, pp. 335–343.

[42] P. Leitão, A.W. Colombo, S. Karnouskos, Industrial automation based on cyber-physical systems technologies: Prototype implementations and challenges, Computers in Industry, Vol. 81, Emerging ICT concepts for smart, safe and sustainable industrial systems, 2016, pp. 11 – 25.

[43] S.W. Lin, B. Miller, J. Durand, R. Joshi, P. Didier, A. Chigani, R. Torenbeek, D. Duggal, R. Martin, G. Bleakley *et al.*, Industrial internet reference architecture, Industrial Internet Consortium (IIC), Tech. Rep, 2015.

[44] H. Loftis, Why microservices matter, Heroku blog, 2016. Available (accessed on 27.6.2018): https://blog.heroku.com/why_microservices_matter

[45] L.E. Lwakatare, P. Kuvaja, M. Oivo, Dimensions of devops, in: Lassenius, C., Dingsøyr, T., Paasivaara, M. (eds.), Agile Processes in Software Engineering and Extreme Programming, Cham, 2015, Springer International Publishing, pp. 212–217.

[46] I. Malavolta, R. Capilla, Current research topics and trends in the software architecture community: Icsa 2017 workshops summary, in: 2017 IEEE International Conference on Software Architecture Workshops (ICSAW), April, 2017, pp. 1–4.

[47] T. Mauro, Adopting microservices at netflix: Lessons for architectural design, 2015. Available (accessed on 20.6.2018): https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/

[48] Microsoft azure service fabric, 2018. Available (accessed on 20.6.2018): https://azure.microsoft.com/en-us/services/service-fabric/

[49] C. Munns, Devops @ amazon, 2016. Available (accessed on 20.6.2018): https://www.slideshare.net/TriNimbus/chris-munns-devops-amazon-microservices-2-pizza-teams-50-million-deploys-a-year

[50] S. Newman, Building microservices: designing fine-grained systems, " O'Reilly Media, Inc.", 2015.

[51] C. Pahl, Containerization and the paas cloud, IEEE Cloud Computing, Vol. 2, Iss. 3, May, 2015, pp. 24–31.

[52] C. Pautasso, O. Zimmermann, M. Amundsen, J. Lewis, N. Josuttis, Microservices in practice, part 1: Reality check and service design, IEEE Software, Vol. 34, Iss. 1, Jan, 2017, pp. 91–98.

[53] R. Perrey, M. Lycett, Service-oriented architecture, in: Applications and the Internet Workshops, 2003. Proceedings. 2003 Symposium on, IEEE, 2003, pp. 116–119.

[54] T. Porrmann, R. Essmann, A.W. Colombo, Development of an event-oriented, cloud-based scada system using a microservice architecture under the rami4. 0 specification: Lessons learned, in: Industrial Electronics Society, IECON 2017-43rd Annual Conference of the IEEE, IEEE, 2017, pp. 3441–3448.

[55] R.S. Pressman, Software engineering: a practitioner's approach, Palgrave Macmillan, 2005.

[56] C. Richardson, Microservices: Decomposing applications for deployability and scalability, Infoq blog, 2014. Available (accessed on 27.6.2018): https://www.infoq.com/articles/microservices-intro

[57] J. Rufino, M. Alam, J. Ferreira, A. Rehman, K.F. Tsang, Orchestration of containerized microservices for iiot using docker, in: Industrial Technology (ICIT), 2017 IEEE International Conference on, IEEE, 2017, pp. 1532–1536.

[58] E. Schäffer, H. Leibinger, A. Stamm, M. Brossog, J. Franke, Configuration based process and knowledge management by structuring the software landscape of

global operating industrial enterprises with microservices, Procedia Manufacturing, Vol. 24, 2018, pp. 86–93.

[59] B. Scholten, The Road to integration: A Guide to Applying the ISA-95 Standard in Manufacturing, International Society of Automation (ISA), 2017, 234 p. Available: https://www.isa.org/store/the-road-to-integration-a-guide-to-applying-the-isa-95-standard-in-manufacturing/44494834

[60] R. Seningtona, B. Patakib, X.V. Wangc, Using docker for factory system software management: Experience report, Procedia CIRP, Vol. 72, 2018, pp. 659–664.

[61] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: Vision and challenges, IEEE Internet of Things Journal, Vol. 3, Iss. 5, Oct, 2016, pp. 637–646.

[62] R. Shoup, From monolith to microservices - lessons from google and ebay, 2016. Available (accessed on 20.6.2018): https://www.slideshare.net/RandyShoup/from-monolith-to-microservices-craftconf-2015

[63] A. Siddaway, What is a systematic literature review and how do i do one, University of Stirling, Iss. I, 2014, p. 1.

[64] D. Taibi, V. Lenarduzzi, C. Pahl, A. Janes, Microservices in agile software development: A workshop-based study into issues, advantages, and disadvantages, in: Proceedings of the XP2017 Scientific Workshops, New York, NY, USA, 2017, ACM, XP '17, Cologne, Germany, pp. 23:1–23:5.

[65] K. Thramboulidis, D.C. Vachtsevanou, A. Solanos, Cyber-physical microservices: An iot-based framework for manufacturing systems, in: 2018 IEEE Industrial Cyber-Physical Systems (ICPS), IEEE, 2018, pp. 232–239.

[66] H. Vural, M. Koyuncu, S. Guney, A systematic literature review on microservices, in: Gervasi, O., Murgante, B., Misra, S., Borruso, G., Torre, C.M., Rocha, A.M.A., Taniar, D., Apduhan, B.O., Stankova, E., Cuzzocrea, A. (eds.), Computational Science and Its Applications – ICCSA 2017, Cham, 2017, Springer International Publishing, pp. 203–217.

[67] D. Walsh, S. Downe, Meta-synthesis method for qualitative research: a literature review, Journal of advanced nursing, Vol. 50, Iss. 2, 2005, pp. 204–211.

[68] R. Wingelhofer, M. Aistleitner, Microservices in a small development organization, in: Software Architecture: 11th European Conference, ECSA 2017, Canterbury, UK, September 11-15, 2017, Proceedings, Springer, 2017, Vol. 10475, p. 208.

[69] M. Wollschlaeger, T. Sauter, J. Jasperneite, The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0, IEEE Industrial Electronics Magazine, Vol. 11, Iss. 1, March, 2017, pp. 17–27.

[70] S. Yi, Z. Hao, Z. Qin, Q. Li, Fog computing: Platform and applications, in: 2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb), IEEE, 2015, pp. 73–78.

[71] X. Yue, H. Cai, H. Yan, C. Zou, K. Zhou, Cloud-assisted industrial cyber-physical systems: An insight, Microprocessors and Microsystems, Vol. 39, Iss. 8, 2015, pp. 1262 – 1270.

[72] F. Zezulka, P. Marcon, I. Vesely, O. Sajdl, Industry 4.0 – an introduction in the phenomenon, IFAC-PapersOnLine, Vol. 49, Iss. 25, 14th IFAC Conference on Programmable Devices and Embedded Systems PDES 2016, 2016, pp. 8 – 12.

[73] H. Zhu, I. Bayley, If docker is the answer, what is the question?, in: 2018 IEEE Symposium on Service-Oriented System Engineering (SOSE), March, 2018, pp. 152–163.

[74] O. Zimmermann, Microservices tenets, Computer Science-Research and Development, Vol. 32, Iss. 3-4, 2017, pp. 301–310.