Deliverable D1.1

State-of-the-Art and Market Requirements

WP 1

Project Acronym & Number:	SmartCLIDE – GA 871177
Project Title:	Smart Cloud Integrated Development Environment supporting the full-stack implementation, composition and deployment of data-centered services and applications in the cloud
Status:	Final
Dissemination Level:	Public
Authors:	UoM
Contributors:	All
Document Identifier:	D.1.1 State-of-the-Art and Market Requirements v.1.0
Date:	31.03.2020
Revision:	1.0
Project website address:	www.smartclide.eu

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No *871177*

Project Consortium

Institut für angewandte Systemtechnik Bremen GmbH (ATB), Germany

INTRASOFT INTERNATIONAL SA (INTRA), Luxembourg

FUNDACION INSTITUTO INTERNACIONAL DE INVESTIGACION EN INTELIGENCIA ARTIFICIAL Y CIENCIAS DE LA COMPUTACION (AIR), Spain

UNIVERSITY OF MACEDONIA (UoM), Greece

ETHNIKO KENTRO EREVNAS KAI TECHNOLOGIKIS ANAPTYXIS (CERTH), Greece

X/OPEN COMPANY LIMITED (TOG), United Kingdom

ECLIPSE FOUNDATION EUROPE GMBH (ECLIPSE), Germany

WELLNESS TELECOM SL (WT), Spain

UNPARALLEL INNOVATION LDA (UNP), Portugal

CONTACT SOFTWARE GMBH (CONTACT), Germany

KAIROS DIGITAL, ANALYTICS AND BIG
DATA SOLUTIONS SL (KAIROS DS), Spain

Dissemination Level

PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Change History

Version	Notes	Date
0.1	Creation of the document	04.02.2020
0.2	Initial Input from all partners	12.02.2020
0.3	Update of input from all partners based on workshop in KO	25.02.2020
0.4	Feedback from Task Leaders to all partners	03.03.2020
0.5	First draft version	10.03.2020
0.6	Review from KAIROS and ATB	15.03.2020
0.7	Second draft version	20.03.2020
0.8	Internal review from all partners	25.03.2020
0.9	Internal review from dedicated reviewers	28.03.2020
1.0	Final version	31.03.2020

Executive Summary

The current document constitutes the deliverable D1.1 “State-of-the-Art and Market Analysis” of the SmartCLIDE project. The deliverable aims to explore the current state-of-research and -practice in the topics of interest for the project, and deliver as a main outcome the baseline requirements for the intended framework. The deliverable has been developed using a well-defined strategy, and received contribution from almost all partners of the consortium, so as to provide an as comprehensive view of the current state of the art and market analysis. The deliverable is going to be provided as input to Task 1.2 “Specification of Requirements”.

Abbreviations

AAIT	Actionable Alerts Identification Technique
AI	Artificial Intelligence
ACL	Access Control Lists
ADALINE	Adaptive Linear Element
ALM	Application Life cycle Management
AOP	Aspect-Oriented Programming
API	Application Programming Interfaces
ASA	Automatic Static Analysis
ASD	Adaptive Software Development
ATDD	Acceptance Test Driven Development
BDD	Behaviour Driven Development
BRMS	Business Rule Management System
BPMN	Business Process Model and Notation
BSIMM	Building Security In Maturity Model
CI	Continuous Integration
CD	Continuous Deployment
CDI	Context Dependence Injection
CLI	Command Line Interface
CNN	Convolutional Neural Network
CNTK	Microsoft Cognitive Toolkit
CRUD	Create Read Updated Delete
CSV	Comma-Separated Values
CVE	Common Vulnerabilities Exposures
CWE	Common Weakness Enumeration
DAML	DARPA Agent Mark-up Language
DARPA	Defence Advanced Research Projects Agency
DL	Deep Learning
DNN	Deep Neural Network
DOM	Degree of Match
DT	Development Team
DSDM	Dynamic Systems Development Method
DURS	Deployed, Updated, Replaced & Scaled
DSL	Domain-Specific Language
ESB	Enterprise Service Bus
FaaS	Functions-as-a-Service
FDD	Feature Driven Development
FX	Executes Functions
GDPR	General Data Protection Regulation
GKE	Google Kubernetes Engine
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
GSOM	Growing Hierarchical SOM
GWT	Google Web Toolkit
HL	High-Level
HTTP	Hypertext Transfer Protocol
ICT	Information and Communications Technology
IDE	Integrated Development Environment
IIC	Industrial Internet Consortium
IIOT	Industrial Internet of Things
IO	Input / Output
IISF	Industrial Internet Security Framework
IR	Information Retrieval
IT	Information Technology
ITG	IT Governance
JSON	JavaScript Object Notation
JPA	Java Persistence API
JTA	Java Transaction API
KIE	Knowledge Is Everything
LDA	Linear Discriminant Analysis
LGPL	Lesser General Public License
LIPS	Learning Inductive Program Synthesis
LL	Lower-Level
LSP	Language Server Protocol
LSTM	Long-Short Term Memory
MIB	Management Information Base
ML	Machine Learning
MLRs	Multivocal Literature Reviews
MLP	Multi-Layer Perceptron
MS	Mapping Study
MSA	Microservice Architecture
MSE	Minimum Square Error
MVC	Model View Controller
NIST	National Institute of Standards and Technology
NN	Neural Network
NLP	Natural Language Processing
OAS	OpenAPI Specification

OS	Open-Source
OSGi	Open Services Gateway Initiative
OT	Operational Technology
OWL	Web Ontology Language
OWL-S	Web Ontology Language for Services
OWASP	Open Source Foundation for Application
PaaS	Platform as a Service
PCA	Principal Component Analysis
PCMONS	Private Clouds Monitoring Systems
PO	Product Owner
QA	Quality Attribute
QoE	Quality of Experience
QoS	Quality of Service
RAD	Rapid Application Development
RCP	Rich Client Platform
ReLU	Rectified Linear Unit
REST	Representational State Transfer
RGB	Red Green Blue
RNN	Recurrent Neural Network
RUP	Rational Unified Process
RV	Runtime Verification
S3	Semantic Service Selection
SAWSDL	Semantic Annotations for WSDL
SaaS	Software as a Service
SE	Software Engineering
SDK	Software Development Kit
SDLC	Software Development Life Cycle
SLA	Service Level Agreement
SLR	Systematic Literature Review
SM	Scrum Master
SNMP	Simple Network Management Protocol
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SOE	Service Oriented Enterprise
SOM	Self-Organizing Map
SoTA	State-of-the-Art
SQLI	SQL Injection
SSL	Semi-Supervised Learning
SVM	Support Vector Machine
TD	Technical Debt
TDD	Test Driven Development
TF-IDF	Term Frequency-Inverse Term Frequency
TLS	Transport Layer Security
UDDI	Universal Description, Discovery and Integration
UI	User Interfaces
UML	Unified Modelling Language
UX	User Experience
VCS	Version Control System
VM	Virtual Machine
VPM	Vulnerability Prediction Model
W3C	World Wide Web Consortium
WP	Work Package
WSDL	Web Services Description Language
WSMO	Web Service Modelling Ontology
WSQBE	Web Service Query by Example
YAML	Yet Another Mark-up Language
XML	Extensible Mark-up Language
XP	Extreme Programming
XSS	Cross-Site Scripting

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Document Purpose

Deliverable "D.1.1 State-of-the-Art and Market Requirements" is part of WP1, and is produced as the main outcome of Task 1.1, namely "Analysis of SOTA and Market Requirements". The aim of this deliverable is two-fold, and can be outlined as follows: (a) to provide all the necessary background information to guarantee the communication among consortium partners; and (b) to describe the state-of-research and practice in research areas related to the project. Achieving the aforementioned goals is important since the partners come from different backgrounds and therefore it is crucial to setup a baseline for communication within the consortium. In addition, the document will underline the research direction that will need to be continuously monitored by the partners and provide a first indication on the expected advancements that can act as the driving force for guiding the research directions. The deliverable will be an important input for Task 1.2 "Specification of Requirements", which will transform the market requirements that SmartCLIDE will bring to the community, to concrete requirements for the SmartCLIDE environment.

1.2 Approach

The research process that has been followed for completing this task is summarized in Figure 1. As a starting point for completing the task, we have used the SmartCLIDE proposal and in particular Section 1.4 (Ambition). Based on the goals of the SmartCLIDE project, we have identified the main research directions of interest, namely: *software development*, *development of microservice applications*, *software quality assurance in microservices*, and *AI for software development* in general. For each one of these research areas, we have collected information from various sources: such as, scientific literature, industrial experiences of the consortium partners, existing market solutions, and international standards. The retrieved information from all these sources has been synthesized, so as to provide uniform reports in the form of a literature/market overview. Based on this, the document will set the minimum / baseline market requirements for the project.

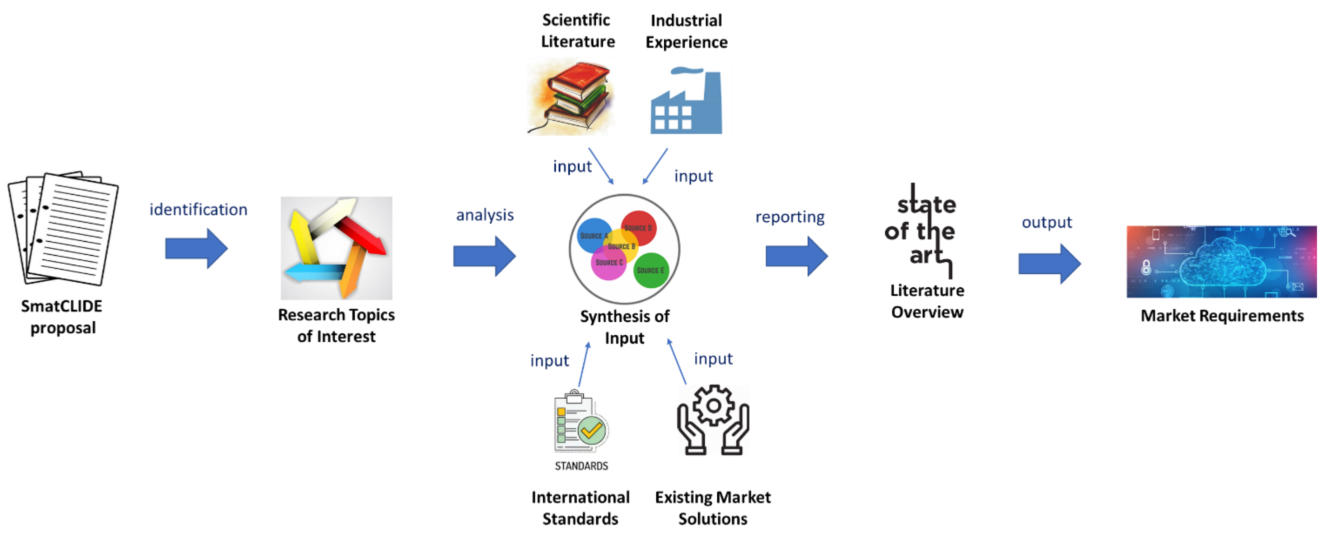

Figure 1: State-of-the-Art and Market Requirements Analysis

Furthermore, in the domain of software engineering, exploring and providing a holistic review of the literature has become a significant research topic [115]; which is currently performed in a highly systematic manner. The design and the execution of secondary (or even tertiary) studies is nowadays following established guidelines and can be performed in three main forms:

- ***Systematic Literature Reviews***. Systematic Literature Reviews (SLRs) use data from previously published studies for the purpose of research synthesis, which is the collective term for a family of methods for summarizing, integrating and, where possible, combining the findings of different studies on a topic or research question. Such synthesis can also identify crucial areas and questions that have not been addressed adequately with past empirical research. It is built upon the observation that no matter how well-designed and executed, empirical findings from single studies are limited in the extent to which they may be generalized [117].
- ***Systematic Mapping Studies***. Mapping studies (MSs) use the same basic methodology as SLRs, but aim to identify and classify all research related to a broad software engineering topic rather than answering questions about the relative merits of competing technologies/approaches that conventional SLRs address. They are intended to provide an overview of a topic area and identify whether there are sub-topics with sufficient primary studies to conduct conventional SLRs and also to identify sub-topics where more primary studies are needed [183].
- ***Multivocal Literature Reviews***. The main difference of multivocal literature reviews (MLRs) compared to the other two types of secondary studies is that in this kind of research efforts the grey literature (e.g., blog posts, videos and white papers) is taken into account, in addition to the published (formal) literature (e.g., journal and conference papers). MLRs are useful for both researchers and practitioners since they provide summaries of the state-of-the art and –practice in a given area. MLRs are popular in other fields and have recently started to appear in Software Engineering (SE) [72].

In addition to these more systematic ways of reviewing the literature, the traditional literature surveys[1] are still considered as a valid way in providing an overview of the literature, putting more emphasis on the raw content of primary studies and their discussion rather than synthesis. Therefore, for delivering the content of this report, any of the aforementioned research methods has been used. We note that industrial standards and existing market solutions that have not been highlighted from the literature, but are deemed as important from some partners (since they are already familiar with them) are included in the corresponding part of the above sections, in the sense that they are included in the starting point of the SmartCLIDE project. At the end of this deliverable, we present the elicited minimum market requirements (e.g., tools, need for technologies, etc.) that must be used in the SmartCLIDE development environment. The market requirements will be short, clear, and straightforward; and will be ranked, based on the Shall / Should / May prioritization method.

1.3 Document Structure

The document consists of the following sections:

- Section 2 "*Software Development*" describes software development methodologies and tool-support.
- Section 3 "*Development of Microservice Applications*" presents the current *statu quo* on the development of microservice applications
- Section 4 "*Software Quality Assurance in Microservice Applications*" describes the important aspects of quality assessment in these applications, i.e. design- and run-time quality attributes.

[1] https://dl.acm.org/journal/csur

- Section 5 "*Artificial Intelligence for Software Development*" covers the related work on the core of the SmartCLIDE project, i.e., the AI technologies that can be used for enhancing software development (in general) and code generation in particular.
- Section 6 concludes the deliverable by summarizing the main outcomes that will be fed to the requirements analysis and specification task.

2 Software Development

2.1 Software Development Paradigms

2.1.1 Software Paradigms: How Software is Designed and Developed

Historically, software developers have experimented with three major software development paradigms: procedural, data-driven, and object-oriented. Each of these paradigms attempts to solve a real-world problem with a software solution (see Figure 2). The ad/hoc path is the longest and least straight. The procedural is shorter and much more direct. The data-driven path is even shorter and even more direct. But the object-oriented path is the shortest and the most direct. Additionally, much of the earliest software produced was developed based on ad/hoc or impromptu paradigms [9].

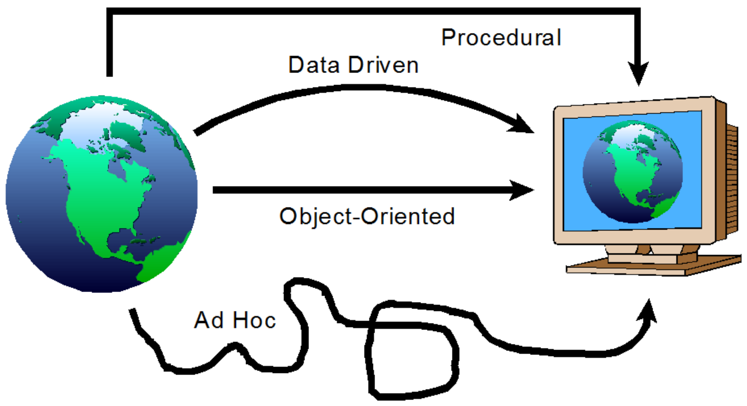

Figure 2: The four traditional software development paradigms

These ways of coding require a huge effort to generate abstractions that can be understood by human brains. Very recently, some visionaries like *Bret Victor*[2], former *Apple* engineer, or *Microsoft's Research Lab at Cambridge*[3], understanding the huge challenges the software community will have to face in the near future due to the lack of professionals, the scaling complexity of systems, and the growing demand of software products in the market, are plying for a radical change in the way we develop software. They have taken a retrospective look to the origins of software engineering, where experimental approaches that were much closer to human discernment were proposed, some of which have been developed to some extent over these decades, like visual programming or the programming-by-demonstration, which has been applied in the most recent robotics [160]. Some of the dysfunctional silver bullets that Brooks mentioned in his famous essay [33] were precisely visual programming or the application of Artificial Intelligence to support programming activities. But these technologies have been polished over the years; resulting to many successful applications of AI in various domains.

The SmartCLIDE consortium pursues the design and development of a Cloud IDE that offers full support to the services creation life cycle: from specification of user stories to deployment in the cloud. Having performed a retrospective look to software development approaches, the consortium aims to recover the **Coding-by-Demonstration / Coding-by-Example principle**, which consists on guiding a computer to

[2] http://worrydream.com/
[3] https://www.microsoft.com/en-us/research/lab/microsoft-research-cambridge/

program a specific behaviour that can be replicated in several occasions, and whose output can be comprehensible for a human with no technical skills. Seminal works of this type of programming can be found by several authors [53]. PbE is an end-user development technique for teaching a computer or system a new behaviour by showing actions on specific examples[4]. The system records user actions and infers code that can be used on other similar examples. The programming by demonstration concept has been mostly adopted in robotics research. The robot is taught new behaviours through physical demonstrations of the task (e.g., teach a robot how to follow a line). Nowadays, the Microsoft Research Labs is the institution that has carried out a deeper research on PbE, having developed successful prototypes in the application domain of data wrangling, establishing a sound basis to impact several other domains like code refactoring [78]. Sometimes in literature there is a slight distinction between PbE and PbD. In PbE the user gives a prototypical product of the computer execution, for example a row in targeted results of a query, while in PbD the user performs a sequence of actions that the computer must repeat, generalizing it to be used in different data sets (an example of this can be a macro recorder). At SmartCLIDE, the consortium will explore the path opened by Gulwani and Jain [78] in treatment of data.

The Coding-by-Demonstration principle starts to be a reality explored by many universities, research centres and software companies. For example, Calinon [35] describes current approaches that apply this new paradigm for programming robots. Li et al. [129] describe a solution that enables the programming of IoT devices using mobile apps and applying the Coding-by-demonstration paradigm. Many software companies recently are using new approaches to involve end-users in the programming, especially in the domain of smart devices, such as home automation, smart phone, etc. Similar approaches, such as visual programming [83], form-based programming[5], or tangible programming [150][151], have also been widely explored. With the programming by example principle, users can directly demonstrate parts of the program behaviour and the end-user programming system builds the necessary application. A macro recorder is one of the most straightforward 'programming by example' approaches where the user simply records a sequence of actions and repeats them later at some point in time by giving appropriate commands.

Introducing the Coding-by-Demonstration principle in SmartCLIDE's development environment is expected to pave the way of a new generation development tools that will allow developers to move:

1. from coding to direct manipulation of data,
2. from procedures to specification of goals and constraints,
3. from text dump to spatial representations, and
4. from sequence to concurrency.

2.1.2 Software Paradigms: Perspective of Software Life Cycle

The present subsection shows how software paradigms have evolved from linear and sequential approaches, through evolutionary models, until reaching agile ways of developing software. Agile paradigm has gained so much relevance in the last years, that it will be further developed in its own section (Section 2.2 Agile Software Development). It is important to remark that new paradigms have not replaced the old ones, so nowadays we can find projects, even in the same organization, that follow the V-life cycle model and others that apply Extreme Programming. The evolution of software life cycle models is visualized in Figure 3.

[4] Proceedings of the 30th International Conference on Machine Learning (ICML), June 2013
[5] https://ifttt.com/

Linear and Sequential Approaches*:* Since the *Waterfall Model* was described in the early ’70s [193], introducing a set of consecutive or *linear* steps for developing software (System and Software Requirements, Analysis, Design, Coding, Testing, and Operations), several development paradigms have been described over the last 50 years. Primary evolution of waterfall was the *V-life cycle*, adopted by highly regulated sectors since it included a quality assurance layer that described a reverse waterfall process for verification and validation activities. It was conceived and created by researchers from Germany and USA [173], and adopted by the military forces of their governments in the early ’90s. When waterfall models were applied incrementally, we talked about *incremental models*. They also follow consecutive steps (Analysis, Design, Coding, and Testing), nevertheless they divide the project into smaller and independent parts or phases, where each release (called increment) provides additional functionality to the product as well as feedback to the following phase. These models, though still linear, show the need to obtain an early functionality provision to obtain feedback and, therefore, try to reduce risk.

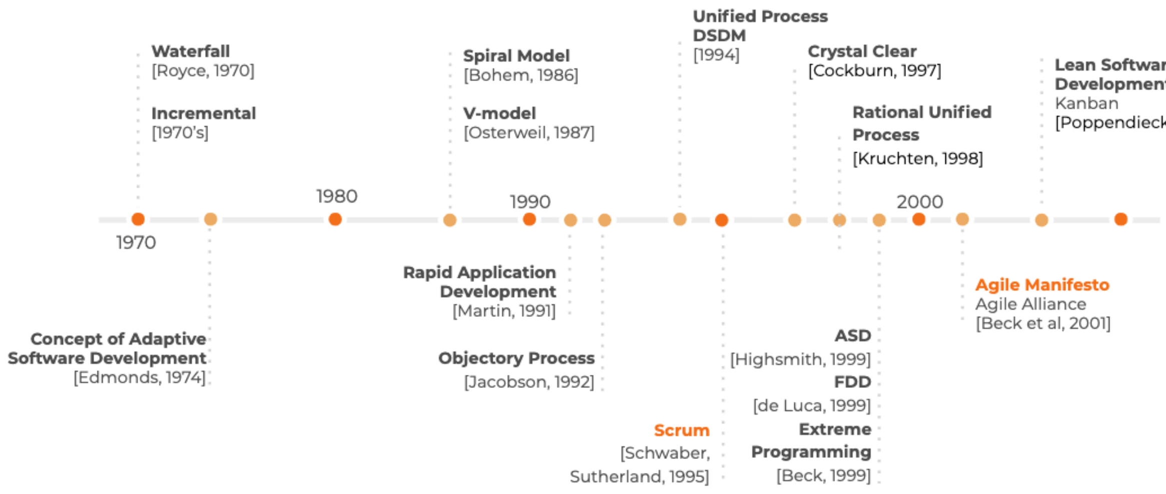

Figure 3: Timeline—from waterfall to agile

Evolutionary Approaches: Having in mind the concept of risk reduction, *evolutionary approaches* recognize uncertainty; for example, the *Spiral Model* [26] added a risk analysis phase in each iteration (also called spiral), with a total of four phases (Planning, Objectives Determination, Risk Analysis, and Development and Testing). As another alternative to the rigid waterfall model, *Rapid Application Development* [140] was proposed to deal with the flexibility of software development. Instead of defining a complete set of requirements and analysis in an early phase, it included a user design or prototype cycle phase (prototype, test, refine) that was repeated until fully understanding and satisfying the users’ needs. This model, therefore, required regular access to users.

The *Rational Unified Process (RUP)* [126] was the obtained result of a work that started looking into why software projects had failed and it went back to the spiral model. RUP evolved from the Objectory Process [97], and became an adaptable process framework rather than a single, concrete, prescriptive process. It served as a guide on how to apply effectively the *Unified Modelling Language* (UML) diagrams in the analysis, design, implementation and documentation of object-oriented systems. RUP divided the development process into four distinct phases (Inception, Elaboration, Construction and Transition). Each of its phases involved business modelling, requirements, analysis and design, implementation, testing and deployment.

2.2 Agile Software Development

Adaptive, Agile Approaches: The concept of an adaptive software development was already developed in 1974 [61], even though it wasn't formalized until 1999 by Jim Highsmith. Along with RAD, *Adaptive Software Development (ASD)* is an antecedent to agile software development. Understanding the unpredictable nature of increasingly complex systems, ASD was built from a different point of view, and it was reflected in the name of its phases (Speculate, Collaborate & Learn). It goes deeper than a change in the life cycle, as it focused its efforts on a different management style that aimed to achieve the capacity to respond to change and does not try to get everything right the very first time. Besides ASD, other processes and frameworks were developed during the '90s as a response to the long-time trend of failing projects that followed waterfall paradigms, leading to the advent of agile approaches. This decade also starts to bring concepts that will be commonly adopted by the agile community in the future years. Such is the case of *"Continuous Integration" (CI)*, or *Scrum*, that was originally invented as a process by Jeff Sutherland, and nowadays it can be considered as the most popular agile framework. The main agile frameworks invented from the last nineties to the first years of the current millennium are:

- *Dynamic Systems Development Method (DSDM)*: In 1994, the DSDM Consortium was founded as a response to the problems related to Rapid Application Development (RAD) process (mostly related to over-spending and late delivery of projects following such paradigm). DSDM is focused on delivering value to business within a budget and delivery date combining different techniques, such as timeboxing, the 80/20 rule, the MoSCoW technique to prioritize requirements, and many others.
- *Scrum* [219]: The Scrum framework was formalized and presented in 1995 by Jeff Sutherland and Ken Schwaber. It includes the concepts of iterative and incremental, along with those of adaptation, inspection and transparency, which are the three pillars of empirical process control. This framework is team-centric, empowering teams to try and learn, to plan and develop the solution, adapting to changes, rather than trying to predict the right solution the first time.
- *Crystal Clear*: The book "Surviving Object-Oriented Projects" [45] collects experiences from previous software projects (dating back to 1993) that served as the basis for the key rules of Crystal Clear. They are focused on people and interactions rather than processes (one of the main principles of every agile framework) to enhance self-managing teams of a small size. The three key properties are frequent delivery, reflective improvement and osmotic communication.
- *Feature Driven Development (FDD)* [44]: The FDD follows a 5-step process, where an overall model is developed, followed by the building of a list of features. Then each feature is planned, designed and built. These three final phases are iterative. This framework is focused on the Continuous Delivery (CD) of value, and it's client-centric.
- *Extreme Programming* [19]: The Extreme Programming framework is based on five values: simplicity, communication, feedback, courage, and respect. Moreover, it includes a set of core practices that reinforce these values (Sit Together, Whole Team, Informative Workspace, Energized Work, Pair Programming, Stories, Weekly Cycle, Quarterly Cycle, Slack, Ten-Minute Build, Continuous Integration, Test-First Programming, and Incremental Design). The main focus of this framework is the frequently release of high-quality software. Along with Scrum, is one of the most used agile frameworks.
- *Lean Software Development* [185]*:* In 1988 the concept of LEAN, Lean Manufacturing and Kanban began to grow in popularity [171], though its concept goes back to the '40s based on the Toyota Production System. However, it wasn't until fifteen years later that it was adopted in software development after describing the agile tasks as a "software Kanban system". The Kanban method allows the visualization of the workflow of the software production, facilitating the communication

and transparency to the members of the team, to keep track of the stage of the development and the status of a project.

Agile Manifesto. All the above-mentioned frameworks and experiences proved the need for a new paradigm as a response to waterfall models. As a result, *The Manifesto for Agile Software Development* [20] was published. It was a turning point in software development which brought together several of the values and principles already seen. The *four values* upon which the manifesto was signed are: (a) Individuals and interactions over processes and tools; (b) Working software over comprehensive documentation; (c) Customer collaboration over contract negotiation; and (d) Responding to change over following a plan. And the *twelve principles* that are focused on short iterative and incremental developments for delivering value to customers or end-users ; continuous feedback, planning, testing, and integration; continuous adaptability to ever changing contexts; continuous delivery of value empowerment of teams; technical excellence; quality assurance as it's built into the project; or continuous improvement.

Scrum and XP. Scrum and Extreme Programming (XP) reflect the agile manifesto's values and principles in the practices they propose. Originally, these frameworks were devoted to small teams, but over the years several scalability frameworks like Large Scale Scrum[6] or Scaled Agile Framework[7] have appeared with the aim of spreading the agile mindset over the whole organization. The most relevant elements and characteristics of Scrum are shown in Figure 4. The main characteristic of the life cycle established by Scrum is the continuous and incremental delivery of value to end-users or customers in short iterations called Sprints. Duration of Sprints may go from 5 days to 1 month, giving preference to the shortest time frame.

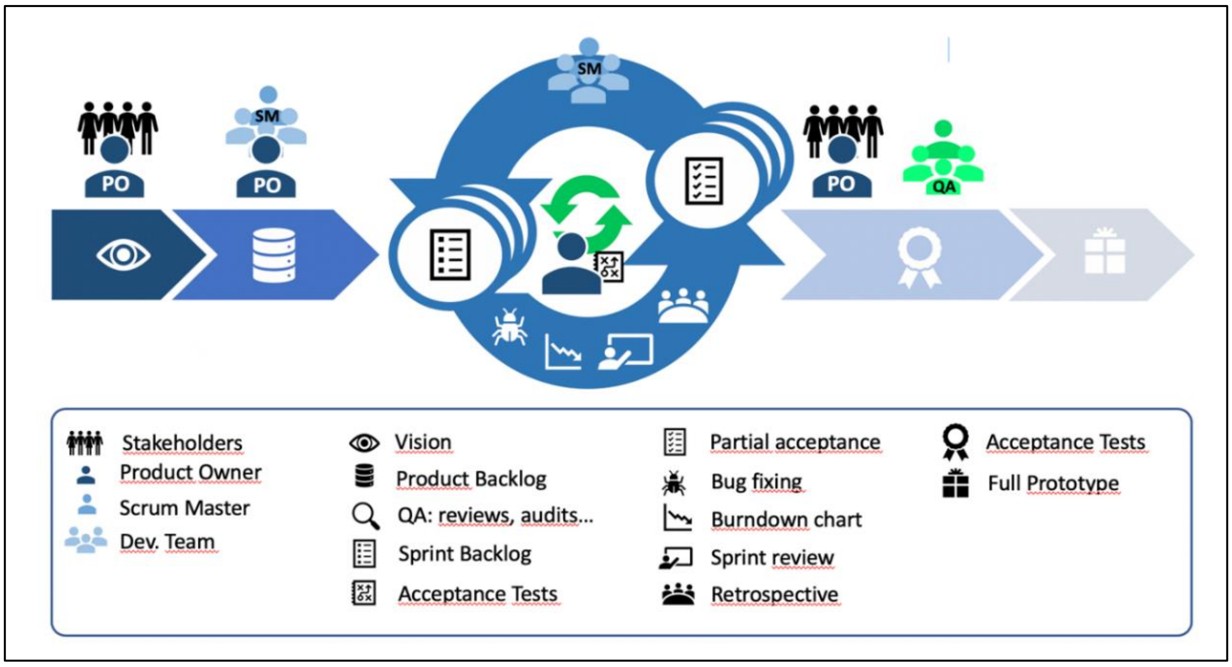

Figure 4: Scrum Framework

[6] https://less.works/less/framework/index.html
[7] https://www.scaledagileframework.com/

The main **actors** involved in the process are:

- **Business Stakeholders**, group of target users for whom value is delivered by the Development Team.
- **Product Owner (PO)**, is the person who "owns" the product. She is responsible for identifying user requirements in the form of **User Stories** and defining **Acceptance Criteria**, which constitute the so called **Product Backlog**. She is also responsible for **prioritizing** the user stories aiming at maximizing the business value they provide. Owns and shares the vision of the product, and product roadmap.
- **Development Team (DT).** Multidisciplinary, autonomous, high-performing teams. Ideally, the team is end-to-end responsible for the delivery and operation of the product, so they do not only code, but also perform QA, security and operations activities.
- **Scrum Master (SM).** Person that guarantees that the Scrum framework is properly applied. She guides the team through the different stages of the framework and challenges the team to reach autonomy and high-performance. The Scrum Master removes impediments for the Development Team to flow towards the continuous delivery of value.

The main **items** of Scrum are:

- **Product Backlog**: set of user stories and acceptance criteria that conform the main features of the product/service to develop.
- **Sprint Backlog**: set of user stories and acceptance criteria that fit into a Sprint and deliver a specific business value.
- **Daily stand-ups**: daily meetings for the development team to organize the daily work. Impediments are identified in these sessions and communicated to the Scrum Master or Product Owner so they can be removed.
- **Sprint Planning:** session where the DT, PO and, eventually, stakeholders define the scope of a Sprint.
- **Sprint Review**: session where the team shows the value generated within the sprint to the PO and the Stakeholders. PO and stakeholder may also accept or reject the developed User Stories, provide feedback and share their vision on the priorities for the next sprints.
- **Retrospectives**: sessions where the team, SM and PO evaluate how the Sprint was performed and improvement actions are identified for the next sprints.
- **Information radiators**: visual artefacts to manage the backlog (e.g. Scrumban boards) or visualize the effectiveness of the development team (e.g. burndown charts).

In opposition to Scrum, that establishes a work framework, **Extreme Programming** (XP) is based upon a similar set of **values** (*communication, simplicity, feedback, respect and courage*) and **principles** (*rapid feedback, assumed simplicity, incremental changes, embracing change and quality work*). **Practices** proposed by XP are *The Planning Game, Small Releases, the use of Metaphors to share the vision of the project, Simple Designs, Testing Automation, Continuous Refactoring of Code, Pair Programming, Collective Ownership of Code, Continuous Integration, the 40-hour week, On-site Customer (always) and the use of Coding Standards*.

Since the first time the software business heard of DevOps in 2008 [54], it has evolved really fast turning the buzzword into a reality that is transforming digital business all over the world. The philosophy behind DevOps aims at demolishing the walls that create operational silos in business, development and operations/infrastructures creating an environment where valuable work continuously flows, there is a continuous feedback up and downstream, and continuous improvement is a common practice

[112]. Among many other practices, the full autonomy and end-to-end responsibility of software development teams can be considered the cornerstone of DevOps. These practices mean that software creation teams will have the full responsibility to take an application to production: from the specification of requirements/user-stories to its deployment in a server. With this in mind, SmartCLIDE shall focus on DevOps organisations offering assistance at all the stages of the software creation life cycle, namely: Specification and Planning, Creation, Verification, Packaging, Release, Configuration and Monitoring. The main practices that back autonomy and responsibility are the creation of multidisciplinary teams (including staff with business and operations knowledge), continuous communication, an extreme automatization of processes and the existence of a solid common knowledge base. In this State-of-the-Art we will set the focus on the extreme automatization and the common knowledge base since they are the elements that are fully related to work to be developed in SmartCLIDE:

- ***Extreme Automatization:*** The automation of a big part of tasks, such as code inspections; the creation, execution and reporting of tests; the creation of host environments; or the integration and deployment of the developed applications along with virtualization and containerization technologies has allowed the software creation teams to automate processes dealing with the setup of environments and deployment of applications, giving them full control over the software delivery. Therefore, operations are becoming a powerful enabler of the continuous flow in the software delivery value stream by providing self-service automatisms [112] to the software creation teams.
- ***Solid Common Knowledge Base:*** DevOps considers that all the knowledge required for the creation of software must be concentrated in a single knowledge repository, providing full traceability of the creation process, and being source code the primary documentary item.

Related to this life cycle, there are two main features that will be provided by SmartCLIDE:

- Integration with build tools for packaging, virtualization and containerization tools to handle images of environments and perform fast deployments. This enables the extreme automatization concept of DevOps.
- Integration of autonomous AI-based Smart Services within the DevOps loops, so end-users will be able to reuse already existing user stories or acceptance criteria, when and where more intensive testing will be required (by monitoring the verification stage), or when is the best moment to build and transport an application to a determined environment.

Waterfall vs. Agile. *The Standish Group* publishes a comparison of success and failure rates between Waterfall and Agile projects. The most recent results (2013-2017) show that, statistically, Agile projects are twice more likely to succeed and three times less likely to fail than waterfall projects. Besides the project approach, the size of a project has a great impact on success, being large projects more likely to fail. These two project factors, size and approach, combined together have the greatest impact on the success of a project. There is, however, room to improve, and the trend is to continue to break big projects down into smaller ones while using agile methodologies as a way to increase success and reduce risk and failure. Nowadays, according to the *13th annual State of Agile™ report,* the top three reasons for adopting agile is to accelerate software delivery (74%), to enhance the ability to manage changing priorities (62%) and to increase productivity (51%). Also, tendency shows that reasons are moving towards improving team morale and reducing project costs.

Short Market Analysis. According to the *13th Annual State of Agile™ report* (see Figure 5), the top three trends in agile that are still continuing are: (a) Scrum as the most widely-practiced agile methodology, or a hybrid; (b) the importance of the organization's culture while adopting agile; and (c) the importance of the DevOps transformation. As for the latter, the core practices of CI/CD along with increasing Test Automation have become very important as a way to transform the culture between the Development and Operations organizations. In this sense, SmartCLIDE will include the practices of CI/CD and automated testing. In this survey, it is stated that a DevOps initiative is either underway or already planned for a 73% of respondents. Improved quality and faster delivering of software are the most critical measures of DevOps success, and thus a 42% of respondents stated that DevOps transformation is "Very Important". Regarding agile methodologies, Scrum and Scrum/XP Hybrid are the most common ones.

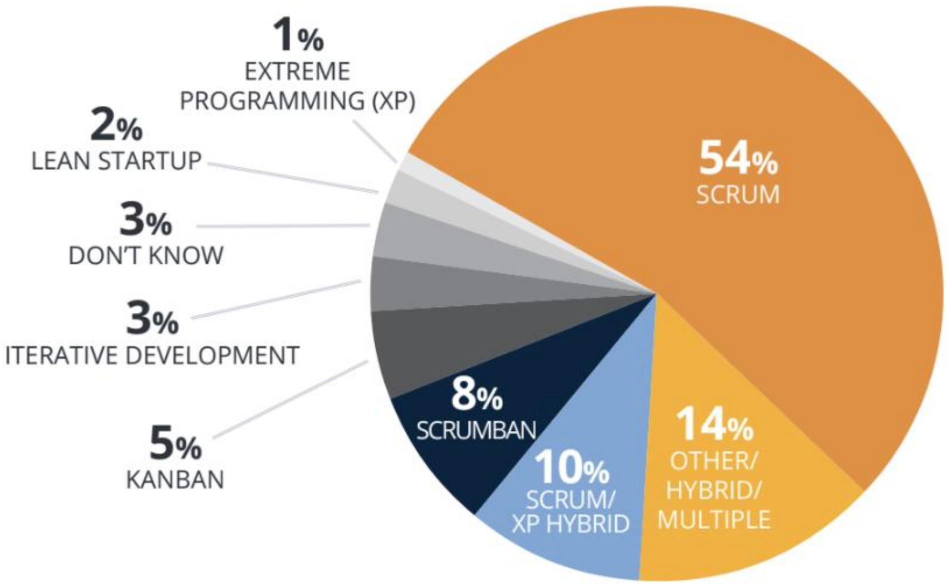

Figure 5: Market Analysis on Top Agile Methodologies

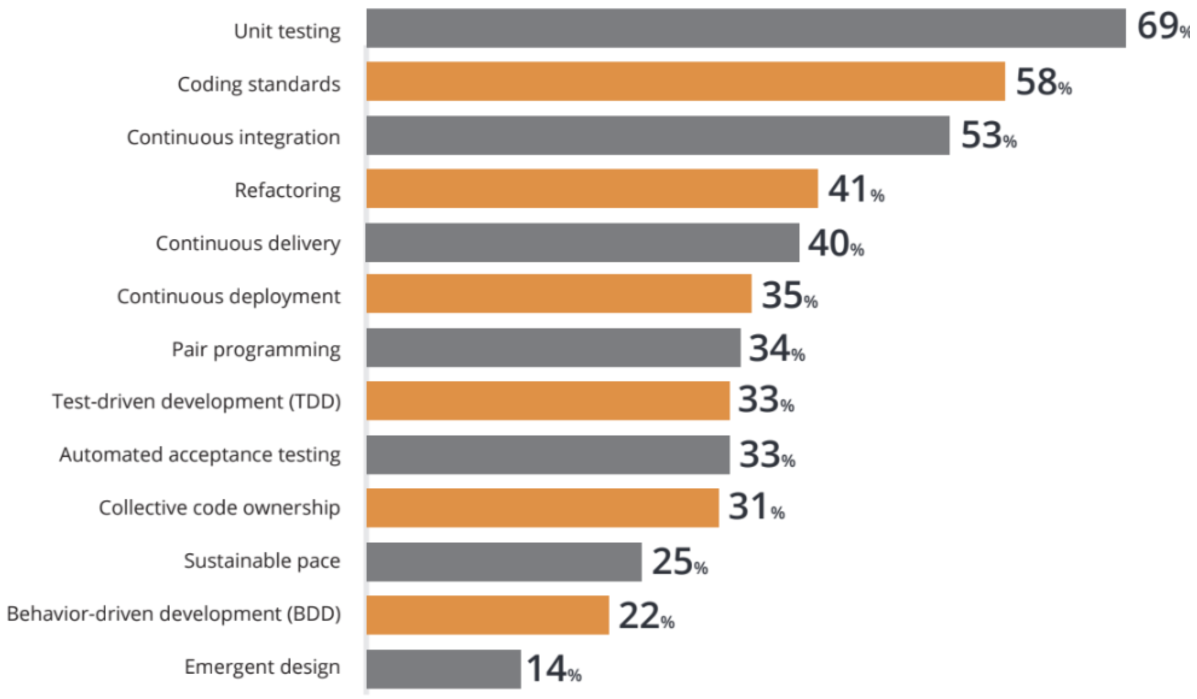

Figure 6: Market Analysis on Top Agile Engineering Practices

Among the most popular agile techniques employed (see Figure 6), *short iterations* are in the top 5, which is also one of the concepts that SmartCLIDE will adopt. In addition to this, some of the top engineering practices according to the survey (*unit testing, coding standards, continuous integration, refactoring, continuous delivery and continuous deployment*) are also to be considered. Currently, the main platform integrating tools that support most of the former agile practices is GitLab, either directly or by integrating different specific plugins. Finally, SmartCLIDE will be designed to fit the agile mindset along with the DevOps philosophy. Thus, it supports the development of software along its full life cycle (specification, development, testing, deployment and runtime).

2.3 Integrated Development Environments

An Integrated Development Environment (IDE) is an application that provides a set of tools and functionalities to help developers in their daily work. The most common features, such as compiling, debugging, version control, and navigating the data structure, help a developer to quickly perform actions without having to switch between applications. Seamless integration of these features maximizes productivity by providing similar user interfaces (UIs) for related components and reduces the time required to learn a programming language. Today, Wikipedia lists more than 100 different IDEs[8]. Some IDEs are dedicated to a single language, such as RStudio[9] or PyCharm[10]. Others support multiple languages, such as Eclipse[11] or VisualStudio[12]. For years, each IDE that wanted to support a specific language had to build its own connector to a specific compiler, build its own code wizard, its own debugger, and so on.

Rich Client Platform. In 2005, the Eclipse platform, through the flexibility of its plugin architecture, attracted users other than software developers. The platform team isolated programming language-specific plugins from plugins that could be used to build just about any client application. This basic platform is called the Rich Client Platform (RCP)[13]. This initiative opened the world of IDE to other communities such as the modelling community[14], the geo-localization community[15], and the scientific community[16]. In other words, we can say that by integrating domain tools on the same platform, the IDE "in the broadest sense", attracts domain communities. This contributes, still today, to the success and popularity of a platform such as Eclipse.

Language Server Protocol. Let's get back to the IDEs for developers. Ten years ago, part of the team that built the Eclipse IDE for IBM joined Microsoft. Instead of rewriting new code to support a programming language in VisualStudio, Microsoft's IDE, they imagined a component that could be used by any IDE to support a specific language; the Language Server Protocol (LSP) was born. "*The Language Server Protocol is an open, JSON-RPC-based protocol for use between source code editors or integrated development environments and servers that provide programming language-specific features. The goal of the protocol is to allow programming language support to be implemented and distributed independently of any given editor or IDE.*"[17, 18]. Today, the LSP is an open standard. It is adopted by many IDEs and

[8] https://en.wikipedia.org/wiki/Integrated_development_environment
[9] https://en.wikipedia.org/wiki/RStudio
[10] https://en.wikipedia.org/wiki/PyCharm
[11] https://en.wikipedia.org/wiki/Eclipse_(software)
[12] https://en.wikipedia.org/wiki/Microsoft_Visual_Studio
[13] https://wiki.eclipse.org/Rich_Client_Platform
[14] https://www.eclipse.org/modeling/
[15] https://www.eclipse.org/locationtech
[16] https://science.eclipse.org/
[17] https://en.wikipedia.org/wiki/Language_Server_Protocol

editors[19] such as Eclipse CHE, VisualStudio, Emacs, IntelliJ. The LSP community currently supports 91 programming languages[20]. This new approach allows IDE vendors to focus their efforts on the user experience (UX) and LSP developers to run on many IDEs.

Application Life Cycle Management. When we talk about IDE, we cannot avoid mentioning Application Life cycle Management (ALM)[21]. If an IDE helps developers in their daily work, an ALM environment helps an entire team. It is interesting to note that an ALM has the same approach as an IDE: improving the integration of tools to improve the productivity of its users. Most of the key players in IDEs are also key players in ALM, such as IBM with the Rational solution for collaborative life cycle management, Microsoft with Team Foundation Server or GitLab and Tuleap as open source solutions. We mention ALM in this section because this type of tool emphasizes an important need for teams, i.e., collaboration.

Virtual Machine Farms. A few years ago, when organizations could build and manage Virtual Machine (VM) farms, some IDEs began to be deployed on a remote VM rather than on a local machine. This approach has several advantages:

- Security: Code is no longer stored on the developer's machine;
- Performance: Performance is no longer dependent on the developer's machine but on the performance allocated to the virtual machine.
- Bandwidth: The VM can stay close to the database, for example, to improve communication performance between the IDE and its target. It no longer depends on the bandwidth of your local WiFi, LAN or DSL.
- Update: You don't have to wait for your developers to update a specific machine, you can simply update their VM for them.

Cloud-Native IDEs. Cloud-Native IDEs bring a new dimension to the development. Obviously, one obtains the advantages of an IDE running on a powerful server and more:

- Packaged as lightweight containers: Cloud-native IDEs are a collection of independent and autonomous services that are packaged as lightweight containers.
- Developed with best-of-breed languages and frameworks: Each service of a cloud-native IDE is developed using the language and framework best suited for the functionality.
- Designed as loosely coupled microservices: Services that belong to the same feature discover each other through the application runtime. They exist independent of other services.
- Centred around APIs for interaction and collaboration: Cloud-native services use lightweight APIs that are used based on protocols such as REST (Representational State Transfer).
- Isolated from server and operating system dependencies: Cloud-native IDEs don’t have an affinity for any particular operating system or individual machine. They operate at a higher abstraction level.
- Deployed on self-service, elastic, cloud infrastructure: Cloud-native IDEs are deployed on virtual, shared and elastic infrastructure.

The SmartCLIDE approach uses such an architecture to take advantage of these possibilities. Not only will it use this architecture, but it will also extend existing Cloud-native IDEs to focus its work on the innovative added values of the SmartCLIDE project.

18 https://langserver.org/
19 https://langserver.org/#implementations-client
20 https://microsoft.github.io/language-server-protocol/implementors/servers/
21 https://en.wikipedia.org/wiki/Application_Life cycle_management

3 Development of Microservice Applications

3.1 MicroService Architectures

Microservices gained popularity the last decade towards building complex and larger applications that can be segregated and handled as a compilation of smaller services. A microservices architecture is basically an emerging development methodology wherein you can fragment a single application into a series of smaller services. Microservices are developed around business capabilities, and as such are independently deployable with automated deployment mechanism. Related DevOps technologies can be used to help these automations. Each microservice is executing in its own process and interacting with lightweight mechanisms with other microservices or applications. This isolation and independence results in a bare minimum of management of these services, which are usually being built in different programming languages and employ different data storage technologies according to each element requirement. A microservices architecture demonstrating the fragmentation into smaller autonomous services versus the older monolithic architecture paradigms is illustrated in Figure 7. Below, we discuss the main features and benefits brought by microservice architectures.

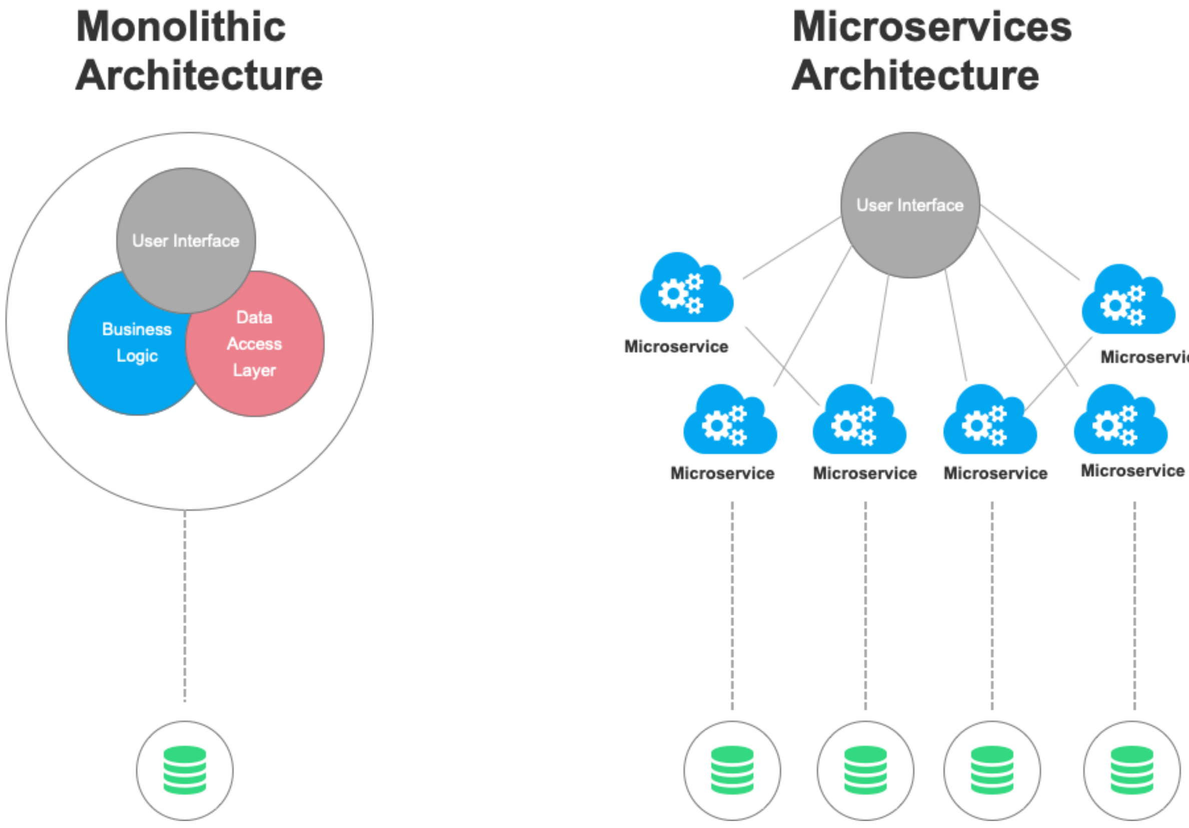

Figure 7: Monolithic vs. Microservices architecture conceptualization

Dynamic Scalability. Based on the development of small isolated components, developer teams can easily scale up or down based on the requirements of a specific element. The flexibility of microservices lets a system expand fast without requiring a significant increase in resources. A monolithic architecture would require scaling the whole application. Each module in microservices can scale independently through: (a) X-axis scaling, by cloning with more memory or CPU; and (b) Z-axis scaling, by size using shading.

Technology Flexibility. This refers to the microservice architecture flexibility on its technology stack that leads to eliminating the constraints of vendor or technology lock-in and platform dependency. Each microservice can be built up using the software stack required for the specific element. Language, framework, data sources or any other dependencies required can be provided from a container without affecting the whole application design or the communication between the microservices in the ecosystem.

Easier and shorter development cycles. These are achieved through the important feature of agility that further leads to productivity and speed, smaller project development, ease of building and maintaining apps, that are independently DURS (Deployed, Updated, Replaced & Scaled). Since each microservice is a separate project, professionals can get involved in the process more easily because they do not have to study the system as a whole and they can work only on their part. Decomposing the monolithic structure into separate services, leads to team decomposition into more small engineering teams that work independently which increases agility. The modern Agile approach is tightly connected with practices as DevOps concepts, continuous integration (CI), and continuous deployment (CD). All of these practices allow for faster deployment, problem-solving, and time to market. This type of agility when combined with CI / CD tools, like Jenkins, and their underlying pipeline configuration capabilities, results in faster and smaller project development life cycle procedures. Compared to a microservices architecture, a monolithic architecture hampers the Agile and DevOps processes because of its tight connections between each and every component.

Fault Isolation. Small isolated microservices can affect less the overall ecosystem when failing. A monolithic architecture is rigid when it comes to replacing functionalities or making changes. Small changes in one place can cause ripple effects, bugs and errors in the entire system due to the extreme coupling. As such microservices architecture improves replaceability and upgradeability of the system.

Reduced Downtime / Quick Response-time. Developers and DevOps have the ability to use another service when components fail, and application continues its work independently. With the use of related technologies that provide virtual servers, containers, pods and clustering this architecture offers reduced response downtime.

3.2 Advancements in Microservice Software Stack

To fulfil the above defined demanding market requirements, SmartCLIDE can leverage the provided functionalities of a great number of available Software Stack technologies. Microservices can be implemented in a variety of Languages, Frameworks and tools. Java, Node.js, .Net, Python are just a portion of the Languages that support the microservice architecture with Java, with its huge palette and proactive community, standing above most language ecosystems.

3.2.1 Java

Java with its annotation syntax and well-established community is probably the best choice to go. Java EE standards like JAX-RS, JPA, and CDI are suited for microservices. Solid Frameworks also focus on developing Microservices architecture. Spring Boot[22], Jhipster[23], Dropwizard[24], Restlet[25], Spark[26],

[22] https://spring.io/projects/spring-boot
[23] https://www.jhipster.tech/
[24] https://www.dropwizard.io/
[25] https://restlet.talend.com/
[26] http://sparkjava.com/

Quarkus[27], Vert.x[28], Project Helidon[29], Micronaut[30], ServiceMix[31], Istio[32], Fn Project[33], Openwhisk[34], OpenFaas[35] are just a portion of the currently available tools. On top of the various technologies, frameworks and platforms have emerged the last years focusing on microservices architecture, along with the rapid changes that occurred on JakartaEE[36] as well as Eclipse microprofile[37]. Open Liberty[38], WildFly[39] and Payara Server[40] are some examples that support the development of cloud-native Java microservices.

On top of that GraalVM[41] is gaining more popularity as a Java VM and JDK based on HotSpot/OpenJDK, implemented in Java.

Spring Boot. Spring Boot is an open source Java-based framework used to create microservices. It is developed by Pivotal Team and is used to build stand-alone and production ready spring applications. Spring Boot provides a good platform for Java developers to develop a stand-alone and production-grade spring application that can run as standalone. It is easy to get started with minimum configurations without the need for an entire Spring configuration setup.

Jhipster. Jhipster combines three very successful frameworks in web development: Bootstrap[42], Angular[43], and Spring Boot. Bootstrap was one of the first dominant web-component frameworks. Its largest appeal was that it only required a bit of HTML to work.

Dropwizard. Dropwizard is a Java framework for developing ops-friendly, high-performance, RESTful web services. Dropwizard pulls together stable, mature libraries from the Java ecosystem into a simple, light-weight package. Dropwizard has *out-of-the-box* support for sophisticated configuration, application metrics, logging, operational tools, and much more, allowing the shipping of a *production-quality* web service in the shortest time possible.

Restlet. The Restlet Framework helps Java developers build web APIs that follow the REST architecture style. Adopted and supported by a large community of Java developers, Restlet Framework benefits from numerous resources available all over the Internet. Fully open source, it is freely downloadable and can be used under the terms of the Apache Software License. Thanks to Restlet Framework's powerful routing and filtering capabilities, unified client and server Java API, developers can build secure and scalable RESTful web APIs. It is available for all major platforms (Java SE/EE, Google AppEngine, OSGi, GWT, Android) and offers numerous extensions to fit the needs of all developers. APIs built using Restlet Framework can be deployed on any platform but close integration with APISpark, one of the

[27] https://quarkus.io/
[28] https://vertx.io/
[29] https://helidon.io/
[30] https://micronaut.io/
[31] https://servicemix.apache.org/
[32] https://istio.io/
[33] https://fnproject.io/
[34] https://openwhisk.apache.org/
[35] https://docs.openfaas.com/
[36] https://jakarta.ee/
[37] https://microprofile.io/
[38] https://openliberty.io/
[39] https://wildfly.org/
[40] https://www.payara.fish/
[41] https://www.graalvm.org/
[42] https://getbootstrap.com/
[43] https://angular.io/

available Platform-as-a-Service dedicated to web APIs, enables developers to save time in documenting and creating SDKs for their APIs, while maintaining the flexibility of working with the framework.

Spark. Spark is a free and open-source software web application framework and domain-specific language written in Java. It is an alternative to other Java web application frameworks such as JAX-RS, Play framework[44] and Spring MVC. It runs on an embedded Jetty[45] web server by default but can be configured to run on other webservers. Spark Framework is a simple and expressive Java/Kotlin web framework DSL. Sparks provides an alternative for Kotlin/Java developers that can develop their web applications with minimal boilerplate by using Spark's declarative and expressive syntax.

Quarkus. Quarkus programming model builds on top of proven standards such as Eclipse MicroProfile or leading frameworks in a specific domain such as Eclipse Vert.x. Coding is based on well-established Java EE and Jakarta EE standards. Dependency injection solution is based on CDI and JAX-RS annotations can be used to define the REST endpoints. JPA annotations also map your persistent entities and JTA annotations to declare the transaction boundaries. Eclipse MicroProfile is used to configure and monitor any application application. Vert.x, Apache Camel and more can be integrated with Quarkus.

Eclipse Vert.x. Vert.x contains several different components designed to make it easier to write reactive applications in a range of different languages. Vert.x is highly modular and modules that only required can be used and nothing more. Vert.x is a library and not a restrictive container, thus applications are not limited in the components provided by Vert.x. It is based on the foundations of the asynchronous and event-based application design. Vert.x uses low level IO library Netty[46]. It supports a variety of languages along with Java and it is single threaded and asynchronous which fulfill all requirements for reactive applications.

Project Helidon. Oracle has introduced its open-source framework Project Helidon, a collection of Java libraries designed for creating microservices-based applications. Helidon is a collection of libraries running on a fast Netty core. Helidon supports Eclipse MicroProfile and provides familiar APIs like JAX-RS, CDI and JSON-P/B. Helidon MicroProfile implementation runs on Helidon fast Reactive WebServer which runs on top of Netty. It is lightweight, flexible and reactive which makes it ideal for microservices. It supports health checks, metrics, tracing and fault tolerance and integrates with Prometheus[47], Jaeger[48] / Zipkin[49] for tracing and Kubernetes[50] for orchestration.

Micronaut. Micronaut is a Modern Microservice Framework for JVM. It is a full-stack framework for building modular and easily testable microservice applications that offers fast startup time and low memory consumption. Dependency Injection and Aspect-Oriented Programming runtime are not using reflection as such Micronaut applications run easier on GraalVM. The server is written in Java and also supports Java, Groovy and Kotlin language.

ServiceMix. Apache ServiceMix is a flexible, open-source integration container that unifies the features and functionalities of Apache ActiveMQ[51], Camel[52], CXF[53], and Karaf[54] into one runtime platform. It provides a complete, enterprise ready ESB exclusively powered by OSGi.

[44] https://www.playframework.com/
[45] https://www.eclipse.org/jetty/
[46] https://netty.io/
[47] https://prometheus.io/
[48] https://www.jaegertracing.io/
[49] https://zipkin.io/
[50] https://kubernetes.io/
[51] http://activemq.apache.org/

Istio. Istio can be used to connect, secure, control, and observe services that can be deployed on a hybrid multi-cloud environment to help DevOps teams to manage the microservices developed and maintained from engineering teams. With Istio[55] you can basically reduce the complexity of these deployments and eases the strain on your development teams. It is a completely open source service and it includes APIs that let the integration with logging platforms, or telemetry or policy systems.

Fn Project. Oracle introduced Fn Project as an open source container-native serverless computing framework offered through Oracle Cloud Platform but is also available on GitHub for deployment on other platforms or on-premises solutions. It's event-driven, Functions-as-a-Service (FaaS)[56] compute platform and supports many programming languages, including Java.

Openwhisk. OpenWhisk was initially developed by IBM with contributions from RedHat, Adobe, and others. OpenWhisk is the core technology in IBM Cloud Functions. Apache OpenWhisk is currently an open source, distributed Serverless platform that executes functions (fx) in response to events at any scale. OpenWhisk manages the infrastructure, servers and scaling using Docker containers. The OpenWhisk platform supports a programming model in which individuals can write functional logic (called Actions), in any supported programming language, that can be dynamically scheduled and run in response to associated events (via Triggers) from external sources (Feeds) or from HTTP requests. The project includes a REST API-based Command Line Interface (CLI) along with other tooling to support packaging, catalogue services and many popular container deployment options. Since Apache OpenWhisk builds its components using containers it easily supports many deployment options both locally and within Cloud infrastructures. Options include many of today's popular Container frameworks such as Kubernetes, OpenShift and Mesos.

OpenFaas. OpenFaaS makes it easy to deploy event-driven functions and microservices to Kubernetes without repetitive, boiler-plate coding. Package of code or an existing binary in a Docker image is available to get a highly scalable endpoint with auto-scaling and metrics.

JakartaEE. Java EE has been a major platform for mission-critical enterprise applications. In order to accelerate business application development for a cloud-native world, leading software vendors collaborated to move Java EE technologies to the Eclipse Foundation where they started evolving under the Jakarta EE brand. Jakarta EE is a set of specifications, extending Java SE 8 with specifications for enterprise features such as distributed computing and web services. Java EE applications can now run on runtimes, that can be microservices or application servers, which handle transactions, security, scalability, concurrency and management of the components it is deploying.

Eclipse Microprofile. The microprofile.io[57] community is a semi-new community dedicated to optimizing the Enterprise Java mission for microservice based architectures. As of the 18th of February 2020, Eclipse Microprofile version 3.3 is available which makes Eclipse Microprofile a stable choice to invest on any architecture design and software decision. Based on MicroProfile's time-boxed release process, this is an incremental release that includes updates to various aspects of Microprofile. MicroProfile Config

52 https://camel.apache.org/
53 http://cxf.apache.org/
54 http://karaf.apache.org/
55 https://istio.io/
56 https://github.com/fnproject/docs/blob/master/fn/general/introduction.md
57 https://microprofile.io/

1.4[58], MicroProfile Fault Tolerance 2.1[59], MicroProfile Health 2.2[60], MicroProfile Metrics 2.3[61], and MicroProfile Rest Client 1.4.[62] and more.

Open Liberty. Open Liberty is one of the most flexible server runtime available for Java. It has been open sourced and it is mainly maintained from IBM following its predecessor WebSphere Liberty[63]. It has been created for building cloud-native apps and microservices while running only the minimum of requirements. Open Liberty is fast to start up with a low memory footprint and live reload for quick iteration. It is very simple to add and remove features from the latest versions of MicroProfile and Jakarta EE, since it fully supports containerization and deployment on any Kubernetes cloud. It required the minimum of migration and offers traceability for microservices and great support for logging when used along with other related tools such as Logstash[64], Prometheus[65], Graphana[66] and more.

WildFly. RedHat's WildFly is probably the most widely adopted JavaEE and now JakartaEE developer community. Formerly known as JBoss application server, is an application server authored by JBoss is developed by Red Hat. WildFly is written in Java and implements the Java Platform, Enterprise Edition (Java EE) specification. It runs on multiple platforms. WildFly is free and open-source software, subject to the requirements of the GNU Lesser General Public License (LGPL), version 2.1. WildFly is JakartaEE certified and along with Open Liberty, Payara Server and Eclipse Glassfish are both Jakarta EE 8 full platform Compatible Products as well as Web Profile Compatible[67]. RedHat previously had released WildFly Swarm[68] and Thorntail[69] whom functionalities have been totally included in the WildFly latest releases.

Payara Server. Payara Server is an open-source application server derived from GlassFish Server Open Source Edition. It supports the migration of existing Jakarta EE applications into the cloud with the Payara Platform, or the building of new, cloud-native applications on public cloud. It offers out-of-the-box support for containerization and deployment on Kubernetes. Payara has direct contribution to the JakartaEE working group and is an Eclipse Foundation Solution Member which makes Payara Server an up to date solution both as a Platform or an Eclipse MicroProfile compatible solution.

GraalVM. GraalVM is gaining more popularity as a Java VM and JDK based on HotSpot/OpenJDK, implemented in Java. Many of the already mentioned solutions run on top or along with GraalVM and leverage all the additional components offered from this solution. GraalVM is a universal virtual machine for running applications written in JavaScript, Python, Ruby, R, JVM-based languages like Java, Scala, Groovy, Kotlin, Clojure, and low-level VM-based languages such as C and C++.

[58] https://github.com/eclipse/microprofile-config/releases/tag/1.4
[59] https://github.com/eclipse/microprofile-fault-tolerance/releases/tag/2.1
[60] https://github.com/eclipse/microprofile-health/releases/tag/2.2
[61] https://github.com/eclipse/microprofile-metrics/releases/tag/2.3
[62] https://github.com/eclipse/microprofile-rest-client/releases/tag/1.4.0
[63] https://developer.ibm.com/wasdev/websphere-liberty/
[64] https://www.elastic.co/logstash
[65] https://prometheus.io/
[66] https://grafana.com/
[67] https://jakarta.ee/compatibility
[68] https://wildfly.org/news/2015/05/05/WildFly-Swarm-Released/
[69] https://thorntail.io/

3.2.2 Python

Python also supports microservices. Being an extremely high-level language is a fast and easy way to develop microservices. A broad range of Python microservices frameworks also emerged. Flask[70], Falcom[71], Bottle[72], Nameko[73], CherryPy[74] are some frameworks worth mentioning.

Flask. Flask is a micro web framework written in Python. It does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.

Falcon. Falcon is a WSGI[75] library for building speedy web APIs and app backends. It is extremely easy to build HTTP APIs in comparison to other frameworks based on the dependencies required from other frameworks. It has a clean design that embraces HTTP and the REST architectural style which make it a reliable, high-performance option for building large-scale app backends and microservices.

Bottle. Bottle is a WSGI micro web-framework for the Python programming language. It is lightweight and is distributed as a single file module with no dependencies other than the Python Standard Library. The same module runs with Python 2.7 and 3.x. It offers request dispatching (routes) with URL parameter support, templates, a built-in web server and adapters for many third-party WSGI/HTTP-server and template engines.

Nameko. Nameko is a microservices framework for Python that features: (a) Advanced Message Queuing Protocol[76], Remote Procedure Call and Events (pub-sub); (b) HTTP GET, POST & web sockets; (c) CLI for easy and rapid development; and (d) Utilities for unit and integration testing

CherryPy. CherryPy is a Python web Framework that allows building web applications in much the same way they would build any other object-oriented Python program. This results in smaller source code. CherryPy can be a web server itself or one can launch it via any WSGI compatible environment. It does not deal with tasks such as templating for output rendering or backend access. The framework is extensible with filters, which are called at defined points in the request/response processing.

3.2.3 Node.JS

Node.JS has become extremely popular the last 5 years - many enterprises currently use microservices Node.JS. Being built on top of Google’s V8 JavaScript framework, it is being extremely fast for IO and CPU tasks and requires a minimum amount of resources. Many industries focusing on IoT and Mobility highly invest in Node JS.

[70] https://palletsprojects.com/p/flask/
[71] https://falcon.readthedocs.io/en/stable/
[72] https://bottlepy.org/
[73] https://www.nameko.io/
[74] https://cherrypy.org/
[75] https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface
[76] https://en.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol

3.3 Related Technologies

3.3.1 Containerization

As flexibility is one of the key benefits of Microservices Architectures, microservices usually are deployed in containers. Containers such as Docker, package microservices along with their required dependencies. As such, Microservices run completely independently on any infrastructure, giving also the flexibility to organizations to easily migrate among datacentres, cloud services, etc.

Docker. Docker[77] is a set of platforms as a service (PaaS[78]) products that use operating system-level virtualization to deliver software in packages called containers. A container is a standard unit of software that packages up code and all its dependencies, so the application can be ported quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings. All containers are run by a single operating-system kernel and are thus more lightweight than virtual machines. The software that hosts the containers is called Docker Engine[79].

3.3.2 Orchestration

Given the independence and isolation provided by containers, orchestration tools have been raised to manage this container ecosystem. A huge list of available tools is currently available that provide a variety of functionalities both on-premises and as cloud container clustering services. These tools provide automated container deployment, scaling, and management of containerized applications. These functionalities along with the use of pods, clustering, load balancing and horizontal scaling results in dynamic scaling that can be achieved for a specific element, microservice or container. A huge list of available tools is currently available that provide a variety of functionalities. Most widely used are Docker Swarm[80], Kubernetes[81], RedHat Open Shift[82] and Apache Mesos[83].

Docker Swarm. Docker Swarm or simply Swarm is an open-source container orchestration platform and is the native clustering engine for and by Docker. Any software, services, or tools that run with Docker containers can also run in Swarm. Also, Swarm utilizes the same command line from Docker that makes it more easily adopted because of many automated configurations in comparison to Kubernetes.

Kubernetes. Kubernetes is an open-source platform created by Google that provides automated container deployment, scaling, and management of containerized applications. These functionalities along with the use of pods, clustering, load balancing and horizontal scaling results in dynamic scaling that can be achieved for a specific element, microservice or container. Kubernetes is the most widely adopted orchestration tool that has been integrated due to its open-sourced nature to many corporate-ready cloud solutions.

[77] https://www.docker.com/
[78] https://en.wikipedia.org/wiki/Platform_as_a_service
[79] https://docs.docker.com/engine/
[80] https://docs.docker.com/engine/swarm/
[81] https://kubernetes.io/
[82] https://www.openshift.com/
[83] http://mesos.apache.org/

RedHat Open Shift. OpenShift is a family of containerization software developed by Red Hat. Its flagship product is the OpenShift Container Platform which is an on-premises platform as a service built around Docker containers orchestrated and managed by Kubernetes on a foundation of Red Hat Enterprise Linux. It offers automated container life cycle management, fast building and deployment and create compatibility due to its use of the well-established Docker and Kubernetes.

Apache Mesos. Apache Mesos is an open-source project to manage computer clusters. Mesos is built using the same principles as the Linux kernel, only at a different level of abstraction. Mesos kernel runs on every machine and provides applications (e.g., Hadoop[84], Spark[85], Kafka[86], Elasticsearch[87]) with APIs for resource management and scheduling across entire data-centre and cloud environments. Apache Mesos has also support for containers, high availability, linear scalability and runs on any platform.

3.3.3 Cloud-based Orchestration

Same functionalities as on-premises orchestration can also be provided as cloud container clustering services. Most widely used are Google Container Engine[88], AWS Elastic Kubernetes Service[89], AWS Elastic Compute Cloud Container[90] and RedHat OpenShift online[91].

Google Container Engine. Google Container Engine formerly known as GKE (Google Kubernetes Engine) is part of the Google Cloud Platform that is a suite of cloud computing services. It is an enterprise-grade platform for containerized applications, including stateful and stateless, AI and ML, Linux and Windows, complex and simple web apps, API, and backend services. It is based on Kubernetes, thus it offers all the key features provided by Kubernetes such as pods, clustering, autoscaling, etc.

AWS EKS Service. Amazon Elastic Kubernetes Service (Amazon EKS) is a fully managed Kubernetes service. It runs upstream Kubernetes and is certified Kubernetes conformant which makes all open source tooling available from the community. EKS runs across multiple AWS availability zones which results to limited downtime.

AWS EC2 container. Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud. It provides the functionality of launching instances with a variety of operating systems, load them with at a custom application environment, manage network access permissions and more.

RedHat OpenShift Online. RedHat OpenShift Online is a cloud application deployment and hosting platform that offers almost all the functionalities provided from the on-premises version.

[84] https://hadoop.apache.org/
[85] https://spark.apache.org/
[86] https://kafka.apache.org/
[87] https://www.elastic.co/elasticsearch
[88] https://cloud.google.com/kubernetes-engine
[89] https://aws.amazon.com/eks/
[90] https://aws.amazon.com/ec2/
[91] https://www.openshift.com/products/online/

3.3.4 Cloud Computing: Function-as-a-Service

Many organizations that require extreme flexibility and scalability for their microservices architecture adopt this architecture to deploy their microservices in the cloud. Serverless computing services (also known as function-as-a-service solutions) are emerging for creating microservices apps.

3.3.5 Platform-as-a-Service

Platform as a Service (PaaS) provides cloud components to certain software while being used mainly for applications. PaaS delivers a framework for developers that they can build upon and use to create customized applications. All servers, storage, and networking can be managed by the enterprise or a third-party provider while the developers can maintain management of the applications.

3.3.6 Monitoring

Along with CI/CD, containerization and orchestration, monitoring and analysing the performance of the system as a whole as well as separate elements of the ecosystem is more required. A number of tools is currently being used by DevOps and System Admins. The most commonly used Prometheus, provides Grafana[92] support for querying Prometheus[93] collected data, and graph visualizations through dashboards. Prometheus is a free software application used for event monitoring and alerting. It records real-time metrics in a time series database (allowing for high dimensionality) built using a HTTP pull model, with flexible queries and real-time alerting. The project is licensed under the Apache 2 License, with source code available on GitHub, and is a graduated project of the Cloud Native Computing Foundation[94], along with Kubernetes and Envoy[95].

3.3.7 APIs

Communication between Microservices or between an application and a microservice can be achieved by exposing Microservice's API, which is usually a portion of a microservice, allowing for interaction with the microservice itself. Most common scenarios involve the use of REST API, usually for CRUD operations that follow certain well-established data-interchange formats such as XML, JSON, etc.

Swagger. Various frameworks have emerged the last years for designing, documenting and implementing APIs, with Swagger[96] being the most commonly used. Being an open-source software framework backed by a large ecosystem of tools Swagger by its turn relies on the OpenAPI Specification (OAS)[97], which defines a standard, language-agnostic interface to RESTful APIs. Designing and implementing microservices became accessible to individuals with no software engineering background. Swagger's CodeGen[98] supports the "Design First" approach where individuals can create microservices by writing an API specification in YAML or JSON format following the syntax guidelines provided by OpenAPI and Swagger Specification[99].

92 https://prometheus.io/docs/visualization/grafana/
93 https://prometheus.io/
94 https://www.cncf.io/
95 https://www.envoyproxy.io/
96 https://swagger.io/
97 https://www.openapis.org/
98 https://swagger.io/tools/swagger-codegen/
99 https://swagger.io/specification/

3.3.8 CI/CD

As noted previously, the modern Agile approach is tightly connected with such practices as DevOps, continuous integration (CI), and continuous deployment (CD). CI automates the integration of new code into a shared repository and CD tools automate the delivery of the software package to the deployment environment. Many application life cycle management and automation tools provide these capabilities. CI/CD tools bring even more value to the essence of microservices, which is the flexibility of software stack being used and the independence of each microservice. Along the rapid changes and adoption of Agile methodology, the evolution of new pipeline syntax techniques that encourage the declarative programming model, lead to a simpler and more opinionated syntax for authoring pipelines, the so called "Declarative Pipelines". This syntax combined with configuration files written in "human-readable data-serialization" languages such as YAML or the well-established JSON format, made these tools accessible and easier to understand from non-specialized individuals. In more demanding scenarios, specialized DevOps with "Scripted Pipeline" DSL knowledge, which also implements the "Pipeline as code" principle, can fully leverage the flexibility and extensibility of these tools and combine these DevOps procedures with Version Control System (VCS) tools, repository management systems and container and orchestration mechanisms to form a full ecosystem. Many application life cycle management and automation tools provide these capabilities. Popular CI/CD tools include Jenkins[100], Chef[101], Azure DevOps Team Foundation Server[102], TeamCity[103], Bamboo[104], CircleCI[105] and more.

Jenkins. Probably the most popular CI/CD tool due to being open source, providing hundreds of plugins and supporting a variety of software stacks like Java with Maven, React with Node.js, Python and more.

Chef. Chef is a configuration management tool that uses a pure-Ruby, domain-specific language (DSL) for writing system configuration "recipes". Chef is used to streamline the task of configuring and maintaining a company's servers and can integrate with cloud-based platforms such as Amazon EC2, Google Cloud Platform, Oracle Cloud, Microsoft Azure and more to automatically provision and configure new machines.

Azure DevOps Server. Azure DevOps Server, previously Team Foundation Server, is a fully cloud solution offered from Microsoft for both CI/CD and other cloud operations. CI/CD offers building, testing and deployment that works for a variety of supported languages.

TeamCity. TeamCity is a build management and continuous integration server from JetBrains. Open Source projects can use a free license of TeamCity and it supports a great number of VCS such as Git, Mercurial and more and a great number of languages such as Java, .Net and Ruby.

Bamboo. Bamboo is a CI/CD server developed by Atlassian. It is available only as an on-premises version, which is a drawback, but it has great integration with all Atlassian tools such as Jira[106] from project and issue tracking, Bitbucket[107] VCS, Confluence[108] for documentation, Trello[109] and more.

[100] https://jenkins.io/
[101] https://www.chef.io/
[102] https://azure.microsoft.com/en-in/services/devops/server/
[103] https://www.jetbrains.com/teamcity/
[104] https://www.atlassian.com/software/bamboo
[105] https://circleci.com/
[106] https://www.atlassian.com/software/jira
[107] https://www.atlassian.com/software/bitbucket
[108] https://www.atlassian.com/software/confluence
[109] https://www.atlassian.com/software/trello

Circle CI. Circle CI is another CI/CD tool. On the main advantages is that offers multiple building environments such as Docker, Linux, Windows, and more. Circle CI focus greatly on cloud-native continuous integration and a great number of big organizations are currently using it.

3.3.9 Business Process Automation

In recent years many tools for business process automation have been introduced, especially from big organizations, with complex systems. Drools[110] and Camunda[111] are the most well-known with RedHat's Drools system to be the most adopted.

Drools. Drools is a business rule management system (BRMS) with a forward and backward chaining inference-based rules engine. KIE (Knowledge Is Everything) is the new umbrella name to Drools, jBPM and other related technologies provided by RedHat. Drools supports the Java Rules Engine API standard for its business rule engine and enterprise framework. It is written in Java and along with jBPM and BPMN2[112] specification (Business Process Model and Notation) provides access to tremendous functionalities to not specialized individuals.

Camunda. Camunda is another open-source workflow and decision automation platform under Apache License 2.0 that is also widely being used. It offers similar functionalities with KIE server for design and automation of decision processing and it also adopts standards as BPMN.

3.4 Cloud Environments, Deployment and Monitoring of Services in Hybrid Contexts

Generally speaking, the architecture of a cloud computing environment can be divided into 4 layers: Hardware / data-centre layer, infrastructure layer, platform layer and the application layer [242]. The four layers are shown in Figure 8.

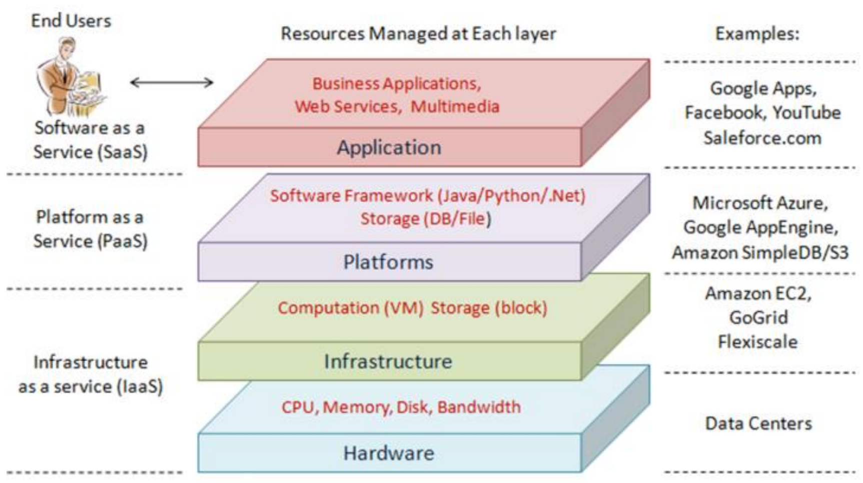
End Users
Resources Managed at Each layer
Examples:
Business Applications, Web Services, Multimedia
Application
Software as a Service (SaaS)
Google Apps, Facebook, YouTube Saleforce.com
Software Framework (Java/Python/.Net) Storage (DB/File)
Platforms
Platform as a Service (PaaS)
Microsoft Azure, Google AppEngine, Amazon SimpleDB/S3
Computation (VM) Storage (block)
Infrastructure
Infrastructure as a service (IaaS)
Amazon EC2, GoGrid Flexiscale
CPU, Memory, Disk, Bandwidth
Hardware
Data Centers

Figure 8: Typical Architecture for Cloud Computing Environments

The four layers provide a comprehensive view of cloud computing. Once this architecture is defined, we have to distinguish between 3 main different deployment models: *public*, *private* and *hybrid* cloud.

- *Public*: when the services are rendered over a network that is open for public use.

[110] https://www.drools.org/
[111] https://camunda.com/
[112] https://www.omg.org/spec/BPMN/2.0/

- *Private*: operated solely for a single organization, whether managed internally or by a third party, and hosted either internally or externally.
- *Hybrid*: is a composition of a public cloud and a private environment, such as a private cloud or on-premises resources, that remain distinct entities but are bound together, offering the benefits of multiple deployment models.

More specifically for public clouds use a kind of service model, in which resources are provided to the general public, as services in a 'pay-as-you-go' manner. Private clouds are intended for an organization or community to build and manage applications that are generally only accessible by a limited number of people. A hybrid cloud is a combination of public and private cloud that tries to bridge the gap between the two approaches. In the following paragraphs we present some considerations for deployment and monitoring of services in this hybrid contexts.

Deployment of Application in Hybrid Environments. Until recently, the assumption would be that, regardless of the chosen cloud architecture, it should be possible to provide the facility to use resources (as virtual machines) within the physical hosts. Containerization has introduced a new dimension, as a lightweight means of deploying and controlling virtual machine-like instances within a cloud environment. In computing, *virtualization* can have a broad meaning, but in terms of Virtual Machines (VMs) it refers more specifically to hardware virtualization, where the entirety of a computer's hardware is created in software, allowing any OS to run on it, regardless of the host's OS. However, due to VMs hosting an entire operating system, they tend to be quite large, and require much overhead, in terms of disk, memory and CPU usage, to facilitate what is often a single application. When multiple VMs are running on a physical host, a considerable amount of memory, disk space, and CPU cycles can be consumed merely by the guest operating systems themselves, before applications running within these VMs are even considered. There is a varying amount of overhead attributable to the type of virtualization in use. *Full virtualization*, for example, consists of simulating all hardware, and translating each CPU instruction in the guest VM to the host in real time. *Paravirtualization*, on the other hand, utilises modern CPU instruction sets to allow much of the guest OS to run on the same hardware, but under an isolated domain. A prerequisite for this is that the guest OS must be of the same hardware architecture as the host, and the guest OS needs to have been specially modified to be able to run in this environment. Hypervisors serve as the software that manages and runs virtual machines. Hypervisors fall into one of two categories, Type 1 and Type 2 [184].

Type 1 virtualization is also known as native, or bare metal hypervisor virtualization. In type 1 virtualization, the Hypervisor runs directly on host hardware. Technologies that are classified as type 1 include VMWare[113] ESX/ESXi, Microsoft HyperV[114] and Citrix XenServer/Xen[115]. In Type 2, the hypervisor runs inside the host OS. Oracle VirtualBox and VMWare Workstation are examples of Type 2 hypervisors. *Containers* are an OS level virtualization mechanism, rather than hardware level one. This manifest itself as an OS's kernel allowing multiple user space instances that are isolated from each other. Due to the OS level virtualization, the host kernel/OS system calls are utilised by the applications within the containers, meaning that little to no overhead is incurred. All of the OS-level requirements are already provided to containers by the host OS, meaning that containers need only include the application itself, and any required libraries and resources required to run. No initialization system or any of the processes an OS normally requires are needed, and so a container can be stripped down to be very lightweight,

[113] http://www.vmware.com
[114] http://www.microsoft.com/en-us/server-cloud/solutions/virtualization.aspx
[115] http://www.xenproject.org

while still bundling together everything needed to run the application. Docker provides a layered file system approach to containers, where at each step of the container's creation, a snapshot is taken of the file system, and only the difference between the current and previous snapshot is stored. By building up these snapshots, the final file system is provided. This allows for containers sharing a base or utilising the same libraries to share the base snapshots for their file system, reducing both storage and download costs. Docker also provides a hub, with many freely available pre-created containers to use, or build from. Unlike VMs, Containers are not 'OS agnostic'. The guests must use the same kernel as the host. This means that running a Windows guest on a Linux host is not possible, for example. Also, many container technologies do not support any kind of snapshotting similar to VMs. This may be due to the prevalence of containers in web-based applications, where state is often stored in a database, with the application itself being stateless, allowing it to be started and stopped at will without loss of data.

Dimensions of Monitoring in Cloud Computing. The cloud computing paradigm contains many shared resources such as infrastructures, data storage, platforms and software. Resource monitoring involves collecting information from those resources in order to facilitate the decision-making mechanism performed by other components in the cloud environment. Accordingly, monitoring systems at application level focuses on collecting data related to the application execution. The most important metrics to be analysed for the cloud applications are provided below [142]:

- *CPU Usage:* While an application is being started, the operating system creates a process and assigns a process identifier to it. CPU usage can then be measured for a specific application, or more generally.
- *Memory Usage:* Just like CPU usage, memory utilization can be measured for a specific application; the total memory usage can also be measured.
- *Storage Usage:* Disk usage shows the number of disk read/write operations for a certain application during a specified sampling period.
- *Response Time:* Response time can be measured for every application. Response time is calculated as the difference between the 'request received' timestamp (*t1*) and the 'response sent' timestamp (*t2*).
- *Network Usage:* It is important for a given service to adapt bandwidth based on demand. It needs to know how much network traffic each application is generating on a given machine [128].

These metrics depict a more precise view of the application's current performance. They allow service providers to define policies for scaling, elasticity or migration of their applications; defining for example in which situations the processing resource needs to be increased or under which conditions the application needs to migrate from one particular cloud infrastructure to another one.

Challenges of Monitoring in Cloud Environments. The main challenges of designing an efficient monitoring system for the Cloud environment are as follows:

- *Computing Overhead:* The main challenge in designing a monitoring framework in a cloud environment is ensuring that the overhead of the monitoring system is kept to a minimum [206].
- *Adaptability:* Since the monitoring system works in a dynamic and distributed cloud environment, where conditions and the number of resources changes dynamically, a monitoring framework must be able to detect the addition and removal of resources in the system. Moreover, in addition to the number of resources, the system should be extensible enough to accommodate new Quality of Service (QoS) metrics added to the monitoring system at both application and infrastructure levels. The basic challenge is that the system should be able to automatically detect and cope with the

changes in the number of resources and metrics without so much overhead that it would affect the performance of the whole system [2].

- *Ability to Extend:* Extensibility is defined as the ability to incorporate heterogeneity into the monitoring system with minimum intrusiveness and maintain appropriate performance. Extensibility is demonstrated by the ease with which extra functionalities can be added (in terms of different types of QoS metrics) into the monitoring system [104].
- *Ability to Manage:* The management overhead of a monitoring system should not scale linearly with the scaling of resources. Moreover, the monitoring system should allow for most of management functions to be automated [168].
- *Frequency of Measurement and Data Delivery:* The best interval between measurement and data transfer in the case of both static and dynamic data should be gradually evaluated to determine efficiency. Static data consists of information that does not change drastically over a particular period of time (such as the physical characteristics of resources). Dynamic data however is composed of information that is subject to change very frequently [168].
- *Data Aggregation and Storage:* Information regarding the various QoS metrics is gathered from the many resources involved, through the monitoring framework. This information has to be aggregated and stored efficiently, as it serves as input for resource provisioning algorithms. The storage mechanism must ensure that it utilizes storage space in an efficient manner and also must be able to fetch desired information with minimal overhead [168].
- *Non-Intrusiveness:* For gathering information regarding the various QoS metrics, a monitoring system should behave in a non-intrusive manner. This means that it should be able to obtain the necessary data without actually halting or affecting the actual performance of the whole system [168].
- *Communication Overhead:* The monitoring information gathered from the various resources should be exchanged with minimum overhead in the network. This would imply that the messages have to be transferred over the network in data formats that consume less bandwidth in the network. Minimal per-resource overhead and minimal overall system overhead should be guaranteed [168].
- *Fault Tolerance:* A monitoring system should be designed with fault tolerance, such that the occurrence of faults should not seriously affect the performance of the whole system. Failing resources should be easily detected and the system should be able to continue to operate and offer useful services even in the presence of such failures [168].

Analysis of Current Multi-Cloud Monitoring Tools. There are many tools that offer continuous monitoring and visibility for cloud applications. The problem with all these tools generally is that nobody offers a unique platform with the possibility of managing and monitoring a multi-cloud application and automatically adapting the monitoring activity at run-time. The tools offered by providers are powerful and complete but are not usable in a multi-cloud context; they also either lack a monitoring component or do not provide support for adaptation. This is a challenge for SmartCLIDE, because one of the most important objectives of the project is to improve the QoS/QoE control efficiency for time critical applications, by designing and implementing an autonomous self-adaptation platform which can deploy, monitor and dynamically adapt applications and federated cloud environments. Finally, different types of multi-cloud monitoring tools and outlines advantages and disadvantages of each are presented below:

- *Private Clouds Monitoring Systems (PCMONS)* is a monitoring tool created by Chaves, et al. [38], which was designed to address the lack of effective open source tools for private cloud monitoring. PCMONS demonstrates that cloud computing is a viable way of optimizing existing computing resources in data centres and orchestrating monitoring solutions on installed infrastructures is viable

[38]. However, it has several disadvantages: as PCMONS is a Nagios module, so it inherits Nagios performance and scalability issues that preclude applicability to huge cloud infrastructures; it is also compatible only with one solution. The system monitoring approach is focused on network security monitoring and response actions inside a cloud.

- *SBLOMARS:* In cloud platforms, recent efforts have been put into improving VMs monitoring and controlling. A number of frameworks have been proposed for VM management, which employ the Simple Network Management Protocol (SNMP). SBLOMARS [138] implements several sub-agents called *ResourceSubAgents* for remote monitoring. Each sub-agent is responsible for monitoring a particular resource. Inside each of these sub-agents, the Simple Network Management Protocol (SNMP) is implemented for management data retrieval. SBLOMARS is a pure decentralized monitoring system in charge of permanently capturing computational resource performance based on autonomous distributed agents. As it integrates SNMP technology, it offers an alternative solution to handle heterogeneous resources. Its distributed agents do not consume significant computational resources in their hosting nodes while they collect resource availability information from different operating systems for a twenty-four-hour period. However, due to the supporting SNMP protocol, the management information base (MIB) is implemented only for certain operating systems. Its second drawback occurs when the same MIBs show similar requested values in different formats. Even though the two different operating platforms use the same MIB, the content needs to be translated.
- *CloudCop:* In [161], CloudCop is a conceptual network-monitoring framework implemented using SNMP. CloudCop focuses on network QoS monitoring and also adopts a Service Oriented Enterprise (SOE) model. The CloudCop framework consists of three components: Backend Network Monitoring Application, Agent with Web Service Clients, and Web Service Oriented Enterprise. Currently the CloudCop Network Monitoring Framework is implemented completely with SNMP, which leads to a number of drawbacks, including its rudimentary information modelling capabilities and lack of support for configuration management.
- *Hyperic CloudStatus*[116] uses specific tools to monitor applications deployed in a multi-cloud context. It provides a view of the health and performance of the most popular cloud services on the Web, with the goal of identifying the cause when the performance of a cloud-hosted application changes. It aggregates multiple metrics from sources inside and outside the cloud and then it calculates the aggregated data to determine overall availability and normalized metrics across the cloud. For each service, CloudStatus results reflect general service levels, and serve as an indicator of whether further investigation of application behaviour or cloud performance is required. It is not a low-level monitoring component; it presents a powerful dashboard to analyse aggregated data and high-level information. However, this offering is not yet comparable to or integrated into existing in-house monitoring tools. Many providers use proprietary standards for their virtual machine containers and their APIs [112]. This situation leads to low technology sharing, among the IaaS providers and hence, current users can become locked-in with one provider.
- *JCatascopia* [227] is another tool with similar functionality. It is not limited to operating on specific cloud providers and can be utilized to monitor federated cloud environments where applications are deployed on VMs residing on multiple clouds. It can retrieve heterogeneous information both at machine level (e.g., CPU and disk usage) and at application level (e.g., throughput, latency, and availability). It also offers a rule mechanism allowing the developers to aggregate and activate new

[116] http://www.hyperic.com/products/cloud-status-monitoring

metrics. Another aspect is the adaptive filtering, performed with the aim of reducing the network and storage overhead by not transmitting values of a metric with very small variance with respect to the previously values. Another interesting feature is the possibility to adapt, in a simple way, the monitoring activity after machine migration. Each message contains the IP address of the monitored resource, so at each change the server is notified. JCatascopia cannot be utilized directly to calculate cost evaluations and estimations since it is not aware of the application topology. Secondly, since it is by nature cloud provider independent, it does not have access to specific pricing schemes. Finally, it does not have any knowledge regarding to which application (or composite) components new VMs belong to, when elasticity actions are enforced.

- *Nagios:* Nagios[117] is an open source computer system monitoring, network monitoring and infrastructure monitoring software application. Nagios offers monitoring and notification services for servers, switches, applications, and services. It alerts users when things go wrong and also alerts them when a problem has been resolved. Nagios is not a perfect fit for cloud monitoring. There is an extensive amount of manual configuration required, including the need to modify configuration when monitored VMs are instantiated and terminated. Performance is an additional issue: many Nagios service checks are resource intensive and a large number of service checks can result in significant CPU and IO overhead. Internally, Nagios relies upon a series of pipes, buffers and queues that can become bottlenecks when monitoring large-scale systems. Nagios was not designed or intended for monitoring large-scale cloud systems, and therefore requires extensive modification to be suitable for the task[118].
- *RESERVOIR*: The RESERVOIR project aims to federate clouds by offering "an open, service-based online economy in which resources and services are transparently provisioned and managed across clouds on an on-demand basis at competitive costs with high-quality service". Its main functionality is to provide wide monitoring information about services deployed in federated clouds for service management purposes, such as service billing, service elasticity, access control, SLA management, etc. However, this system does not address the issue of directly providing monitoring information to cloud customers [43].

[117] http://nagios.sourceforge.net/docs/3_0/configmain.html
[118] http://blogs.gartner.com/jonah-kowall/2013/02/22/got-nagios-get-rid-of-it

4 Software Quality Assurance in Microservice Applications

Relevant Perspectives. Quality assurance addresses itself to both local and cross-cutting concerns in different proportions during the various phases of the software life cycle. The following subsections provide first an overview of the software quality assurance process, and then we present views of quality assurance. It should come as no surprise that there is some overlap among these perspectives, and that will be reflected in our presentation. Each of the first two perspectives is presented by two subsections: the first one for assured characteristics generally, and then a section on security specifically; for life cycle phases, there is a section on development-time quality assurance, and a section on run-time quality assurance. The third perspective of generic vs microservices-specific qualities will be treated within each of the sections as appropriate. Each of the perspectives will be further elaborated during the course of the SmartCLIDE project in future deliverables.

State-of-the-Art Methodology. To identify the most relevant primary studies for the corresponding State-of-the-Art analysis, the following steps have been followed:

- we queried Google Scholar (i.e., a well-known research index that collects publications from various digital libraries), using the terms: (“service” OR “cloud computing”) AND (“mapping study” OR “literature review” OR “survey”)
- we manually browsed the first 200 results
- we retained studies that were relevant to (at least had a research question) software quality

Following the aforementioned process, we ended up with 9 relevant secondary studies (accumulating knowledge from more than 200 primary studies). These studies can be divided in three main categories, as follows: (a) 5 studies that explore all quality attributes related to service-oriented architectures (SOA); (b) 1 study that explores only the design-time quality attributes related to SOA; and (c) 3 studies that focus on specific design-time quality attributes.

Organization of this Section. In the following subsections we present both generic software quality assurance findings, and ones that are specific to Microservice Architecture (MSA). In addition to literature search, we present State-of-the-Art contributions by the consortium partners and other projects in which they have participated. To summarize, in Section 4.1 we present an overview of software quality assurance and identify quality characteristics to be assured. In Section 4.2 we discuss general quality assurance of key characteristics. Acknowledging the importance of software security characteristic while developing microservice-based applications, we devote Section 4.3 to security assurance. In Section 4.4, we present our findings related to design-time quality attributes, i.e., qualities that are discriminable at design-time [18], whereas in Section 4.5, we present our findings related to run-time quality attributes.

4.1 Overview of Software Quality Assurance

In the literature one can identify various ways to define the term “software quality”. According to Kitchenham et al. [118], software quality is a complex and multifaceted notion, which can be recognized, but not easily defined. For example, from the viewpoint of the end-user, quality is related to the appropriateness of the software for a particular purpose. From the software engineer’s point of view, quality deals with the compliance of software to its specifications. From the product viewpoint, quality is related to the inherent characteristics of the product, while from a cost viewpoint, quality depends on the amount that a customer is willing to pay to obtain it (i.e. higher quality products are generally expected to be more expensive).

To ease the management of software quality, stakeholders (e.g., software engineers, end-users, customers, etc.) usually negotiate and specify certain quality attributes (QAs) of interest for their projects. Quality attributes are organized into quality models, which in the majority of the cases are organized in a hierarchical manner [16][28][95][96][144]: high-level (HL) quality attributes are decomposed into Lower-Level (LL) ones (some quality models include more than one levels of LLs), which are subsequently mapped to quality properties that are directly quantified by software metrics.

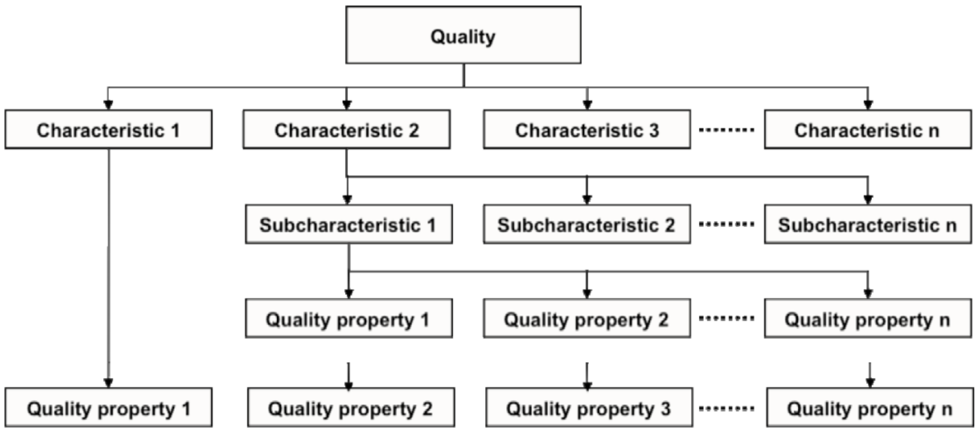

Figure 9: ISO 25010 Hierarchical Structure

For example, in the ISO/IEC 25010 model, product quality is defined as follows (see Figure 9):

- ***the first level*** (HL / characteristics) separates product quality into eight QAs (Functional Suitability, Performance / Efficiency, Compatibility, Usability, Reliability, Security, Maintainability, and Portability);
- ***the second level*** (LL / sub-characteristics) decomposes each quality attribute into sub-characteristics, e.g., Maintainability is decomposed into Modularity, Reusability, Analysability, Testability, and Modifiability);

The LL sub-characteristics can be evaluated by measuring internal quality properties (typically static measures of intermediate products), or by measuring external quality properties (typically by measuring the behaviour of the code when executed), or by measuring quality in use properties (when the product is in real or simulated use) (Figure 9) [95]. The goal of this sub-section is to identify and highlight the most important quality attributes for microservice applications, which are the focus of this project. Some attributes may be inherited from general quality considerations and some are specific to microservice applications.

Key defining characteristics of Microservices Architecture (MSA). As we are discussing microservice-based systems in particular, we refer to The Open Group's white paper, Microservices Architecture[119], which identifies five key defining characteristics of a microservices architecture, as shown in Table 1. All of these key defining characteristics are expected to be intrinsic characteristics and therefore must be addressed at development time. However, at least one, "*Highly Resilient*," also has a distinct run-time behavioural manifestation.

Table 1: Key defining characteristics of an MSA

[119] The Open Group. Microservices Architecture, 2016. https://www.opengroup.org/soa/source-book/msawp/p3.htm

Key	Description
Service Independence	Minimizes the impact to the service infrastructure by identifying and isolating those services that undergo constant churn. Once identified, these services should be upgradeable or replaceable without any additional changes to the software landscape.
Single Responsibility	Is the direct alignment of a service to a singular business activity. A business activity can be described as a unit of work performed by the organization that supports an existing business process or function.
Self-Containment	Dictates that a service shall encompass all external IT resources necessary to support the business activity. It also necessitates that service dependencies falling outside the scope of the development team should be minimized or preferably eliminated.
Highly Decoupled	To maintain minimal service dependencies, microservices must be highly decoupled. To achieve this, the business function must be capable of being decomposed down to the level where a microservice is implementing a single atomic business function.
Highly Resilient	Microservices within an MSA must be designed for potential failures because individual service failures should not impinge negatively on the user experience. Since a microservice represents a single responsibility and is self-contained, a service failure could mean that a given business function or process is unable to complete successfully. *Assurance that a system is "Highly Resilient" can be achieved by design-time analysis and construction supplemented by real-time service monitoring to provide a proactive means of identifying services that are struggling to satisfy Service-Level Agreements (SLAs)* .

Assurance of these key defining characteristics must begin with requirements and design. The last of these, "Highly Resilient" is subject to a run-time assurance component. We also note that "Highly Resilient," identified in the MSA White Paper, intersects with a key characteristic identified by the Industrial Internet Consortium for IIoT systems, as we shall discuss in the following.

Key characteristics of IIoT systems. For our quality assurance effort, we believe the identification of general high-level quality attributes should follow a contemporary industry standard. The Industrial Internet Consortium (IIC) Security Framework (IISF)[120] identifies "key system characteristics" of Industrial Internet of Things (IIoT) systems. Such systems that span from the "cloud" to individual devices or cyber-physical products are becoming more typical and it is more useful to expect all systems to have these varied aspects and then possibly have some aspects be absent. Table 2 identifies five key characteristics as general high-level attributes requiring assurance of trustworthiness. The IIC Vocabulary[121] defines trustworthiness as the, "degree of confidence one has that the system performs as expected with characteristics including safety, security, privacy, reliability and resilience in the face of environmental disruptions, human errors, system faults and attacks." We believe this list of attributes is a

[120] Industrial Internet Consortium (IIC), Industrial Internet of Things Volume G4: Security Framework, IIC:PUB:G4:V1.0:PB:20160926, 2016.

[121] Industrial Internet Consortium, Industrial Internet of Things Volume G8: Vocabulary, IIC:PUB:G8:V2.00:WD:20170522, 2017.

good one to start with for our high-level quality attributes, both because it is fairly comprehensive and because it is part of a relevant community standard. One or more of these characteristics apply to different kinds of systems in differing proportions, as noted below, but IIoT systems tend require them all in comparable importance.

Table 2: Key Characteristics of IIoT Systems

Key	Description
Security	The condition of a system being protected from unintended or unauthorized access, change or destruction. *Assurance of security* is often assessed in terms of risk.
Safety	The condition of a system operating without causing unacceptable risk of physical injury or damage to the health of people, either directly or indirectly, as a result of damage to property or to the environment. *Assurance of safety* endeavours to eliminate both systematic and probabilistic failures.
Reliability	The ability of a system or component to perform its required functions under stated conditions for a specified period of time. *Assurance of reliability* requires detailed understanding of the operational environment, the system's composition and how it was engineered and pre-fielded to establish the likelihood of failure.
Resilience	The emergent property of a system that behaves in a manner to avoid, absorb and manage dynamic adversarial conditions while completing the assigned missions, and reconstitute the operational capabilities after causalities. *Assurance of resilience* adds physical or logical redundancy for elements and interconnections and provides for transfers to the alternate elements and connections when needed.
Privacy	The right of an individual or group to control or influence what information related to them may be collected, processed, and stored and by whom, and to whom that information may be disclosed. *Assurance of privacy* depends on whether stakeholders expect, or are legally required, to have information protected or controlled from certain uses. It is important to stay up to date with regulations and standards, such as the new framework for transatlantic data flows called the EU-US Privacy Shield and the EU General Data Protection Regulation (GDPR).

An IIoT system is an amalgamation of information technology (IT) and operational technology (OT) systems. In the past IT and OT systems have evolved separately and have been subject to diverse emphasis upon specific characteristics. Figure 10 illustrates how traditional IT has placed primary emphasis on privacy, security and reliability, less emphasis on resilience, and none on safety. Traditional OT has placed primary emphasis on safety, resilience and reliability, less on security, and none on privacy. However, in the convergence of IT and OT in IIoT systems all of these characteristics have become important individually and in combination for overall system trustworthiness.

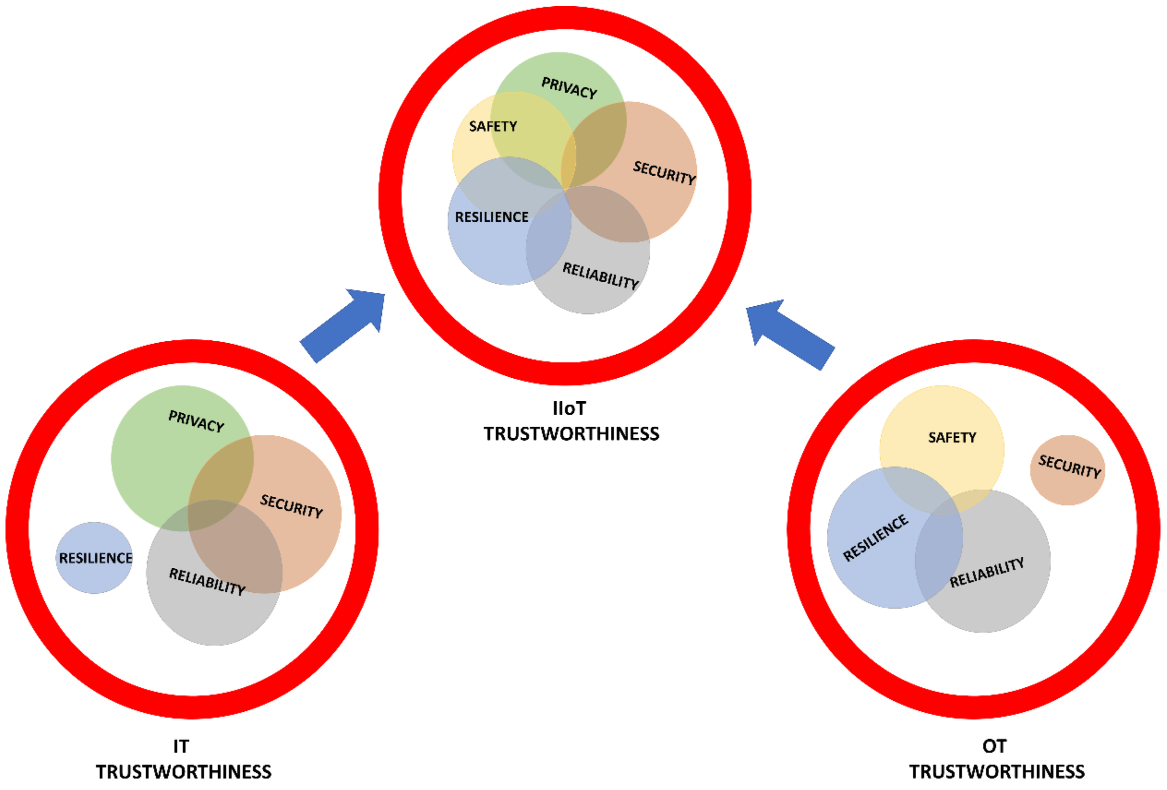

Figure 10: Convergence of IT and OT key characteristics

4.2 Quality Assurance of Key Characteristics

Quality assurance (for trustworthiness) of key characteristics has traditionally been pursued through distinct techniques and tools for each characteristic, applied within the development communities corresponding to the kind of system (e.g. IT, industrial automation, avionics, nuclear, medical, defence, automotive). The cultures of these isolated communities differed, and their methods were specialized, esoteric, and often proprietary. There was little commonality of efforts, method, or terminology, particularly in domains requiring augmented assurance, which have been relatively small communities.

Since the turn of the 21st century, we and other researchers and practitioners have begun to notice that common techniques for property specification and assurance are applicable to the diverse properties that are of foremost concern to various communities. A broad class of *emergent properties*, that is properties that are required to be exhibited at the system level but are typically not present in the individual components of the system, can be expressed as sets of behaviours of the system in temporal logics (or, for hyper-properties, sets of sets of behaviours). Advancements, especially over the past 20 years, in automated formal verification of non-trivial systems, based on model-checking of temporal logic specifications have revealed a common approach to the verification of all such properties that can be specified behaviourally. By applying common techniques and tools it has become possible to dispense with quirky domain-specific techniques and tools that could never achieve critical mass to be more widely adopted. This has enabled a larger population of practitioners to fuel further advancements. However, the techniques and tools still demanded exceptional skills to apply them manually.

Formal methods, previously manually applied and requiring specialized skills, began to be incorporated into other engineering tools so that their benefits could be had with little or no additional effort.[122] Certain kinds of formal methods can be made to "'disappear' into the familiar fabric of software engineering practice." [123] This trend is amplified by the growing interest in and adoption of model-based design approaches and tools.

In two previous projects, D-MILS[124] and CITADEL[125], we have pursued techniques and tools for the assurance of general behavioural properties of MILS systems (a particular approach to the development of assured component-based systems that is somewhat akin to MSA). Among the advancements to the State-of-the-Art made by those projects were languages for the specification of the structure and properties of distributed and dynamically reconfigurable systems, and analysis tools for compositional verification of temporal logic properties of such systems using a contract- and model-based approach.

We intend to explore the quality assurance of both the general and MSA-specific properties enumerated above in the context of the SmartCLIDE project. In keeping with the main theme of the project, which is to provide the ability to develop applications by non-developers through intelligent support, we similarly will seek to use intelligent support so that the quality assurance of the applications may be achieved by automated methods.

We have just discussed how similar techniques and tools may be applied to a variety of key properties. Previously, we have identified and sought to address the security characteristic in particular. For the present we discuss in further detail only the key characteristic of security in the Section 4.3. During the course of the project and in future deliverables we will report on opportunities to apply the common quality assurance approach to other key characteristics.

4.3 Security Quality Assurance

We now consider, specifically, the key characteristic or quality attribute, *security*. We first acknowledge the "C-I-A" triad of security aspects in Table 3 (at times referred to variously as pillars of security, or basic security services). This interpretation of the three aspects is one of the oldest and widely used decompositions of computer security.

Table 3: Aspects **of** Security

Aspect of Security (traditional triad)	Description
Confidentiality	This aspect ensures that there is no unauthorized disclosure of data.
Integrity	This aspect ensures that there is no unauthorized modification of data. Also refers to protection of data or processing of data throughout its life cycle, to its origin, or to the trustworthiness of mechanisms that process or transmit data.
Availability	This aspect ensures data, services or resources are usable when desired.

[122] John Rushby. Integrated Formal Verification: Using Model Checking With Automated Abstraction, Invariant Generation, and Theorem Proving, 1999.
[123] John Rushby. Disappearing Formal Methods, 2000.
[124] Distributed MILS Project, European Commission FP7, 2012-2015, http://www.d-mils.org.
[125] CITADEL Project, European Commission H2020, 2016-2019, http://www.citadel-project.org.

Because they are broad, the interpretation of these aspects of security must be made with respect to a particular situation and multiple times throughout a system. Other variations in the definition of this triad have appeared that at times tend to create confusion. (E.g. it should not be confused with another distinct triple of risk categories, presented in the following, that also begin with the initial letters A, C and I). These variations tend to arise from interpretations within a particular situation or application domain. It is better to maintain the standard definition and to present these variations as situation-dependent interpretations of the standard, or if necessary, to extend the standard.

To cover all cases that may be encountered, the aspect of Integrity may be given a broader interpretation than the first description that is presented for integrity in Table 3. The usefulness of the first description item is that it makes the complementarity of Confidentiality and Integrity apparent. However, the broader considerations surrounding the processing and protection of data in a trustworthy fashion should also be considered part of Integrity, as reflected in the second description item.

Depending upon the data objects (or classes of objects) to be protected, the operations that may legitimately be performed on the objects by different subjects (or classes of subjects), and the authorizations that must be granted to subjects to do their function, specific policies may be defined. These policies can be used to actively enforce restrictions on operations or to passively detect when operations are performed that contradict the policy. Detection may trigger remedial actions.

With respect to Availability, is should be noted that this aspect of security is often classified as an aspect of *dependability*, particularly when Security is also classified as a distinct aspect of dependability, as in Olovsson [172]. This is a fair alternative view, particularly when the context includes other quality attributes besides security. Furthermore, availability is different in character to confidentiality and integrity, which have specifically to do with data protection. Olovsson identifies Dependability as the superclass of Availability, Reliability, Safety and Security. While failure of availability can certainly have security consequences, it may also have impact upon the other high-level quality attributes identified previously (e.g. Safety, Reliability, Resilience).

Security Concerns in Microservices Architectures. Microservices architectural style assumes that autonomous, independently deployable services collaborate to form a broader software application or a system. Although there is no a precise definition of this architectural style, microservices are generally considered as a variant of service-oriented architecture [71]. The main benefits of microservices are outlined as a high degree of decoupling into small and independently scalable services, isolation of issues, and easy adoption of new technologies. Fortunately, most aspects of security in microservice architecture are similar to monolithic application. However, microservice architectural pattern introduces specific security challenges and problems [189], which should be treated differently. Based on the existing literature review and best practices adopted by many leading IT companies (e.g. Amazon, Netflix, Spotify, Twitter) we have identified several areas of security concerns and risk categories that have arisen along with the microservice paradigm. Due to the large investment by aforementioned companies, the industrial state-of-practice on microservices is rich. On the contrary, academic research efforts are not mature, as discussed in existing secondary studies [176] [214] [232]. Authors in [232] point out that security issues were not listed in the top microservice challenges (i.e. they were addressed in only 3 of the 33 papers examined). This indicates a gap in the microservice security research. Authors in [239] present an overview of security challenges in microservice architectures and propose a hierarchical decomposition of microservice security issues into 6 layers: hardware, virtualization, cloud, communication, service/application and orchestration. Besides, they discuss industry developments of Docker Swarm and Netflix public key infrastructure solution.

Security Risks: A microservice security architecture could be defined as a set of measures to minimize risks [77]. In order to select right design, it is important to understand potential risks. Risks for microservices could be classified into three categories [214], as shown below. The three main risk categories and corresponding subcategories are outlined in Table 4.

- **Availability** – is the risk that the microservice will not be available;
- **Connectivity** – category includes a large variety of risks that occur as a consequence of using internet and other public networks;
- **Integrity** – category identifies weaknesses that can make a service or the server it runs on vulnerable to attack.

Table 4: Microservice Risk Categories

	Category	Definition
Availability	***Availability of Infrastructure***	This risk occurs if the infrastructure that microservices run on becomes unavailable. The most common mitigation action is to have a redundant physical and virtual infrastructure. It is important to consider various deployment options (i.e. different racks, data centres, etc.). This risk is addressed by infrastructure architecture, taking care to ensure adequate performance level.
	Distributed Denial of Service	This risk occurs when an attacker sends a flood of bogus requests to a service. A challenging issue is to distinguish between legitimate and bogus sources. A possible mitigation strategy is to assign a unique API key to each client.
Connectivity	***Secrecy of Data in Flight***	This risk occurs when sensitive data in service request or response are transferred over insecure network or public internet. The common mitigation practice in this case is encryption over HTTPS connection.
	Server Spoofing	This risk occurs when network traffic is rerouted to a false server (i.e. Address Resolution Protocol spoofing, Domain Name System spoofing, IP Address spoofing, and Dynamic Host Configuration Protocol spoofing). The common mitigation measure to require a service to prove its identity to the caller (e.g. using X.509 certificate over HTTPS).
	Inauthentic Messages	Two common types of attacks based on inauthentic messages are: (1) the sender/origin of the message or (2) message content has been altered in transit. The standard mitigation measure is to use HTTPS with the mutual authentication option using a decryption key.
	Man-in-the-Middle Attack	Man-in-the-Middle Attack occurs when a third party inserts itself between a client and a server. Man-in-the-Middle Attack is similar to Server Spoofing, but the consequences could be worse.
	Replay Attack	This risk occurs when attacker records a request (including signatures and security tokens) and then sends recorded request to the service. Common mitigation measure is HTTPS connection, but every request should contain a unique random number and a timestamp.

	Category	Definition
Integrity	***Secrecy of Data at Rest***	This risk could appear if the service is responsible for providing access to sensitive data. Modern distributed cloud applications often store and process large amounts of sensitive data, and it is of the utmost importance to ensure that only authorized users can access it. One of the mitigations measures is to ensure the security of log messages and data stores.
	Bad Message Format	This risk is introduced when a service is tolerating incorrectly formatted requests, which are commonly employed by hackers for probing security flaws. A common mitigation issue is early identification and rejection of all wrong/malicious requests.
	Malware and Other Corruptions of the Service's Host	Services run in environments (e.g. VMs, containers) which should be secured by both internal and external firewalls. This is a standard risk, which is not typical only for microservices. Typical mitigation measure is based on run-time monitoring with the goal to prevent attacks to or from microservice's containers or VMs.

Common Countermeasures: To ensure secure microservices, several security issues should be addressed on both hardware and application levels. Microservice applications should implement the following security services [189] [214]:

- Authentication is the process of verifying the identity of the application or human that is attempting to access the microservice. There are several different ways that authentication could be achieved at the microservice level. The most common approach is to include an authorization header in each HTTP request. For example, this might contain a username and password, which is a technique known as a basic authentication. Another approach would be to include an API key into authorization header or to create user specific certificates. However, certificates could be complex to install and manage. A better option could be to implement API gateway to authenticate a request before forwarding it to the microservices. Centralizing API authentication in API gateway is widely adopted in practice. One of the popular patterns is *Access token*, where API gateway passes a token containing information about the user (e.g. identity and roles)[126]. An alternative approach would be to adopt an industry-standards protocols such as OAuth 2.0[127] and OpenID Connect[128]. OAuth 2.0 is a complex topic and should be carefully investigated. Aaron Parecki provides a good standard overview and examples on how a microservice architecture might use OAuth 2.0 in the online book OAuth 2.0 Servers[129]. Besides, Chapter 7 of Spring Microservices in action discusses practical advantages and disadvantages of OAuth 2.0 standard approach[130].
- Authorization is the process of verifying that the user is allowed to perform the requested operation on the specified data. The popular ways to address authorization are access control lists (ACLs), role-based security, etc. Many web API frameworks (e.g., ASP.NET Core) provide built-in

126 https://microservices.io/patterns/security/access-token.html
127 https://www.oauth.com/playground/
128 https://openid.net/connect/
129 https://www.oauth.com/
130 https://livebook.manning.com/book/spring-microservices-in-action/chapter-7/

mechanisms to perform authorization by checking various constraints like user roles. However, developers should be responsible to secure all single endpoints of public microservice APIs and clearly define what actions each potential user should/shouldn't be allowed to perform. Implementing authorization and authentication could be a challenging task. The best option would be to adopt a proven security framework such as *Spring Security*[131] (a sophisticated Java framework with advanced security support), *Apache Shiro*[132] (Java framework), *Passport*[133] (a popular security framework for NodeJS focused on authentication).

- Auditing is the process of tracking user operations to detect security issues. Richardson suggests several patterns that a service developer must implement to ensure microservice security and maintainability during the operation SDLC phase. The observability patterns that enable developers and operation managers to understand the behaviour and prevent security incidents are briefly described in Table 5. More is said about security auditing as a run-time quality assurance activity in Section 4.5.

Table 5: Overview of the Observability Patterns

Observability Patterns	
Health Check API	A service exposes a health check API endpoint, such as GET /health, which returns the health of the service[134].
Log aggregation	Aggregate the logs of all services in a centralized database that supports searching and alerting[135].
Distributed tracing	Aggregate the logs of all services in a centralized database that supports searching and alerting[136].
Application metrics	Services report metrics to a central server that provides aggregation, visualization, and alerting.
Exception tracking	Services report metrics to a central server that provides aggregation, visualization, and alerting. There are several exception tracking services (e.g. Honeybadger[137], Sentry.io[138]
Audit logging	Record user actions in a database in order to help customer support, ensure compliance, and detect suspicious behaviour[139]

- Secure inter-process communication should be based on Transport Layer Security (TLS). In order to simplify the development, microservices should be implemented on top of the microservice chases[140] (i.e. a framework that handles various crosscutting networking related functions such as service discovery, distributed tracing, application metrics, etc.). A key disadvantage of microservice chases pattern is that you need a separate framework for each programming

[131] https://spring.io/projects/spring-security
[132] https://shiro.apache.org/
[133] http://www.passportjs.org/
[134] http://microservices.io/patterns/observability/healthcheck-api.html
[135] http://microservices.io/patterns/observability/application-logging.html
[136] http://microservices.io/patterns/observability/application-logging.html
[137] http://www.honeybadger.io
[138] https://sentry.io/welcome/
[139] http://microservices.io/patterns/observability/audit-logging.html
[140] http://microservices.io/patterns/microservicechassis.html

language. Richardson suggests that is likely that many of the network related functions will migrate into a service mesh[141].

- Defence in Depth methods proposed in the literature are a combination of different technical measures on various levels of depth, and good architectural choices to enable benefits of microservice architecture without sacrificing security. The traditional idea of network-based security is not sufficient to provide adequate security level. The *defence-in-depth* principle states that developers should not rely entirely on a single technique to secure microservice application, because if that one defence is breached, then everything is lost, and the attacker might gain free access to everything. By combining several different layers of security, we can significantly reduce the possibility of a data breach. Authors in "*Defence-in-depth and Role Authentication for Microservice Systems*" assess mutual Transport Layer Security (TLS) as a solution to achieve advanced overall security. They concluded that establishing overall TLS is hard and requires source code modifications [98]. The techniques that should be considered whenever possible include encrypting data in transit with TLS, encrypting data at rest in files and databases, authenticating users, ideally by means of industry standard protocols such as OAuth 2.0 and OpenID Connect, and ensuring that we also correctly authorize access to our microservices so that users can only perform the actions that they have permission for. In addition, we can configure various network level security such as virtual networks, IP address whitelisting, firewalls, and making use of API gateways that can create a single point of entry for external traffic. It is also a great idea to make use of security experts to perform penetration testing and provide feedback on our *system design*. We should configure alerts for attack detection and regularly review the system audit logs to identify any suspicious access patterns or behaviour. In the next paragraph, we are going to discuss the concept of security by design for software products, with specific focus on microservices.

Development of Secure Microservices. Microservices, as software products in nature, need to be developed having security in mind from the early stages of their development [63]. Simply ensuring the implementation and deployment of mechanisms (either external or internal) that enhance the protection of the microservices with respect to important security aspects, including Availability, Confidentiality, and Integrity, is not enough for fully-protecting them against attacks. Important vulnerabilities should be identified and mitigated prior to the release of the microservices, in order to reduce the probability of critical security breaches [63] [154]. In fact, the exploitation of a single vulnerability may lead to far reaching consequences to the owing enterprise of the compromised software, including financial losses and reputation damages. For instance, Equifax Breach (CVE-2017-5638[142]) [134], allowed criminals to expose the personal data of more than 143 million Equifax customers, leading to a total cost of $1.35 billion according to the company's financial results of the first quarter of 2019. In addition, according to another recent report [143], the annual cost to the global economy from cybercrime is estimated at $400 billion. Hence, appropriate countermeasures are needed in order to avoid the introduction, or at least the exploitation, of software vulnerabilities, and, in turn, their associated devastating consequences.
Most of the software vulnerabilities stem from a small number of common programming errors [91]. For instance, SQL Injection and Cross-site scripting, which are listed both by OWASP[143] and NIST[144] as the most dangerous and common vulnerabilities of web services and applications, are caused by lack of input

[141] http://microservices.io/patterns/deployment/service-mesh.html
[142] https://nvd.nist.gov/vuln/detail/CVE-2017-5638
[143] https://www.owasp.org/index.php/Main_Page
[144] https://www.nist.gov/

validation/sanitization, which is a relatively simple to address programming error. Another source of security issues is the selection of insecure third-party reusable components and Application Programming Interfaces (APIs) [78]. Heartbleed [37] and Equifax Breach [134] constitute two representative examples of such issues, since they were caused by security-related bugs found in the OpenSSL and Apache Struts 2 libraries respectively. Finally, even code refactoring performed for the elimination of security issues or quality improvements is known to lead to the corruption of previously flawless code, and, in turn, to the introduction of new vulnerabilities [30][88]. For example, a minor coding flaw in a revised fragment of AT&T switching code (in fact, a misplaced break statement) led to the most disastrous service disruption that the company has ever experienced [88].

From the above analysis it is clear that these common programming errors with security implications are introduced by the developers during the coding phase mainly due to their lack of security expertise [147]. However, it is unrealistic to expect from them to remember thousands of security-related bug patterns to avoid. Another reason that leads to the introduction of security issues in software products is the accelerated production cycles [27]. Strict deadlines that should be met often force developers to focus mainly on the implementation of the predefined functional requirements, neglecting in that way the security of the code they produce [78] [237]. Appropriate tooling is required to help them avoid the introduction of such security issues during the SDLC [147], and therefore write more secure code [78] [237].

Automatic Static Analysis (ASA) tools have been proven effective in uncovering security-related bugs early enough in the software development process [39] [148]. Their main characteristic is that they are applied directly to the source or compiled code of the system, without requiring its execution [146]. In fact, automatic static analysis is considered an important technique for adding security during the software development process. This belief is expressed by several experts in the field of software security (e.g. [39] [147]), while well-established secure Software Development Life Cycles (SDLCs), including the well-known Microsoft Security Development life cycle [90] [91], OWASP's Secure SDLC[145], and Cigital's Touchpoints [146], propose the adoption of static analysis as the main mechanism for adding security during the coding (i.e., implementation phase) of the SDLC. In addition, ASA is a security activity commonly adopted by major technological firms including Google, Microsoft, Adobe and Intel, as reported by the BSIMM[146] initiative.

Moreover, static analysis is believed to be more effective in detecting code-level vulnerabilities compared to other dynamic testing approaches like penetration testing and fuzzing, as it is observed to produce significantly fewer false negatives [66]. This can be explained by the fact that software vulnerabilities often exist in states that are hard-to-reach, and that ASA tools can peer into these states more efficiently than dynamic analysis [39]. In addition to this, contrary to dynamic analysis, static analysis does not require an executable version of the source code to be available in order to be applied, allowing its execution from the early stages of the development process, and therefore the early identification of security issues, which is important for secure software development. Hence, based on the above analysis, ASA tools constitute an attractive solution for software development enterprises that wish to consider security from the coding phase of their applications.

However, despite the benefits of the ASA tools in the domain of software security, it has been observed that they are underused in practice [23] [101]. The main reason for their limited adoption is their poorly

[145] https://www.owasp.org/index.php/OWASP_Secure_Software_Development_Life cycle_Project
[146] https://www.bsimm.com/

presented results [147]. These comprise long lists of raw warnings (i.e., alerts) or absolute values of software metrics, which do not provide real insight to the stakeholders of the software products [210]. In addition, the large volume of the produced alerts often hide important issues that may correspond to exploitable vulnerabilities, whereas significant effort is required by the developers in order to triage these long lists and detect actual issues [210]. A great number of ASA tools have been proposed over the years, leading to the generation of a large volume of raw results, which potentially contain security-relevant information. Hence, appropriate knowledge extraction tools are needed on top of the raw results produced by the ASA tools in order to extract knowledge that may facilitate the development of more secure software products, including microservices. This knowledge may include the identification of potentially exploitable vulnerabilities or of software components (e.g., parts of microservices or the microservices themselves) that are likely to contain exploitable vulnerabilities. Developers and project managers can leverage this information for prioritizing their testing and fortification efforts, by focusing on parts and alerts that are interesting from a security viewpoint, increasing the probability of identifying and mitigating actual vulnerabilities, and leading, in that way, to more secure (i.e., vulnerability-free) software.

ASA Tools for detecting software vulnerabilities. As already mentioned, a multitude of static code analysers able to detect important security issues that reside in the source code of software products have been proposed over the years. Table 6 shows the most representative static code analysers that are widely used in practice for security auditing purposes. Subsequently, a short description of these analysers is also provided. It should be noted that all the tools presented in Table 6 are included by both OWASP[147] and NIST[148] in their lists of recommended tools that can be used to detect common vulnerabilities. A more detailed survey of security-specific static code analysers can be found in [66].

Table 6: Static Code Analysers for Security Auditing Purposes

Tool	Category	Vendor	Availability	Artefact
FindBugs	Bug finding tool	University of Maryland	Open-source	Code
PMD	Bug finding tool	PMD	Open-source	Code
Lapse+	Taint Analyzer	OWASP	Open-source	Code
WAP	Taint Analyzer	OWASP	Open-source	Code
Fortify SCA	Bug finding tool & Taint Analyzer	HP	Commercial	Code
Veracode Static Analysis	Bug finding tool & Taint Analyzer	Veracode	Commercial	Code

FindBugs [89] is an open-source static code analyser, widely used in practice for security auditing purposes. It applies data- and control-flow analysis in order to detect common bug patterns that indicate the existence of software bugs (including vulnerabilities) in Java applications. These bug patterns are grouped into bug categories based on their relevance, while they are also classified into four ranks denoting their severity, which are: (a) scariest, (b) scary, (c) troubling, and (d) concern. The

[147] https://www.owasp.org/index.php/Source_Code_Analysis_Tools
[148] https://samate.nist.gov/index.php/Source Code Security Analyzers.html

FindSecurityBugs[149] plugin provides 125 additional bug patterns that correspond to different vulnerability types. Findbugs provides both a Command Prompt tool and a standalone GUI application. It also integrates with well-known IDEs including Eclipse, IntelliJ IDEA, and NetBeans, as well as with software analysis platforms like SonarQube.

PMD[150] is a source code analyser that searches for violations of specific rules that correspond to best coding practices in software applications written in several programming languages, including Java and JavaScript. It applies pattern matching to a model extracted from the product source code (in fact, its Abstract Syntax Tree) in order to identify violations of specific patterns (i.e. rules) that correspond to best coding practices. Newer versions of the tool also support data-flow analysis in order to improve the overall precision. Although PMD is generally considered a code quality tool, it contains security-related rules (e.g. rules related to resource handling, exception handling, and synchronization), and has been used in the literature for vulnerability detection (e.g. [211]). Similarly to FindBugs, PMD integrates with well-known IDEs (e.g. Eclipse, IntelliJ IDEA, and NetBeans), as well as with software management and building tools like Ant[151] and Maven[152].

LAPSE+[153] is a taint analyser for Java EE Applications. As a taint analyser it searches for suspicious paths from tainted sources (i.e. inputs to which untrusted data are injected) to security-sensitive sinks (i.e. statements that perform a security-critical functionality) [66]. Hence, LAPSE+ is able to detect injection vulnerabilities, including SQL Injection (SQLI) and Cross-site Scripting (XSS), which are considered as the most common and dangerous threats of web applications according to both OWASP Top 10[154] and CWE Top 25[155] lists of most common software vulnerabilities. LAPSE+ is released as a plugin for the Eclipse IDE.

WAP [152] is an ASA tool that semantically analyses the source code of web applications written in PHP in order to detect input validation vulnerabilities, like XSS and SQLI. In particular, similarly to LAPSE+, it performs taint analysis to examine whether untrusted inputs inserted by the system's entry points reach security-sensitive sinks. Data mining is employed to discriminate actual bugs from false positives, while actual bugs are automatically fixed by the tool.

Fortify SCA[156] is a highly accurate and precise commercial static analysis tool for detecting code-level security issues. It is widely used in practice for security testing purposes (e.g. [58]). It is originated from ITS4 [231], which is believed to be the first ASA tool for security auditing [146]. Fortify SCA supports over 25 programming languages including C/C++, C#, Java, JavaScript, and Python, and is able to detect 770 unique vulnerability types, providing their severity and location. The tool also provides potential remediation strategies for facilitating the mitigation of the identified vulnerabilities. It consists of multiple specialized source code analysers, which utilize secure coding rules to analyse the code base for violations of secure coding practices. Apart from a standalone application, Fortify SCA integrates with major IDEs through a set of plugins.

[149] https://find-sec-bugs.github.io/
[150] https://pmd.github.io/
[151] https://ant.apache.org/
[152] https://maven.apache.org/
[153] https://www.owasp.org/index.php/OWASP_LAPSE_Project
[154] https://www.owasp.org/index.php/Top_10-2017_Top_10
[155] http://cwe.mitre.org/top25/
[156] https://software.microfocus.com/en-us/products/static-code-analysis-sast

Veracode Static Analysis[157] is another commercial security-specific ASA tool, which supports almost all the widely-used programming languages. Similarly to Fortify SCA, it is able to detect vulnerabilities that are caused both by injection of untrusted data, and by violation of secure coding practices. A SaaS-based environment is available in order to facilitate its usage, while it integrates seamlessly into the developers' workflow through the Veracode Greenlight[158]. Veracode Greenlight provides a set of plugins for widely-used IDEs, which analyse the source code while the developers are coding, providing in that way alerts and remediation strategies as early in the development process of software products as possible.

The above analysis suggests that a large number of ASA tools that are highly effective in detecting software vulnerabilities are available to be used in practice. Hence, as already mentioned, an interesting topic is to investigate potential ways for extracting security related information from the results produced by these tools, which may facilitate the development of secure software products. More specifically, this information can be leveraged to facilitate the identification and, in turn, mitigation of actual vulnerabilities that reside in the source code. This is expected to address the major shortcomings that these tools encompass, which prevent their wider adoption in practice.

Vulnerability Prediction. The results of ASA tools can be leveraged for the conduction of more efficient vulnerability prediction. Vulnerability prediction is a subfield of software security, aiming to predict software components that are likely to contain vulnerabilities (i.e. vulnerable components). Vulnerability prediction models (VPMs) are normally built based on Machine Learning techniques that use software attributes (e.g. software metrics) as input, to discriminate between vulnerable and neutral components.

Although the overall topic of vulnerability prediction is a relatively new area of research, several VPMs have already been proposed. As stated in [100], the main VPMs that can be found in the literature utilize software metrics, text mining (e.g. [197]), and security-related static analysis alerts (e.g. [73]) to predict vulnerabilities. Different studies have shown that text mining-based models exhibit better predictive performance in comparison to other State-of-the-Art techniques [100] [223] [233]. However, they perform poorly in cross-project prediction, which indicates that they are highly project-specific [233], while excessive amount of time and memory is required for their construction and regular application [100] [223]. Hence, VPMs that use software metrics (such as complexity, code churns, density of alerts etc.) may be a more viable solution in practice [223], as they are less expensive to build and apply, and they perform slightly better in cross-project prediction [233].

Although a large number of VPMs have been proposed over the years, none of them have managed to achieve a satisfactory trade-off among accuracy, practicality and performance [238]. In addition, their predictive performance in cross-project prediction has been found to be poor [210]. Hence, further work is required towards building VPMs that achieve a good compromise among accuracy, practicality, and performance, for example, through the combination of static analysis and Deep Learning, which have already demonstrated promising results. Accurate vulnerability prediction will help developers better allocate their effort and time, by focusing on parts that are more likely to contain vulnerabilities, instead of checking components in an arbitrary manner. This is also expected to enhance the efficiency of the vulnerability identification and mitigation approach followed by the development team.

Exploitable Vulnerability Identification. Although ASA tools are known for their effectiveness in detecting vulnerabilities early enough in the SDLC [147], they are underused in practice [101]. The main

[157] http://www.veracode.com/products/binary-static-analysis-sast
[158] http://www.veracode.com/products/greenlight

reason for this is their poorly presented results, in the form of long reports of raw warnings (i.e., alerts) [101]. These warnings need to be manually examined to determine whether they correspond to actual bugs that require immediate corrective actions (i.e. actionable alerts [80]) or they are just false positives. This process is normally called triaging [58] and it is highly time-consuming and effort-demanding. The large number of unactionable alerts (i.e. false positives), discourages developers from using them regularly throughout the overall SDLC.

Several attempts have been made in order to reduce the number of the produced warnings either by implementing more precise tools (e.g., [89]), or by post-processing the produced warnings to identify which of them are actionable. The latter mechanisms, which are normally known as actionable alerts identification techniques (AAITs) [81], constitute the most promising solution to the problem of false positive reduction so far. They utilize Machine Learning to discriminate between actionable and unactionable warnings, by using information about the warnings (e.g. type, severity etc.) or their surrounding code. Despite the multitude of the AAITs that have been proposed over the years [81] [159], almost no attempts exist focusing on the extension of these techniques towards the security perspective. More specifically, no significant contributions have been made emphasizing on the identification of ASA warnings that may correspond to potentially exploitable vulnerabilities.~~-~~Contrary to common software bugs that can be triggered at any time of the product execution, vulnerabilities can infringe a security policy only if they are exploited by malicious individuals. However, in order to be exploited they should be reachable from the entry points (i.e. surface) of the software product. In fact, the same vulnerability may have different severity based on its exploitability (i.e. reachability).

The idea of using reachability for assessing the exploitability of vulnerabilities reported by static analysis has already been highlighted by several research attempts. For instance, in [157] and [240] investigated the ability of determining the exploitability risk of software vulnerabilities based on software structure properties, including (a) the attack surface entry points, (b) the vulnerability location, (c) the presence of dangerous system calls, and (d) the reachability of the vulnerabilities. In fact, the adoption of reachability analysis constitutes the most complete attempt so far for the identification of exploitable vulnerabilities. However, more work is required in order to develop a reliable framework that will allow the accurate identification of ASA warnings that are likely to correspond to exploitable vulnerabilities. For this purpose, knowledge from the field of AAITs should be also leveraged and extended into the security realm.

4.4 Design-time Quality Assurance in Microservices Architectures

There are several distinct software life cycle phases preceding deployment, operation and retirement, namely, requirements, specification, design and implementation. These phases generally precede the deployment and running of software. For the purpose of this section we will consider quality assurance activities in several of these development-time phases. In the following we may refer separately to design and implementation phases or we may loosely refer to them collectively as design-time QA.

We note that there is a bit of a grey line between the final phases of development time and the beginning of run time, in particular, testing activities straddle these phases. Unit testing, though it involves execution of code, can reasonably be considered part of implementation, whereas system qualification testing may reasonably be considered as part of run-time quality assurance, as it executes software with its intended sets and in an environment that mimics its intended deployment. Certainly, the software doesn’t know the difference between testing and production execution, and on the basis of this we view system testing as the first of several run-time quality assurance activities, which are discussed further in Section 4.5.

Important Design-Time Quality Attributes in Service-Oriented Architectures (SOA). Zhang et al. [241] performed a systematic literature review to identify and synthesize knowledge on the quality management process for microservice architecture. Among others, the authors aimed to list the quality attributes that are deemed as important in Service-Oriented Architectures. The authors identified primary studies by browsing 3 digital libraries (ACM, IEEE, and Scopus, until June in 2017) and analysed in total 135 studies. The results suggested that 5 design-time quality attributes are relevant to microservices architecture: flexibility, maintainability, reusability, replaceability, and modifiability. Furthermore, Alshuqayran et al. [7], have suggested two additional quality attributes of interest, based on their secondary study on 33 studies. The (in total) seven quality attributes of interest are summarized in Table 7. In particular, for each quality attribute, we report: (a) the expected impact of each QA, when proper levels of quality are achieved; and (b) the definition of the QA in a recognized quality model, when applicable.

Table 7: Design-Time **Quality** Attributes

Quality Attributes	Impact	Definition in Quality Models
Flexibility	Flexible Configuration	Characteristics that allow the incorporation of changes in a design. The ability of a design to be adapted to provide functionally related capabilities [16]
Maintainability	Facilitated Maintenance and Evolution	The degree of effectiveness and efficiency with which a product can be modified to improve it, correct it or adapt it to changes in environment, and in requirements [95]
Reusability	Reusable Component	The degree to which an asset can be used in more than one system, or in building other assets [95]
Replaceability	Fewer Bugs When Replacing Another Version	The degree to which a product can replace another specified software product for the same purpose in the same environment [95]
Modifiability	Easy Change Through API	The degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality [95]
Independence	Reducing Complexity, Isolation, Loose Coupling, Decouple, Distributed, Containerization, Autonomy	The degree of interdependence between components [230]—**note**: referred as *coupling*

Quality Attributes	Impact	Definition in Quality Models
Modularity	Single Responsibility, Reduce Complexity, Separate Business Concern, Specialization, Customizable	The degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components [95]

Regarding the popularity of design-time quality attributes in SOA, Zhang et al. [241] suggest that flexibility is the most studied design-time quality attribute in the domain of microservices, followed by maintainability. The importance of maintainability has also been highlighted in three independent secondary studies [6] [57] [167]. Nevertheless, we need to note that all the aforementioned quality attributes are highly related, and in many standards or quality models, they are even considered as sub-characteristics of higher-level quality attributes. For example, ISO 25010 [95] defines 8 high-level quality attributes; among them maintainability, which is decomposed to 5 low-level quality attributes, including reusability, modifiability, and modularity. In addition, by focusing on the level of granularity at which these quality attributes are assessed, Daud and Kadir, suggested that the assessment can be performed at four levels of granularity: service, application, system, and others (e.g., middleware, workflow) [167]. Among them, service-level has been highlighted as the most common level of granularity, followed by system level. This result complies with results on general-purpose software engineering, in which the most frequent level of granularity are classes for object-oriented systems, and files for functional programming, followed by system-level assessments [10].

Metrics for Assessing Design-Time Quality Attributes in Service-Oriented Architectures (SOA). Hutapea et al. [94] performed a systematic literature review on design-time quality attributes and metrics in service-oriented architectures. The goal of this study was to identify: (a) the design principles that could act as quality attributes, and (b) the metrics for measuring them. To achieve this goal, they browsed five digital libraries: Scopus, IEEE, Springer, ACM, and ScienceDirect, from 2005 to 2018. Upon the completion of the search process, 15 articles have been selected for in-detail evaluation. The study led to 9 quality properties (lower level characteristics, compared to QAs) that have been adopted in the evaluation of service-oriented architectures, as defined in Table 8.

Table 8: Design-Time Quality Properties

Quality Properties	Definition
Cohesion	The degree to which the methods and attributes of a class belong together
Coupling	A measure of the interdependencies between different modules
Reusability	The degree to which a functionality of a service can be reused in the future
Composability	The ability of a service to work in a different context
Granularity	The number of functionalities encapsulated in a service
Complexity	The effort that required to maintain and to comprehend the implementation of a service or set of services
Design Size	The size of the system designed

Quality Properties	Definition
Business	What a company expects to accomplish over a specific period of time
Statelessness	Stating that services should not store specific information about activities, such as service requests, so that they can support low coupling, reduce memory requirements and enable scalability

We note that according to Hutapea et al. [94], reusability is considered as property (i.e., a sub-characteristic of maintainability), similar to ISO 25010 [95], but in contrast to Quality Model for Object-Oriented Design [16]. To quantify these 9 properties, 74 metrics were retrieved: we note that for some quality properties more than one metrics have been proposed. The mapping between metrics and quality properties is presented in Table 9, accompanied with the level of granularity at which the measurement takes place.

Table 9: An Overview of Metrics of Service-Oriented System

Source	Name	Scope	Quality Properties
[212]	Business Entity Convergence (VCONVE)	System	Business
	Lack of Cohesion of Service Operation 1 (LCOS1) Service Functional Cohesion Index Service Cohesion (SFCI) Lack of Cohesion of Service Operation 2 (LCOS2)	Service	Cohesion
	Service Cohesion (VCOHES)	System	Cohesion
	Service Message Coupling Index (SMCI)	Service	Complexity
	Service Composability Index (SCOMP)	Service	Composability
	Inter-Service Coupling Index (ISCI) Service Operational Coupling Index (SOCI)	Service	Coupling
	Service Coupling (VCOUPL)	System	Coupling
	Service Data Granularity (SDG) Process Service Granularity (PSG) Process Operation Granularity (POG) Service Capability Granularity (SCG)	Service	Granularity
	Service Reuse Index (SRI)	Service	Reusability
[154]	Cohesion Metrics (CM) Cohesion Factor (CohF)	Service	Cohesion
	Cohesion Factor (CohF)	System	Cohesion
	Total Complexity Metric of service (TCM)	Service	Complexity
	Complexity Factor (ComF) Total Complexity Metric of System (TCM)	System	Complexity
	Indirect Coupling (IC)	Service	Coupling

Source	Name	Scope	Quality Properties
	Coupling Factor (CoupF)	System	Coupling
	Direct Coupling Reusability (DC)	Service	Coupling Reusability
	Number of Services (NS) Number of Operation (NO)	System	Design Size
	Reusability Factor (ResF)	System	Reusability
[67]	Data Coupling Index (DCI)	Element	Coupling
	Normalized Data Coupling Index (NDCI)	System	Coupling
[5]	Average Service Operation Cohesion (ASOC)	System	Cohesion
	Average Service Operation Complexity (ASOM)	System	Complexity
	Service Operation Cohesion (SOC)	Service	Cohesion
	Data Granularity Score (DGS) Operation Function Granularity (OFG)	Operation	Granularity
	Average Service Operation Coupling (ASOU)	System	Coupling
	Average Service Operation Granularity (ASOG)	System	Granularity
	Operation Function Granularity (OFG)	Service	Granularity
[111]	Business Value of a Service (SBV)	Service	Business
	Weighted Granularity Level Appropriateness (WGLA)	Service	Granularity
[86]	Number of Human Tasks (NHT) Weighted Service Interface Count (WSIC) Service Support for Transactions (SST) Stateless Service (SS) Service Realization Pattern (SRP)	Service	Complexity
[87]	Density of Aggregation (DOA) Extent of Aggregation (EOA)	System	Complexity
	System's Service Coupling (SSC) Service Coupling Factor (SCF)	System	Coupling
	Number of Services (NS)	System	Design Size
[132]	Average Service Depth (ASD)	Service	Complexity

Source	Name	Scope	Quality Properties
[180]	Service Interface Data Cohesion (SIDC) Strict Service Implementation Cohesion (SSIC) Service Interface Usage Cohesion (SIUC) Loose Service Implementation Cohesion (LSIC) Service Sequential Usage Cohesion (SSUC) Total Interface Cohesion of a Service (TICS)	Service	Cohesion
[181]	Response for Operation (RFO)	Operation	Complexity
	Weighted Intra-Service Coupling between Elements (WISCE) Element to Extra Service Interface Outgoing Coupling (EESIOC) Weighted Extra-Service Incoming Coupling of an Element (WESICE) Weighted Extra-Service Outgoing Coupling of an Element (WESOCE)	Element	Coupling
	Total Response for Service (TRS)	Service	Complexity
	Total Structural Coupling of a Service (TSCS) Extra-Service Incoming Coupling of Service Interface (ESICSI) Service Interface to Intra Element Coupling (SIIEC) Total Weighted Intra-Service Coupling (TWISC) Total Weighted Extra-Service Coupling of Elements (TWESCE) Total Weighted Extra-Service Indirect Coupling (TWESIC) Total Structural Coupling of an Element (TSCE) Total Structural Coupling of Service Interface (TSCSI)	Service	Coupling
	System Partitioning Factor (SPARF) System Purity Factor (SPURF) Total Structural Coupling of a Service-Oriented System (TSCSYS)	System	Coupling

Based on the findings of Hutapea et al. [94], **coupling** was the most studied quality property, followed by **cohesion** and **complexity**. As expected, among the 74 identified metrics, 33% corresponds to metrics that are developed for assessing coupling. However, there are still some quality properties for which no metrics have been still introduced, namely: composability and statelessness.

Assessing Maintainability in Service-Oriented Architectures (SOA). Bogner et al. [29] conducted a literature review on metrics that are able to assess the maintainability of microservice-based systems. More specifically, the aim of this study was to investigate if there are quality metrics that automatically assess the maintainability of service-based architectures. To achieve this goal, the authors searched in three digital libraries: ACM, IEEE, and Google Scholar, and identified 8 relevant papers. By studying these papers, Bogner et al. [29] suggested that the most common quality properties that are related to maintainability are: size, complexity, coupling, and cohesion. Table 10 presents a selection of maintainability metrics for the service-based architectures found in the literature.

Table 10: Maintainability Metrics for SOA

Name / Abbreviation		**Scope**	**Property**	**Source**
Number of Services Involved in the Compound Service	NSIC	Service	Complexity	[194]
Services Interdependence in the System	SIY	System	Coupling	
Absolute Importance of the Service	AIS	Service	Coupling	
Absolute Dependence of the Service	ADS	Service	Coupling	
Absolute Criticality of the Service	ACS	Service	Coupling	
Weighted Intra-Service Coupling between Elements	WISCE	Element	Coupling	[181]
Weighted Extra-Service Incoming Coupling of an Element	WESICE	Element	Coupling	
Weighted Extra-Service Outgoing Coupling of an Element	WESOCE	Element	Coupling	
Extra-Service Incoming Coupling of Service Interface	ESICSI	Service	Coupling	
Element to Extra Service Interface Outgoing Coupling	EESIOC	Element	Coupling	
Service Interface to Intra Element Coupling	SIIEC	Service	Coupling	
System Partitioning Factor	SPARF	System	Coupling	
System Purity Factor	SPURF	System	Coupling	
Response for Operation	RFO	Operation	Complexity	
Total Response for Service	TRS	Service	Complexity	
Service Interface Data Cohesion	SIDC	Service	Cohesion	
Service Interface Usage Cohesion	SIUC	Service	Cohesion	
Service Sequential Usage Cohesion	SSUC	Service	Cohesion	
Strict Service Implementation Cohesion	SSIC	Service	Cohesion	
Loose Service Implementation Cohesion	LSIC	Service	Cohesion	
Total Interface Cohesion of a Service	TICS	Service	Cohesion	
Service Granularity	SG	Service	Complexity	[186]

Name / Abbreviation		Scope	Property	Source
Relative Coupling of Service	RCS	Service	Coupling	
Relative Importance of Service	RIS	Service	Coupling	
Service Coupling Factor	SCF	System	Coupling	
Service Coupling Factor	SCF	System	Coupling	[87]
System's Service Coupling	SSC	System	Coupling	
Extent of Aggregation	EOA	System	Complexity	
System's Centralization	SCZ	System	Centralization	
Density of Aggregation	DOA	System	Complexity	
Aggregator Centralization	ACZ	System	Centralization	
Weighted Service Interface Count	WSIC	Service	Size	[86]
Stateless Services	SS	System	Complexity	
Service Support for Transactions	SST	System	Complexity	
Service Realization Pattern	SRP	System	Complexity	
Number of Services	NOS	System	Size	
Service Composition Pattern	SCP	System	Complexity	
Service Access Method	SAM	System	Complexity	
Dynamic vs. Static Service Selection	DSSS	System	Complexity	
Number of Versions per Service	NOVS	System	Complexity	
Average Number of Directly Connected Services	ADCS	System	Coupling	[207]
Inverse of Average Number of Used Message	IAUM	System	Cohesion	
Number of Operations	NO	System	Size	
Number of Services	NS	System	Size	
AVG # of Operations to AVG # of Messages	AOMR	System	Size	
Coarse-Grained Parameter Ratio	CPR	System	Size	

Assessing Reusability in Service-Oriented Architectures (SOA). Regarding reusability, we have been able to identify two studies. First, Choi and Dong-Kim [40] proposed a model for evaluating the reusability of service-oriented architectures. Specifically, the authors selected five quality properties of reusability based on the key features of services, as well as metrics for assessing them (see Table 11):

- *business commonality*: the degree to which functionality and non-functionality of the service are commonly used by consumers in a domain
- *modularity*: the extent to which a service provides independent functionality without relying on other service
- *adaptability*: the capability of the service to be well-adapted to different service consumers

- *standard conformance*: the extent to which a service conforms to the widely accepted industry standards such as Organization for the Advancement of Structured Information Standards, etc.
- *discoverability*: the extent to which the service, service consumers expect to look for, is easily and correctly found

Table 11: An Overview of Metrics for Evaluating Reusability of SOA

Quality Attribute	Quality Metric
Business Commonality	Assesses the degree of functional commonness and non-functional commonness of the service in a business domain
Modularity	Assesses the degree to which a service is independent on other services
Adaptability	Counts how many variation points can be adapted as the consumer wants them to be
Standard Conformance	Captures the degree to which the service conforms to the relevant standards
Discoverability	Quantifies the extent to which the service is easily and correctly discovered by consumers

To aggregate these measures, under a common reusability index, the following formula is used:

$$RE = BCM * (MD * W_{MD} + AD * W_{AD} + SC * W_{SC} + DC * W_{DC})$$

$W_{\#\#}$ is the weight for each metric.

Next, Lee et al. [127] extended the previous model for evaluating software as a service (SaaS) in cloud computing. In this study Lee et al. [127] studied various quality attributes, among which reusability was the only design-time one. For measuring the reusability, three independent quality metrics were defined, which however are not synthesized in a single measurement. We note that the application of the metrics has as a precondition the existence of a family of SaaS applications. In families of products commonalities refer to features that exist in all products of the family, whereas as variabilities to features that are used only in a portion of them, or they are used in a different form.

- *Functional Commonality*: Measures an average of commonality of each functional feature defined in a target SaaS. Commonality of each functional feature can be measured by calculating the degree of family members who use each functional feature
- *Non-functional Commonality*: Measures the average of commonality of each non-functional feature
- *Coverage of Variability*: Measures how many of the variation points included in the domain are actually realized in the SaaS

4.5 Run-time Quality Assurance in Microservices Architectures

Run-time quality assurance seeks to achieve freedom from run-time errors. We will first identify a collection of quality attributes that may properly be the subject of run-time quality assurance. Then we consider the mechanisms that are typically employed to achieve specific objectives with respect to these quality attributes. Finally, we will consider the techniques and tools that we propose to employ, in conjunction with the design-time measures described elsewhere in this document, to assure that specific agreed quality properties within these quality attributes are continuously achieved at run-time.

4.5.1 Run-time Quality Attributes

We previously presented in Section 4.1 five key system characteristics identified by the IISF: *security*, *safety*, *privacy*, *resilience* and *reliability*. Even though they must be identified as requirements and be realized by design, all of these characteristics are manifest as run-time quality attributes through the system's behaviours. In this section we explore run-time measures for quality assurance of these characteristics. In addition to the five general quality attributes, we will identify a list of additional more specific attributes and properties that may later be shown to be instances or subclasses of these.

Run-time quality attributes relating to microservices. Several general quality attributes relevant to microservices are identified in Table 12.

Table 12: Quality Attributes Relevant to Microservices

Microservice-specific quality attributes	Definition
Integrity of microservice functionality	No tampering with microservice code or operation. Without the ability to assure the integrity of the functionality and execution of an individual microservice it is not possible to build a predictable MSA.
Integrity of microservice composition	No tampering with microservice interoperation. The integrity of a composition of microservices is protected.
Integrity of microservice identification	The microservice you get is the one you expect to get. The binding of microservice names, description, and implementation is maintained with integrity.
Confidentiality of microservice user data	User data is not disclosed to unauthorized agents
Integrity of microservice user data	User data may not be modified by unauthorized agents
Availability of microservice applications	A microservice application can be used when needed
Correctness of microservice implementation	The behaviour of the microservice implementation is faithful to its specification
Correctness of microservice composition	The behaviour of a composition of microservices is faithful to its specification (or composition of specifications of individual microservices)
Correctness of microservice-based application	The behaviour of the application that employs a microservice or a composition of microservices is faithful to its specification.

Clearly, while these attributes are not exclusive to microservices (they apply to any SOA), they represent an instantiation of generic attributes in the context of microservices that may be manifest as distinct quality properties or measured by distinct methods and metrics. While the "correctness" attributes must be addressed at design time, such as correctness-by-construction, they are tested at run time, and it may be feasible to monitor that the relevant properties continue to hold at run time. The precise definition of the properties or metrics will depend upon the specific correctness conditions and the particular microservices framework in use. After we have identified specific quality attributes and quality properties for microservices, to realize our run-time verification quality assurance approach for microservices we will need to specialize our methods to the implementation of the microservices framework. Quality properties that are important for SmartCLIDE will be determined as 'User Quality Stories' when specifying the claimed applications.

Run-time processes to assure quality attributes. We consider techniques and tools that may be employed for run-time quality assurance of identified quality attributes. In this document, we refer to run-time quality attributes through the term *Runtime Verification* [17]. Before describing runtime verification further let us briefly discuss a few other forms of runtime quality assurance. The most common form of run-time quality assurance is, of course, *testing*. Testing is used in conjunction with design-time quality measures to confirm that correctness of implementation has been achieved by comparing the output or result of run-time execution of software with expected results. The quantities observed by testing typically are confined to the outputs that software has been written to provide. Disciplined and thorough testing involves considerable preparation, and the run-time execution of the test cases usually is preceded by careful considerations of the range of supplied test inputs, as well as analysis of the depth and breadth of testing. In domains where the run-time behaviour of software is particularly critical, such as in safety-critical domains such as aviation, testing discipline may entail the use of instrumentation to obtain run-time measures of the code covered by a particular test suite to help ensure that the testing is adequate.

Debugging, another kind of run-time quality assurance, on the other hand may employ tools that enable a developer to examine deeper details of the run-time operation of software than is available solely from the outputs, leading to enhanced ability to understand to find subtle errors when things go wrong. Testing and debugging are well explored and as an extension of the design and implementation process will not be considered further in this exploration of run-time quality assurance approaches.

Security auditing, considered a security function, is the gathering of run-time security-related events into a *security audit log*.[159] The audit log is to serve as a record of events, which may represent things that are supposed to happen and/or things that are not supposed to happen, or events that are otherwise relevant to security. The log is indispensable for accountability and for forensic analysis. It is a tailored form of runtime monitoring, in which the events are predefined, the sensors for the events are coded in-line within the implementation of the security functions, and it may be configured to record a subset of many defined events during any particular period of time. An audit system provides a persistent high-integrity store for the audit log and possibly additional functions for querying or analysing the audit log.

Monitoring systems, generally, are used to gather information about the operation or performance of an information system, network, hardware, application, or a combined manual/automated process, in order to provide a basis for analysis to detect problems with or to improve the process being monitored. To enable monitoring, some kind of sensors need to be installed within the process or system being monitored to provide the raw measurements and events of interest to the monitoring system. As we shall see in the following section, monitoring is an important component of runtime verification. In a recently completed project, CITADEL[160], the monitoring plane provided a framework for the implementation of monitor applications that employed virtual sensors in applications and the platform to generate events and alarms to the adaptation and reconfiguration planes to perform adaptive reconfigurations of the system.

[159] Note that elsewhere in this document we have also used "security auditing" in a different sense: rather than the runtime security service of a security audit system, it is also used to signify the security-related activity of taking stock of security practices and mechanisms that are applied.

[160] CITADEL Project, European Commission H2020, 2016-2019.

CITADEL also developed monitor synthesis techniques to create monitors from the properties attached to architectural elements of a system model.

Health monitoring, commonly used in safety-critical systems, comprises a set of sensors and activities undertaken to maintain a system in operable condition. The health monitoring system observes the state or communications of system components to detect when some action is required to restore the system to nominal operation and to initiate such action. For example, a critical component of a system may generate a periodic *heartbeat* that is detected by the health monitor. If the heartbeat is not detected for a given period of time actions are taken to restart the component.

4.5.2 Runtime Verification (RV)

Runtime Verification is a computing system analysis and execution approach based on extracting information from a running system and using it to detect and possibly react to observed behaviours satisfying or violating certain properties. Some very particular properties, such as data-race and deadlock freedom, are typically desired to be satisfied by all systems and may be best implemented algorithmically. Other properties can be more conveniently captured as formal specifications. Runtime verification specifications are typically expressed in trace predicate formalisms, such as finite state machines, regular expressions, context-free patterns, linear temporal logics, etc. or extensions of these. However, any mechanism for monitoring an executing system is considered runtime verification, including verifying against test oracles and reference implementations. When formal requirements specifications are provided, monitors are synthesized from them and infused within the system by means of instrumentation. Runtime verification can be used for many purposes, such as security or safety policy monitoring, debugging, testing, verification, validation, profiling, fault protection, behaviour modification (e.g. recovery), etc.

Runtime verification (RV) is like formal verification (a design-time quality assurance activity) but using the actual system implementation and its execution as a substitute for a formal model that provides the predictive and inductive source for permissible evolution of a system. Thus, RV dynamically examines actual executions of a system rather than statically analysing all possible executions as allowed by the formal model.

Some researchers have incorporated runtime verification as an *aspect*, in the sense of aspect-oriented programming (AOP), of the application being developed. Indeed, aspect-oriented programming arose because concerns such as logging, policy enforcement, security management, profiling, trace visualization, and verification are typically not implemented in a modular fashion, but rather are cross-cutting concerns [65]. AOP approaches this by allowing cross-cutting concerns to be implemented by cross-cutting aspect modules. RV is based on finite traces of events generated by the run-time system. A set of events is defined in advance for the run-time system, and instrumentation is built within the system to generate events under appropriate circumstances. A property corresponds to a set of traces with some common characteristic. A specification is a textual construct that describes a property. The instrumentation, in the form of virtual sensors, are capable of being connected to one or more monitors. There are several steps that need to be completed internally within the RV system:

1. Create a specification of the desired (or undesired) behaviour in terms of event traces
2. Synthesize a monitor from the specification
3. Determine the instrumentation needed to support the monitor
4. Configure the instrumentation in the system

5. Attach the monitor to the instrumentation in the system

These steps should go hand-in-hand with the creation of a microservices-based application and the deployment of the application within the microservices run-time environment.

5 Artificial Intelligence for Software Development

This section focuses on the use of Artificial Intelligence (AI) on software development problems. An overview of AI is added to the background, describing basic concepts which will be observed later on, or taken as a model for the development.

5.1 Background Information

5.1.1 Artificial Intelligence

The idea of creating machines capable of showing an intelligent behaviour is prior to computer science. The origin of AI as a scientific field is related to Alan Turing's work [228], in which he stated several principles to determine is a machine is, in fact, intelligent. These criteria are called Turing's test.

A lot of problems have been tested with AI, from which some useful tasks have arisen:

- Reasoning and problem solving, by imitation of step-by-step human reasoning
- Knowledge representation, storing rules or conditions which represent experts' knowledge, or problem-solution items to solve problems not faced before
- Creativity, by using neural networks to create art
- Natural Language Processing, to enhance communication between man and machines, allowing automatic translation or sentiment analysis
- Conversational AI, creating systems capable of imitating human behaviour and communication
- Perception, interpreting information incoming from sensors to deduce aspects from the real world, as human face recognition, audio transcription, etc.
- Learning, designing systems which become more efficient by acquiring experience
- Planning, which allows intelligent systems to select the best actions to achieve their goals
- General-purpose AI, whose purpose is to unify different approaches and methodologies to overcome human performance

5.1.2 Machine Learning

Machine Learning is a set of techniques and algorithms which give machines the ability to perform a task without being explicitly programmed for it. It began with high-level symbolic representations of problems and knowledge: the era of expert systems, which used a set of rules to interact with symbols the same way if/then clauses work in programming. The problem was the inability to learn autonomously from the data. Machine Learning came to solve this limitation. The k-nearest neighbour algorithm [48] invention in 1967 made feasible to infer a solution to an unseen problem by checking resolved problems in the past. The introduction in 1995 of Support Vector Machines (SVM) represented the State of the Art during years until the appearance of Deep Learning, in terms of pattern recognition. The same year, the random forest method would be applied extensively to a wide variety of problems, particularly data mining. The emergence of richer datasets would allow to create better models, and the improvements in hardware to resolve harder optimization problems

Classification. Machine Learning methods can be categorized depending on several features

- Depending on the task to solve
    - *Classification*. Classification algorithms are specialized in assign categories to unseen samples. In these cases, tags are usually text, and can be replaced by numeric ones. These algorithms can be sub-divided in binary classification and multi-class classification. Examples: Gender classification, handwritten digit recognition.
    - *Regression*. Regression algorithms try to associate a numeric value to each sample. Usually tags linked to each sample are real numbers, although integers can also be seen. There is no need to make an absolutely precise prediction to consider the model is working correctly, as the values, distinct but close, refer to similar realities. Examples: Predict the age of people in pictures, predict fuel prices.
    - *Dimensionality reduction*. These algorithms do not try to predict any aspect of the samples. Their purpose is to generate an alternative representation of the samples to perform another task. Usually this transformation consists in reducing the number of dimensions or features which conform the sample, usually by combining the original ones. This allows the visualization of 2D/3D graphs and later apply classification or regression algorithms, which have problems with high dimensionality samples.
    - *Clustering*. These algorithms group samples with similar features. The main difficulty to be solved is to find a similarity measure among them as, for example, the Euclidean distance.
- Depending on the tagged data or target values
    - *Supervised learning algorithms*. These require, along with the design matrix, a vector with the correct tags for each sample. To evaluate the algorithm, it is necessary to measure its performance against a different dataset than the one used for training with unseen samples. In other words, its generalization capability.
    - *Unsupervised learning*. These do not need a vector with the correct tags during the training phase. They usually classify the samples depending on basic statistic features and different similarity features. Either way there are also unsupervised algorithms for dimensionality reduction and feature extraction.
- Depending on the use of parameters. The parameters of an algorithm are those which determine its behaviour, and that are learnt automatically during training phase. Hyperparameters also determine the behaviour of the algorithm, but their abstraction level is higher and their values cannot be, usually, learnt.
    - *Parametric*. A parametric model sums up training data with a fixed size dataset. The number of parameters used does not change depending on how big the training dataset is.
    - *Non-parametric*. In a non-parametric model, the dimension and number of training samples determine the set of parameters to use. This way, the algorithms are capable of reaching significant rates of accuracy because they take into consideration the full variability present within them.

Selection of Hyperparameters. As already said, hyperparameters are those parameters which control the algorithm behaviour at a high abstraction level. They cannot be learned automatically by the algorithm over a training dataset. The most straightforward approach to adjust them is to apply a procedure called grid search, over a subset of hyperparameters defined by the user. Hyperparameters have to be tested all at once because of the interactions amongst them.

- A subset of data –validation set- is used to test different combinations of hyperparameters and adjust them. Thus, the available dataset is divided into three sets: training, validation and test –usually in a 60%, 20%, 20% proportion respectively-.
- Subsequently a grid search is performed to evaluate the different hyperparameter values against the validation subset, e.g. the number of layers, activation function, etc.
- To get an estimation of the efficiency of the algorithm, the model is evaluated against the test dataset. It is also possible to train algorithm over the union of training and validation set, to later assess it over the test one, which it usually provides better results.

Data Pre-processing. A key aspect to the correct functioning of Machine Learning is to apply a pre-processing to the available data. A common task is to transform text attributes into numeric ones. The easiest way to do this is assigning series of binary characteristics, each of them associated to one of the text ones (i.e. in an attribute called "colour" which possible values are "green", "red" and "yellow", if we wanted to define a yellow item the values would be 0,0,1). This method is known as *one-hot encoding*.

With regard to the distribution of characteristics, there are several alternatives to normalize data depending on which algorithm is to be used later on. This addresses the problem of very different ranges amongst features. In addition, some algorithms require that all features' averages are 0, or other constraints in terms of data distribution.

- *Min-max scaling*. Each feature is translated individually to fit the [0, 1] interval.
- *Data standardization*. Features are standardized individually by subtracting the average value and scaling to unit variance
- *Normalization*. Consists of normalizing each feature to have unit length with respect to a norm. It is especially used in text classification.

It is remarkable that if any normalization method is applied to the training set, then the same transformation will have to be applied to the test one.

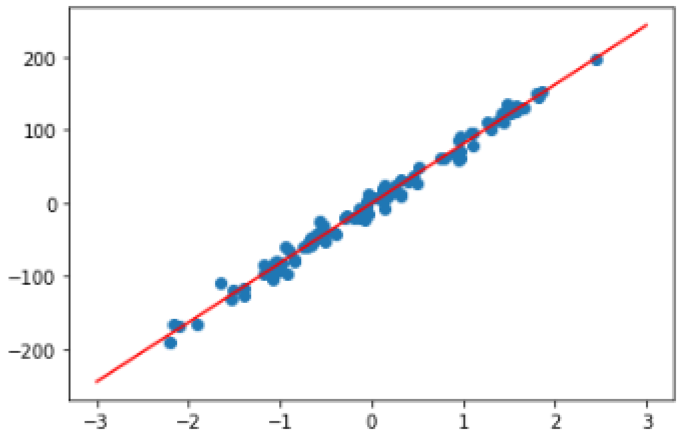

Figure 11: The linear regression model

Linear Regression. This is one of the simplest methods of Machine Learning. It is supervised and parametric. Its purpose is to predict a continuous numeric value associated with each sample of a set, e.g. finding the price for a dwelling based on its surface, placement and age. It tries to model the objective as a linear combination of the inputs. Usually a constant called intercept is added, independent from the inputs. The aim is to find w weights to throw a good prediction, which is made from the training dataset. First, we need to define a function to evaluate the quality of the predictions: the cost function. Minimum Square Error is the choice for this; by calculating the average of the differences between model output and the target values for every sample in the training set. Once the error function has been defined, the w vector of parameters which minimize that function can be found –a main feature of this optimization problem is that it is convex and, hence, the first derivative can be equated to 0 to find the minimum values-.

K-Nearest neighbours. It was one of the first classification models proposed in the field of Machine Learning. It is supervised and non-parametric. It belongs to the lazy learning paradigm, where the model's generalization is extracted once the query is thrown –instead of prior to the testing phase-. Its behaviour is based on the premise that samples belonging to the same class are closer than the ones belonging to others. Therefore, the distance between points is given by a function chosen by the user; at training stage algorithm stores samples along with their class tag. When a new sample is taken to the model two steps are performed: 1) the k closest samples to the sample are found according to the chosen distance, and 2) a prediction is returned among the k closest samples (neighbours). The selection of the parameter k is crucial to determine the efficiency of the model. In terms of the distance function we have several options:

- *Euclidean distance*. It matches the intuitive idea of the distance between two points, but generalized for a random number of dimensions.
- *Squared Euclidean distance*. Similar to the Euclidean –square root is deleted-, it is used as an alternative due to its simplicity and akin properties.
- *Manhattan distance*. It matches the idea of reaching from point A to B following a grid.
- *Chebyshev distance*. It corresponds to the biggest difference amongst the features of two vectors.

There is no thumb rule to select one distance or another; the choice has to be made based on problem domain knowledge or a hyperparameter selection process. The biggest problem with this algorithm is the computational cost, which is proportional to the number of samples used.

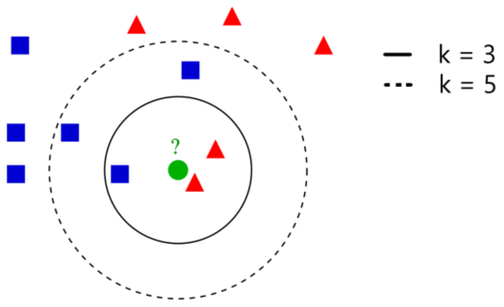

Figure 12: K-nearest neighbours' problem

Support Vector Machines (SVMs). One of the most popular methods in Machine Learning is the SVMs or Support Vector Machines [47]. It is a classification algorithm, supervised and parametric. This model uses a similar function to the one used in linear regression to solve binary classification problems. In particular, the model outcome is a tag between 0 and 1, which is calculated as the sign of the weighted addition of the features from a dataset plus a constant. It is necessary here to define the *decision boundary*. The decision boundary consists of the subset of points which are at the classifier edge, that is, the points for which there is a transition between the prediction of a class and another. These points define a hyperplane. Thus, SVMs training will consist in finding the weights for which the hyperplane separates samples of both types, having the biggest distance amid both the samples and the hyperplane. This is why they are called maximum-margin classifiers. The technique which allows SVMs to fit non-linear classification problems is the kernel trick, founded on the idea that several Machine Learning problems can be rewritten in terms of products between samples: the *representer theorem* [203]. This way a non-linear transformation can be applied to the input dataset to obtain a linearly separable problem. That transformation is called *kernel*. The kernel multiplied by an alpha factor gives a new extended space. It's the calculation of this alpha factor which has to be calculated in terms of SVMs for non-linear problems. Once again, the selection of the kernel function along with its parameters has to be based on the user's experience or a hyperparameter protocol selection. Another downside for this method in case of the non-linear problems is the need to calculate al the kernels to deliver a prediction, which makes it computationally expensive.

Since this algorithm only faces two categories of classification problems, there are several approaches that have arisen to observe a broader scenario [93]. The most common ones are:

- *One vs all / One vs rest*. A number of SVMs equal to the number of classes is trained, one as positive for each class and the rest as negative. To determine which class is predicted, the input is processed by all the SVMs and the one with the highest output value is taken.
- *One vs one*. A SVM is trained with every possible pair of classes. To determine which class is predicted, each SVM votes between the two that has evaluated. The class with more votes is chosen.

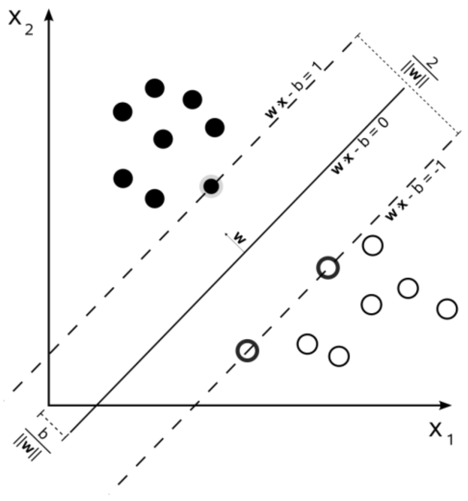

Figure 13: SVM Hyperplane Definition

Principal Component Analysis (PCA). This is one of the most important and most employed algorithms in Machine Learning and applied statistics. Its purpose is dimensionality reduction, and it is unsupervised and parametric. Dimensionality reduction algorithms try to process data by reducing the number of features, with the aim of visualizing them, use another algorithm or make easier the resolution of the problem. PCA is a linear algorithm, which means that the resulting features will be a weighed addition of the input features. This is done by multiplying the sample dataset by the weights' matrix. As PCA is unsupervised it does not use target values nor class tags, but analyses patterns and distributions in the input data.

Input data have to be centred (pre-processed) to have an average value equal to zero. Furthermore, all input features have to acquire a similar value, where the variance amongst the output features is taken to a maximum. To this effect PCA looks for directions into the input features in which data are more variable, given that these directions have to be orthogonal to each other and unit-length. The directions are known as Principal Components, and they will be the columns of the aforementioned matrix. The result of the product of the input features by the matrix is the projection of the features over the Principal Components. The number k of Principal Components determines the dimensionality of the output, and is given by the user. Particularly important is the fact that if the training data have been centred, the test ones have to be centred too. This algorithm was specially used for facial recognition (plus other classification algorithm), task in which it competed with Linear Discriminant Analysis (LDA) [21], mainly due to the fact that PCA takes the variant as main resource of useful information. This was prior to the Deep Learning paradigm.

5.1.3 Neural Networks

Neural networks (NNs) are a computation model based on an inter-connected net of artificial neurons. Each neuron represents a processing unit, which simulates -to a point- the processing of biological neurons. This –artificial- neurons have a series of inputs and an output, which is usually a non-linear function of the input. In particular, each of the inputs is multiplied by a synaptic weight. Thus, these weights determine neuron's behaviour. History of neural networks has fluctuated depending on the limitations due to the different models.

- *First stage* (1949-1970). Probably the first model was the McCulloch-Pitts [145] neuron. The output was a weighted addition of the inputs followed by a non-linear function. The problem was that weights had to be established manually, instead of being learnt. Some years later Frank Rosenblatt proposed the Perceptron [192], in which weights could be learnt on the basis of a training dataset. Afterwards the Adaptive Linear Element (ADALINE) [235], could infer its weights correctly based on a training dataset and the gradient descent algorithm, which is the basis for today's Neural Networks weight adjustment. Marvin Minsky truncated this model by demonstrating that XOR function could not be solved, setting off the first Neural Network Winter for 15 years.
- *Second stage* (1986-1995). The back-propagation algorithm allowed to train multi-layer Neural Networks [195]. In this model one layer's outputs feed the next layer's inputs. Thus, non-linear function problems could be resolved and hence non-linearly separable classification problems. The introduction of SVM and some other limitations respecting to the expectations created the second Neural Network Winter.
- *Third stage* (2006-Nowadays). The problem with deep NNs (>2 layers) was that they were thought too hard to train, particularly due to hardware limitations. Geoffrey Hinton demonstrated that a particular NN model could be trained by greedy layer-wise pretraining [85]. This was the starting point to an ever-growing interest in NNs, probably due to the continuous increase in: a) models size b) available data c) computation capabilities and use of specialized hardware (GPUs), and d) applicability and effectiveness of models with respect to commercial applications.

ADALINE. ADAptive LInear Element [236] is a one-layer neural network whose parameters or weights are adjusted iteratively based on a set of training dataset. Taking a single layer, single neuron model, it will be capable of distinguishing between two patterns or classes. The addition of more neurons to the layer will provide, in a one vs all way, distinction amongst several classes.

Each neuron is a computation unit whose behaviour is determined by its weights. Several inputs are weighted by multiplying by those, later on added to a value independent from the inputs (the bias) to get the activation value. This value is not constrained, so it must be transformed to a discrete one. The operation is provided by the sign function which returns 1 for positive values and -1 for negative ones. In so doing, the output of the neuron can be read as the tag predicted for the input. That function is the activation function, which depends on the problem to resolve.

But the main advantage of ADALINE versus the old McCulloch-Pitts neuron [145] is the capacity to infer proper weight values taking the training dataset as a base. This is performed by setting weight calculation as an optimization problem, solved in an iterative way. A cost function is needed (loss function) to measure the error committed by the neural network with respect to the training set. Minimum Square Error (MSE) is usually the selected function to this effect, in order to make its value grow independently of the sign by means of the network error. The optimization problem over the error function is tackled by the gradient descent method, which by slight modifications in the weight values will eventually take the network error to a minimum. These modifications, taken as increments, are found by multiplying the calculated values from the cost function partial derivative with respect to the weights by a constant called learning rate which controls the learning speed of the neural network. A balance between iterations and accuracy must be found. Too small values will get no improvement, too big values will not allow the error to decrease. As the problem is convex, having a small enough learning rate and a linearly sortable problem, the model will be able to classify all the test samples correctly [169].

In brief, if a multiple classification problem has to be solved more neurons can be added, each one associated to one of the classes. Thus, when a sample is thrown to the network, the neuron with the class associated to the prediction will have the higher value. In this case, prediction class' detection activation function cannot be discrete, so an identity function is used. Apart from that the weight vector will be transformed to a matrix created with the one-hot encoding method, where each input represents the desired output for each of the neurons.

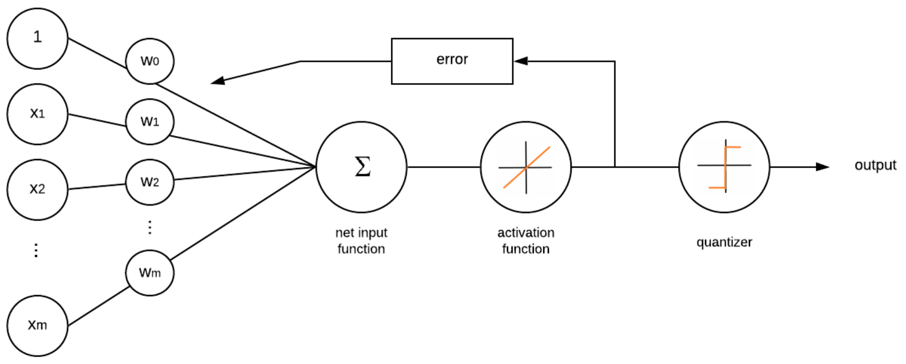

Figure 14: ADALINE Neuron

Perceptron. Perceptron is older than ADALINE and has minor differences with it, the main one being that uses class (discrete) labels to learn model weight (while ADALINE uses continuous values to learn the weights, which is more accurate).

Particularly interesting and widely used is the **Multi-Layer Perceptron** (MLP or vanilla neural network), which uses a *Feed Forward* [161]architecture. It appeared as a response to the inability of one-layer NNs to solve non-linearly separable classification problems –those in which categories cannot be sorted by a hyperplane-. The *backpropagation*[162] algorithm brought the solution by allowing more than one-layer

[161] Feed forward models are those in which information goes always from the input to the output

[162] Backpropagation algorithm works repeating iteratively two phases: propagation and weight modification. A sample is thrown into the network. After going through it, the output is compared to the ideal output to get the error values. Then the errors are back-propagated using the derivatives chain rule, to get the error value for each neuron. These values are used to calculate the gradient of the error with respect to the weights, to finally adjust them.

NNs training, as seen gradient descent optimization process becomes far more complicated. MLPs are composed by an input layer, one or more hidden layers and an output layer. The purpose of the hidden layers is to generate a series of non-linear characteristics which allow the output layer to resolve non-linearly separable problems, hence, the activation function is non-linear. The main feature of these models is that those intermediate characteristics are not designed manually, but learnt from the training dataset by adjusting the hidden layer weights –thence the black box model name, intermediate characteristics are nor explainable nor interpretable by a human-. Normally, the functions employed are hyperbolic tangent, sigmoid and Softmax amid others [170] depending on the problem. These models in combination with their functions are fairly easily implementable by the tools mentioned below, but the main idea is that they are the predecessors to more specialized ones like Convolution Neural Networks or Recurrent Neural Networks.

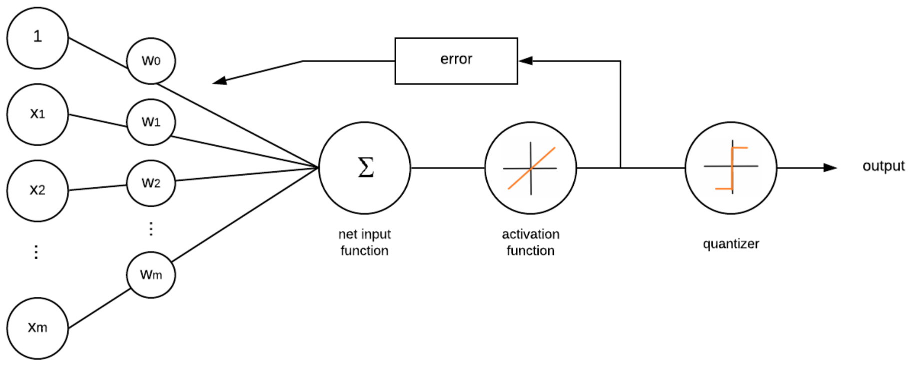

Figure 15: Perceptron Neuron

Deep Learning. Deep Learning concept refers to training neural networks with more than two hidden layers. The problem of *vanishing gradient* [163] and to a lesser extent exploding gradient, when neuron layers are added, stopped the development of deep NNs for several years, although backpropagation algorithm was already developed [121]. Since these problems have been as aforementioned solved to a great extent by mathematical toolkits, Deep Learning refers to training Neural Networks (NN) independently from how deep the neural network is. Nonetheless actually a lot of business problems can be solved by using two or three hidden layers in the middle of input and output ones. Although NNs were dwarfed by the effectiveness of alternatives such as SVM, the improvements in hardware–special mention GPUs-, the increment in rich datasets given by increasing sources and the refining of the algorithms have caused Deep Learning to break out with a high success rate.

Convolutional Neural Networks (CNNs). These NNs take an image as input and return another as an output. Instead of by means of a weight matrix, the behaviour of each neuron is given by a series of matrix called convolution filters, usually smaller than the input images.

By way of illustration [124], an image will have three channels (RGB) each of one with associated two-dimensional matrix (pixels). Each neuron will have a bias too, as in the prior NNs, which will be added to each element of the convolution matrix. The convolution operation consists in applying a filter to the input on each channel, dragging this filter all over the input image's surface to get the output image. The

[163] The backpropagation algorithm, by the derivative chains rule, multiplies several gradients to calculate the ones in the first layers, making them decreasing exponentially and thus making first layers' neurons learn too slow

filter displacement units can be set as a hyperparameter called stride, which in case of being more than one will reduce the size of the convolution's resulting matrix. The zero-padding technique consists in adding zeros in case the filter on its way falls off the edge of the image. Finally, an activation function, commonly *ReLU* (Rectified linear unit) [3] by de-facto standard, will be applied element by element.

Thus, a multi-channel image can derive into a single one (features map) depending on a series of filters which determine the neuron's behaviour. Same way as the conventional neurons, convolutional ones are grouped by layers. In this manner, output from a convolutional neuron layer will be a series of features map of the same size (as many as neurons).

It is possible to stack one convolutional layer after the other; as convolutional neurons take like input an image with a random number of channels, the features map generated by one layer can be the input of another one, building this way a deep convolutional network.

There are some other types of layers, the pooling ones. These layers' aim is to reduce the number of parameters to maintain by the network and hence the computational cost: the gradual reduction of the images spatial dimension will allow to get features in an increasing level of abstraction. Typically, 2 by 2 filters with a 2-unit stride are used to get the maximum value of each position. Pooling layers operate independently over each features map generated by the prior layer. A characteristic convolutional network will alternate between regular convolutional layers and max-pooling ones. Additionally, standard neuron layers (fully connected) [133] can be added as final layers to apply a *Softmax* [170] function in order to emit predictions about the input images.

Finally, a technique has to be mentioned by its interest and efficiency to the training of Deep Learning models: the *dropout* [85]. It aims to avoid overfitting in networks to get a correct model generalization. Essentially it consists in fixing to zero the values of random neurons at each training iteration. Once the iteration id finished, the neurons work as they usually do and another set of random neurons is chosen. This way the co-adaptation of one neuron to another maintaining the generalization capacity (the skill of working correctly when facing unseen samples). Most libraries allow to set different percentages of neurons to drop and in which layers should it be applied. Dropout technique can be applied in normal layers and convolutional/pooling ones.

Recurrent Neural Networks (RNNs). Recurrent neural networks are used to label, classify or generate sequences. A sequence is a matrix each row of which is a feature vector. They are used in text processing and speech recognition because sentences are sequences of words or characters. A RNN is not feed forward because it contains loops: each layer has a state that can be seen as a memory, and a layer receives two inputs: the vector of states from the same layer and the vector of states from the same layer in the previous state.

Given an RNN with two layers the first one will receive the features vector whilst the second will receive the first-one's output. To calculate the state at each time-state at each neuron, first a linear combination of the input feature with the previous state for the same layer is calculated by means of two vectors, let's say w and u, and an extra parameter b. Then an activation function is applied to the linear combination, usually the hyperbolic tangent one. Meanwhile, the output of a layer is calculated for the whole layer at once, which takes a vector and returns another one of the same dimensionalities. This is done by another activation function, usually the Softmax one. The activation function in this case is applied to a linear combination of the state values vector using a parameter matrix W and adding a parameter vector c. All of the seen parameters (b, c, w, u) are calculated by gradient descent and backpropagation algorithm application, whilst W dimensionality is chosen by the analyst such as multiplied by the states vector match c parameters vector dimensionality. Both activation functions suffer from the vanishing gradient problem. Apart from that, this kind of networks have problems with storing the layer states; features from the beginning tend to be forgotten as the sequence grows, this is because they are affected by the newer ones. This implies a potential loss of cause-effect information.

The most effective subtype are the gated RNNs, covering Long-Short Term Memory networks (LSTM) and Gated Recurrent Unit networks (GRU) [41] [130] [209]. These can store the information the way regular memory does, with the particularity that “memory management” is performed by activation functions. A trained neural network can decide to keep information in order to use it later on to process the feature vector. The decisions on whether to store or access the data are made by the neurons, and learned from the data. The access operations are performed by gates, which in their simplest form are the minimum GRU –composed by a memory cell and a forget gate-. It takes the values of the previous states vector (t-1) and a features vector, applying the sigmoid function. If the gate is close to zero then memory will keep the former value, whilst if gate’s value is closer to one the new value is stored. The gated neurons, then, take an input and store it for some time. This operation is equivalent to the identity function, so no vanishing of the gradient will occur as derivative is constant. Other important subtypes are sequence-to-sequence RNNs and bi-directional RNNs [204].

*Sequence-to-sequence learning (seq2seq)***.** [220] The possible applications of these models are application in machine translation, conversational interfaces, text summarization and others[164]. They have two parts: encoders and decoders. Encoders in this case are a neural network of any kind which processes the input to retrieve a numerical (vector) representation for the meaning. The decoder takes this vector called embedding and generates a sequence of outputs, whilst, it updates its state by combining the embedding with the input. Both of them are trained with the same data simultaneously back-propagating errors from the decoder to the encoder. This is a fairly new research domain.

Semi-supervised learning (SSL). In a nutshell, what SSL intends is to get good predictions from partially tagged datasets, to avoid asking the expert to tag the whole dataset. This approach has reached high levels of success by classifying hand-written digits (MNIST) with only ten labelled examples per class, using a ladder neural network. A ladder neural network makes use of encoder-decoder feed forward architectures in what is called an autoencoder[165] [14]. This neural network tries to reconstruct its input in the most possible similar way, having the embedding in between. The reconstruction of the input by the decoder is done by means of using a central embedding layer (see seq2seq paragraph) which has less dimensionality than the input. Hence the name of bottleneck layer as dimensionality is reduced to be increased again later on.

[164] https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346
[165] https://towardsdatascience.com/how-to-make-an-autoencoder-2f2d99cd5103

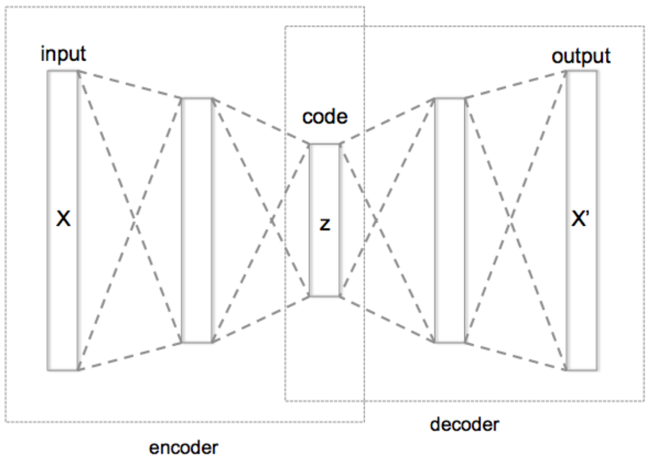

Figure 16: Autoencoder (Ladder Neural Network)

Tools. Normally adaptations to existing algorithm implementations and later combinations are required in AI. Probably the most popular languages among data scientists are R and Python. R is thought to be used inside its own IDE (RStudio) [166] and designed as a tool for statistics crew. Python, in turn, is a general-purpose language, clear, simple and expressive, which covers a wide spectrum of fields, having packages for physics, simulation, biology, etc. The trend in AI is to use Python, which provides neat and readable code, and has multiple well-documented libraries. Normally, when working in the field of AI with Python, a web IDE is used, which is called Jupyter. This IDE has a notebook interface which consists in a series of cells for independent code execution, controlling the data flow.

When it comes to training multi-layer NNs, it is good to take into consideration tools for automatic differentiation such as TensorFlow or Theano. Essentially, these tools allow us to define an architecture and an error function. Afterwards they apply the backpropagation algorithm to optimize the weights of the model by minimizing the error over a training dataset. Thus, changing the number of layers or even using new types of neuron can become trivial so the tool will calculate a new function to optimize the parameters and to minimize the error. In fact, they can optimize the models for specialized hardware such as GPUs.

There are several levels of abstraction, and go from hardware components to graphic interfaces:

- *Level 0*. Hardware elements for faster execution and training of NNs, especially GPUs
- *Level 1*. Drivers and libraries which take advantage of distributed computation capabilities of hardware elements
- *Level 2*. Automatic differentiation tools, which allow to define models and optimize parameters in an automatic way
- *Level 3*. Libraries with implementations of NNs. These are based on automatic differentiation to provide out-of-the-box implementations of the different NN types.
- *Level 4*. Graphical tools that apply NNs, in order to make it easier for non-technical users to apply them to real world problems

[166] https://rstudio.com

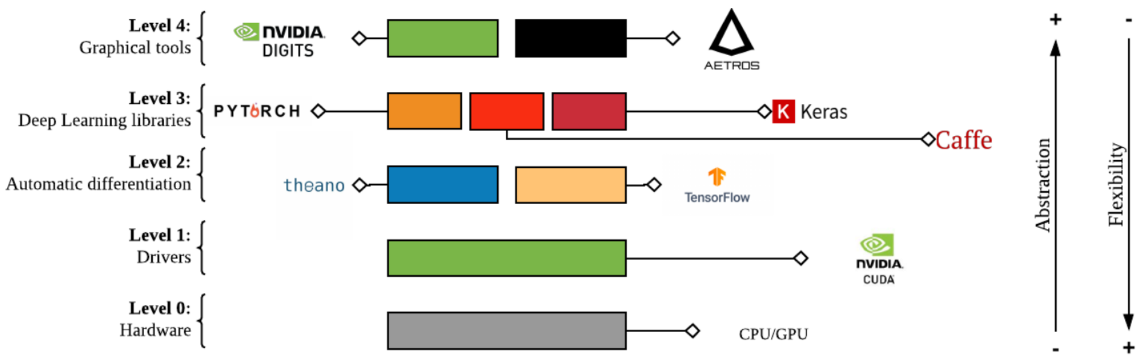

Figure 17: Neural network implementation tools abstraction levels

Level 1

- **NVIDIA CUDA**[167] is a toolkit for leveraging GPUs potential to create applications which transparently make use of the parallel processing capabilities –targeting, but not limited to, NVIDIA GPUs-. CUDA programming language is C/C++. This framework [34] [166] provides abstractions to solve parallel computation problems in a scalable way: several applications can be found such as astrophysics simulation or molecular dynamics.

Level 2

- **TensorFlow** [1] is an interface for expressing Machine Learning Algorithms and an implementation for executing such algorithms, with the capability of doing so in a wide variety of systems. It has been used in computer science areas suck like speech recognition, robotics, NLP, geographic information extraction and computational drug discovery. The programming model is based on flows represented by graphs, and using C++ or Python as frontend languages. It has a release with an Apache 2.0 License and a wide set of maintainers in Google's Machine Learning community, apart from the Google Brain team. TensorFlow offers several levels of abstraction, being able to interact as a backend with Keras in order to keep neural networks programming at an accessible level. It is focused on distributed multi-node computation.
- **Theano** [229] is a Python library that defines mathematical expressions and compiles them in a transparent way, after storing them as a graph. Its API works the same way as NumPy for performing computations over n-dimensional arrays, which allow users to easily switch to Theano using a familiar syntax, but generating high-performance code for CPUs and GPUs. It can be extended with Python, C++ or CUDA. It is Open Source software with a BSD license, and relies on the GitHub community for its development and maintenance. As for TensorFlow, some other software can be used to ease its use with Deep Learning and Machine Learning paradigms: Keras, Lasagne or Pylearn2. A comparison amongst Deep Learning engines can be found at [123].

Level 3

- **PyTorch** [109] [179] provides an imperative Deep Learning framework based on Torch, a mature Machine Learning Library implemented in C. PyTorch allows to evade Lua and write Python code to implement neural networks and its tools. It is, consistent with other scientific computing libraries –enables bidirectional exchange of data- and supports hardware acceleration (GPUs). It is focused

[167] https://docs.nvidia.com/cuda/

on both ease of use and performance. PyTorch was primarily developed by Facebook's AI Research lab (FAIR), being free and released under the Modified BSD License.

- **Caffe** [99] is a Deep Learning framework for training and deploying general purpose convolutional neural networks aside of other models on commodity architectures. It allows CUDA GPU computation and separates model implementation from its representation, enabling switching between platforms. Caffe is C++ at its core having bindings with Python and Matlab, and has been adopted for speech recognition, robotics, neuroscience and astronomy. Berkeley Vision and Learning Center along with GitHub community are in charge of its maintenance. Finally, it includes Deep Learning algorithms [213] and a collection of reference models.
- **Keras**[168] is a high-level abstraction neural network API, which permits multiple backends such as TensorFlow, Theano or CNTK to support MLPs, CNNs and RNNs, and has implementations of elements such as layers, pre-processing tools or activation functions for writing deep neural network code. It is written in Python and its purpose is to enable fast experimentation [110], taking as principles user friendliness, modularity, extendibility and Python as a development tool. It is provided with MIT License and François Chollet is his main author and maintainer.

A high-level comparison amongst the formerly described frameworks can be found at [64], with some other alternatives to the technologies seen so far.

Level 4

Several level 4 tools can be found which try to reduce the steepness of the learning curve for Deep Learning intricacies. An interesting study puts them all together [221] to evaluate their effectiveness in a no-code environment.

- **NVIDIA DIGITS**[169] is a set of tools for engineers and data scientists which eases all Deep Learning tasks such as managing data, designing and training neural networks with GPU hardware and model visualization. It allows to train models interactively and select the best model –even pre-trained ones- in an easy way, letting data scientists avoid the difficulties associated with programming.
- **AETROS/Deepkit**[170]. Former AETROS[171] –now Deepkit- provides a graphical interface to design and train Machine Learning models supporting Keras, TensorFlow and PyTorch, and giving a plethora of tools such as metrics, parameter visualization, model debugging, Git integration and cluster management to ease data analysis by means of AI.

5.2 Service-Oriented Architectures and Artificial Intelligence

5.2.1 Code Plugins and Code Generation

Some coding paradigms have to be presented prior to address code generation. By specifying different levels of functionality requirements, those paradigms can help enable the automation of code generation and continuous integration of the software life cycle.

Test Driven Development (TDD) [141] focuses on code generation with tests as a starting point of agile methodologies and quality code generation, avoiding repetitive tasks. Tries to help developers leverage

[168] https://keras.io/
[169] https://developer.nvidia.com/digits
[170] https://deepkit.ai/
[171] https://www.producthunt.com/posts/aetros

productivity by dividing code generation into smaller tasks, in a repeated cycle that increments itself (write tests, write code, refactor), and isolating it from implementation details. The underlying idea is to turn code generation upside down, making problem solving more concise and obtaining documented and tested code as a result.

Behaviour Driven Development (BDD) is an evolution of TDD, hence an agile technique, where QA and non-technical/business participants can get involved in the definition of the functional behaviour by using a Domain Specific Language (DSL). At the same time, developers can focus on the implementation rather than domain details. Thus, translation between domain language and technical language is kept to a minimum. This approach allows for a better alignment with project purposes where abstractions are key in-service definition. Tools such as Cucumber[172] or Tidy Gherkin[173] can be used to manage the desired project features and their specification. BDD is more user standpoint-oriented, while TDD is for small isolated functionalities. BDD involves a test engineer and a product manager, and possibly other stakeholders, while TDD only requires a developer. A third approach is ATDD[174] (Acceptance Testing Driven Development) which is tightly coupled to BDD. While BDD claims that the specification of the behaviour must be defined first, ATDD claims that Acceptance Tests must be agreed among the development team first. If implemented with a DSL like Gherkin, the ATDD approach would combine both since user stories are initially defined in the "AS A/AN-I WANT TO- SO AS" fashion, and then the user acceptance test is defined in the "GIVEN/WHEN/THEN" fashion.

It is also necessary to talk about programming paradigms. First of all, we have the classic, imperative system in which several instructions are interpreted. Additionally, there is a symbolic paradigm, focused on neural networks, which compiles instructions into a graph for subsequent execution. Even though it provides less flexibility, this paradigm represents an enhancement in terms of memory and speed efficiency. Depending on the development framework used in terms of user's implementation preferences, there are Torch [178], Caffe [99] [179], and some others as abstraction layer for TensorFlow [1]. Finally, we have probabilistic languages as GEN (MIT) for general purposes in terms of robotics, NLP or AI [52]. GEN's aim, as long as other alternatives such as Edward[175] or Pyro[176] is to make probabilistic programming usable in the same way TensorFlow did with Deep Learning.

Software Generation Platforms: Deep Learning development platforms provide more or less complex abstractions over standard algorithm implementations, to both ease access to newcomers and reduce implementation tedium when it comes to experts. According to [125], at higher abstraction level flexibility decreases, and probably a real-world working platform needs a purpose-built item. There is no one-for-all algorithm; developers base their work on a mix amongst experience, best practices and a lot of trial and error. Even if a code generation platform does not provide a final solution, it will probably help with the boilerplate code, making it possible to focus on the task. The platform can act as a cookbook, creating snippets for how-to questions on the basis of previous working and tested code.

There are also, like the aforementioned code completion plugins, very extensive cloud platforms for AI assisted code development[177] [15] which make use of code repositories to find and propose code implementation solutions combining neural and combinatorial techniques. **Deep Coder** is a state-of-the

[172] https://cucumber.io/
[173] https://chrome.google.com/webstore/detail/tidy-gherkin/nobemmencanophcnicjhfhnjiimegjeo?hl=en-GB
[174] https://www.browserstack.com/guide/tdd-vs-bdd-vs-atdd
[175] http://edwardlib.org/
[176] http://pyro.ai/
[177] https://www.deepcode.ai/tech

art platform which implements the code-by-example or inductive programming by learning from several Open Source repositories. It gives recommendations, best practices and security suggestions over provided code, classified by importance and topic. Particularly Deep Coder is interesting because of its Learning Inductive Program Synthesis or LIPS, which gives an idea on neural Network application (RNN) to searches between previously generated code which matches a set of input-outputs; i.e. solve the same problem to provide a generalization. Additionally, **Bayou**[178] is a rather similar alternative, but its results and code quorum analysis are different. Bayou is a DARPA-owned platform whose aim is to generate Java code to interact with APIs from descriptions or “hints”. Input comes from a sketch or minimum program structure where variables are declared and initialized, and a “query” with keywords for the code’s objective or wanted variable types[179]. To create Bayou’s corpus Open Source repositories with program sketches have been considered (claimed 150,000 methods), associating queries with the shape of their solutions via neural networks with a method called “Neural Sketch Learning” [158]. Bayou represents a simpler approach as it is focused on syntax rather than semantics. Both Bayou and Deep Coder agree on the definition of a DSL to narrow down problem resolution.

Narrowing it down to Deep Learning, Blueprint AI is a platform which has been designed to speed up development in the context of a Theano to TensorFlow migration, which enforced to re-learn the AI coding process from scratch. It prosecutes to lower down AI barriers to newcomers. It is based on diagram creation to define Deep Learning implementations, maintaining flexibility by enabling code modifications. The aim is to evade thinking of boilerplate, granting developers the capability to focus on resolving the problem. It leaves modelling to other platforms such as Floydhub, Valohai, RiseML, but at the same time generates PEP8 compliant documented code from a drawing assisted interface[180,181].

Finally, Ludwig [156] is a platform whose purpose is to keep the user away from Deep Learning complexity, allowing them to generate and train models, apart from generating predictions from a simple CSV and a YAML style configuration file, for defining input/output types. All this with no deep knowledge. For expert users it represents a productivity enhancer. It works as a TensorFlow wrapper by facilitating the use of standard algorithm implementations. Ludwig is Open Source and sets a good standard at which to aim. As a general caveat, the following paragraph extracted from an automatic code generating article reveals the necessity of comparing Deep Learning solutions with other approaches:

“The experiment demonstrated that CDS-POOLING is the most effective approach outperforming other complicated models. This also tells us that a simple and effective method should be a prevailing concern. Additionally, it helps researchers to avoid the blind use of Deep Learning algorithms when solving software engineering problems. Considering the nature of this problem, more sophisticated models reveal outstanding results but are excessively computationally expensive, because they need to optimize thousands of parameters, e.g., RNN or CNN. On the contrary, maybe some simpler models can be robust, which only compute the sentence embedding by simply adding or averaging operation over the word embedding, just as our CDS-POOLING. But that also means, such a simple pooling operation does not take word-order information into account. However, pooling operation has the advantage of having significantly fewer parameters, which means it can train much faster and obtain an equally good precision, comparing to RNN or CNN. Thus, there is a trade-off between training speed and efficiency”. This can be a caveat against blindly applying Deep Learning techniques across all the project. Other

[178] http://www.askbayou.com/
[179] https://info.askbayou.com/
[180] https://medium.com/@creaidAI/introducing-the-ai-blueprint-engine-a-code-generator-for-deep-learning-31093499b246
[181] https://blueprints.creaidai.com/

platforms described in this section show that results can be achieved via Deep Learning, but maybe it is not always the best approach [222]. An example can be found online[182]. To this effect –the suitability of Deep Learning with helping purposes- it is worth remarking again the existing trade-offs between flexibility, needed to solve concrete problems, and the ease of use. And conversely, this ease of use does not necessarily mean that there will be no room for improvement at a later stage (via coding or refining algorithms).

Other resources. General purpose NNs, such as ONE, which aim to build a one-for-all RNN [190] [198] do not seem to have a practical implementation[183] despite the features and applications it claims to have. H2O represents another example of broadly used framework for Neural Network problems [119], such as classification problems [153] in different conditions [120] [218], but as many others its purpose is aligned with Big Data and the IoT [64].

Ultimately, the problem amounts to the level of abstraction that will be considered from the perspective of the final user, given the project’s goals. An optimal point seems to be between Blueprint AI and Ludwig, considering the project’s aims. With respect to the code-by-example paradigm, Smart Assistant can take Deep Coder into consideration as a reference for what a baseline can be.

5.2.2 Service Discovery

A major problem in modern service-oriented environments is to implement service-based applications that would automatically perform the search for services which satisfy specific requirements. No literature has been found on service discovery *and* Deep Learning. High level abstractions and ontologies can be used, as stated in the project proposal, to infer the programming context and offer suggestions that are based on the software classification described in the next subsection. Neither platforms nor plugins mentioned in the previous section talk about it, so a further investigation into the topic has to be performed during the project.

Upon a more exhaustive research, classical approaches to service discovery have been identified. Regarding this topic specifically, CERTH experience is key to determining the feasibility and appropriateness of the developments. Examples of context awareness via Machine Learning and agents which use the semantic web can be found along with benchmark approach, but the majority of them are not applied in industry because they are tagged as “academic” [163]. There are some approaches, such as querying for service discovery based on hierarchical clustering via binary trees [49], which provides a performance upgrade on Service Discovery [46]. There is a plethora of documentation on SOA as an architecture pattern of combined services. These services (or loose-coupling functionality units) communicate with each other to share data and coordinate activities, building applications based on independent functionalities with standard interfaces [46] [49], for example, the tools and approaches for automated service composition and service discovery, such as TF-IDF and cosine for matching syntactic information embedded into description languages.

Service description languages such as WSMO, OWL-S, SAWSDL, RDF, etc., try to reduce heterogeneity, however, both interoperability and reusability seem to be a problem. OWL is a language designed by the W3C Web Ontology Working group. Among the OWL-S ontologies, which have been proposed for this project, we can find systems [162] which can compose web services from different

[182] https://www.codeproject.com/Articles/1156694/A-Look-into-the-Future-Source-Code-Generation-by-t
[183] https://medium.com/@creaidAI/lowering-the-entrance-barrier-of-deep-learning-with-code-generation-9a49e62cf165

ontologies to satisfy requests if a particular service cannot be found. There is a difference between discovery and composition of web services, as composition systems usually don't support both, and usually they have to be based on the same ontology. **Discovery** is a process in which an individual advertised Web Service satisfies a Web Service Requirement, while **composition** is a process in which multiple advertised web services are composed to satisfy a request. [162]. For example, the discovery component in MODiCo checks the request requirements to see if they match any advertised Web Service in the repository. If found, no further composition is needed. Otherwise, the composition component is executed to build up a service which matches the requirements. To check the similarities between services inputs, outputs and operations are compared. The one that best fits a threshold will be selected.

We can resort to **SA-REST** services to set a basis of semantic descriptions for web services, adding meta-data REST API descriptions in HTML or XHTML. **WDSL** is a W3C language which allows to create web services descriptions, for which usually combines SOAP and **XML**. WSDL specifies an abstract interface for users through which they can access web services and it gives guidelines for interface use. This language is the basis of Web Services. **UDDI** (Universal Description, Discovery and Integration) is a registry which provides service updating with a WSDL ontological basis, relying on three categories: yellow pages, white pages, green pages. White pages stand for provider information, green pages store technical information about the available services e.g. a WSDL document, and yellow pages allow publishers to associate their services with standard taxonomies. [50]. UDDI is to provide the key to discovery with SOAP communications but the problem is that only keyword-based search is supported. **SOAP**, which once meant Simple Online Access Protocol, is a W3C protocol/format for message transmission (usually via HTTP) among services and using **XML. OWL** is a W3C standard for ontology description in the semantic web, and it is used when the information contained in documents has to be processed by applications instead of humans [149] [164]. It can contain descriptions and properties from classes, constraints on properties and relationships between classes and properties. Similarity can be measured among concepts (classes) from different ontologies [164] via different methods such as n-gram, token matchers, etc. The different aspects of similarity include syntactic similarity, property similarity, context similarity, neighbourhood similarity and equivalent concepts similarity. This is important for the detection of "basic" similarity between services. Context similarity is the similarity between two root concepts of two different ontologies. Machine Learning approaches (GLUE system – [164]) have been performed to find mappings between ontologies, particularly the most similar concepts based on joint probability distribution (Jaccard coefficient) resulting in a poor performance. Other standard semantic approaches such as measuring similarity between the concepts' tags, properties and super-concepts [62], syntax [36] and different selections of features(components) [174][191] are already explored with better results, but, again, from a classical perspective. **OWL-S [**139**]** is an ontology, rooted in **DAML-S** for describing Semantic Web Services and enabling automatic web service discovery, automatic web service invocation and automatic web service composition and interoperation[184]. It has three parts: service profile, service capabilities, and service grounding. The OWL-S service profile shows functional and non-functional information needed for an agent to discover a service, while the OWL-S service model and OWL-S service grounding, taken together, provide enough information for an agent to make use of a service once it has been found. [163]. **WSMO** (Web Service Modelling Ontology) provides a conceptual framework and a formal language for the semantic description of all relevant aspects of Web services.

[184] Some examples related to Amazon Web Services can be found here

Most of the semantic web service approaches work with the concepts above, using S3[185] (Semantic Service Selection) as a benchmark. S3 provided a test-bed for evaluating the performance of the service matchmakers (recall, precision, F1, response time, etc.). S3 contest has not been supported since 2014. The major problem with respect to semantic web vision is that it is not yet accepted on the Internet. A Semantic Web would provide web-based indexing, but the problem is that it requires the rewriting of web resources, so it is not extended on the Internet.

Some approaches have used the syntactical description method for services (WSDL) and some of them investigated semantic service descriptions (WSMO, OWL-S). In hybrid methodologies[186] are introduced that contain both syntactic and semantic approaches, in a recommendation called SAWSDL.

Some other approaches consider service information in text format and then apply some classic Information Retrieval (IR) techniques. IR is a method to retrieve information from documents. They use WSDL and UDDI, but still there is a problem associated with the management of large-scale services. In [50] a new search method for web services called WSQBE is presented. Two-step matching approaches are applied, which reduce the document search space by using vector space to describe web services, and then apply IR techniques. The vector space model is the classic algebraic model for representing text documents. In later work, some academic works extended WSQBE+ by improving IR techniques.

Most of the discussed approaches assume that service requirement and service description are in the same description, but in the real world, there are some environments which contain a different service description. In [68] a service discovery method is proposed that covers different semantic, syntactic and hybrid service description languages. In some works, Degree of Match (DOM) concepts are addressed. [46].

The cited works make use different kind of service descriptions. They categorize the services according to semantic/syntactic IOs and syntactic tags for distance computation. Initiatives that aim to create pools of services have been described[187], however, they seem to no longer be available. One of the problems associated with the Deep Learning approach is the absence of big scale service code repositories that would enable the tagging of services and the eventual training of models, mostly learning from the code itself. Nevertheless, examples of context awareness as an organization structure tool via Deep Learning can be found in [31]. However, it is a rather abstract concept which needs to be explored in combination with the implementation. Other key concepts of service discovery are:

- **Governance**: IT governance (ITG) is defined as the process that ensures the effective and efficient use of IT in enabling an organization to achieve its goals[188]. In other words, “The way in which a company can consolidate the process management initiatives within standards, rules, and guidelines that all go together towards a common goal”.
- **API**: Consist of a minimal stable interface which can be used by other software systems to access or manipulate underlying systems or data.
- **REST** (Representational State Transfer): A software architecture which defines how networked resources are defined and addressed. Resources have a universal identifier, and they are manipulated using a standard interface (HTTP) between network components (client/server).

[185] http://www.dfki.de/~klusch/s3/index.html
[186] https://www.w3.org/TR/sawsdl/
[187] www.service-finder.eu
[188] https://www.gartner.com/en/information-technology/glossary/it-governance

5.2.3 Software Classification

As stated above, Bayou and DeepCoder platforms make use of existent repositories to analyse code and extract abstractions. For instance, Bayou gets its “hints” by taking advantage of Java’s highly structured and specific code, to get keywords from methods’ camel case name [51] [158] [188] [234] from decompiled applications and later translating them into AML language. In this case it is necessary to perform a classification for code reuse in order to make Service Composition easier in terms of automation. Automatic code and artifact classification and indexation [75] are generally based on lexical, syntactic and semantic information taken from software description. Its aim is to leverage code reuse. According to academic results, semantic techniques are more powerful than traditional keyword search systems, but the problem is that they require a generated knowledge structure for each application domain. This topic has been studied extensively over the years [12], it has even been considered in Machine Learning, where unsupervised incremental algorithms are run to extract hierarchies and cluster the assets in code repositories. A basic classification of retrieval methods is given in [13]:

- Classification schemes
- Component storage and retrieval method
- SOM (Self-Organizing Map) and GSOM (Growing Hierarchical SOM) Techniques
- Structural and behavioural techniques
- Hypertext technique
- Browsing technique

In most of the Machine Learning solutions, a good dataset is key to the design of a highly accurate model. Datasets are improved by increasing the available code with Open Source community contributions and resources (such as GitHub) are a well-known platform for the generation of datasets. GitHub provides an API to retrieve the desired data. In [136] various types of software artefacts were used for classification purposes, by means of a code crawling application known as GHTorrent. The gathered information contained items such as user comments, pull requests, etc.

Some other works use code comments to classify software [177] with the aim of understanding their purpose. Code comments have valuable information about the implementation.

There are several malicious code classifiers based on clustering [25] [131] (nearest neighbour, n-gram patterns) [205] and even on Deep Learning, using Convolutional Neural Networks [51] [188]. As for automatic classification for large Open Source repositories (its main purpose is to avoid manual tagging), there are systems with no publicly available implementation (MUDABlue) which use statistic methods over the code like Latent Semantic Analysis [105] [225], based on extracting the contextual meaning of the words from statistical computations over a large corpus of text. More evolved solutions use Latent Dirichlet Allocation [225], which analyses source code in search of topics (code is a mixture of topics) according to manual or automatic categories. There are some other approaches more oriented towards code reuse than software system classification. A DNN plus TF-IDF approach for Java projects can be found in [165].

5.2.4 Context Awareness

Context Sensitivity is a concept propagated in various domains, such as ubiquitous computing and AI. It is the idea that computers can be both sensitive and reactive, based on their environment. As context integrates different knowledge sources and binds knowledge to the user to guarantee that the understanding is consistent, context modelling is extensively investigated within Knowledge Management (KM) research [215].

It is difficult to find a single definition for the notion of context, but its importance in communication, categorization, intelligent information retrieval and knowledge representation has been recognized for many years. In the AI domain, the concept of context is usually defined as the generalization of a collection of assumptions [32] [208]. A common, pragmatic definition for context-aware applications, defines context as "any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves" [55] [56]. The current research on knowledge context is primarily oriented towards capturing and utilization of contextual data for actionable knowledge [4] [224]. A number of systems to handle context awareness were proposed by the research community [22] [114] [42]. Thereby an important aspect is to make the data available for contextual analysis and processing. Various solutions for monitoring and ingesting data into contextual analytics services are available, e.g. U-QASAR [122], SAFIRE Context Monitoring Framework [202] or FIware Context Broker[189].

The key elements of the context sensitivity solutions are context models (that describe the situation to which the ICT environment has to adapt), context extractors which identify often in real time the current context to which the ICT environment has to adapt and context adapter which adapts the behaviour/outputs of the ICT environment to the identified (extracted) current context. The basis for context-aware applications is a well-designed Context Model. As context integrates different data and knowledge sources and binds knowledge to the user to guarantee that the understanding is consistent, context modelling is extensively investigated. Different kinds of contexts should be represented in a common "language" when possible, but the representation must be extensible enough to support domain and application specific concepts. Typical context modelling techniques include key-value models, object-oriented models, and ontological methods [216]. A semantic model (or ontology) provides a representation flexible enough to support common modelling of context in a structured way, as well as domain specific extension to the model, thus it is chosen for representation purposes. Ontology based modelling is considered the most promising approach, as it enables a formal analysis of the domain knowledge, promoting contextual knowledge sharing and reuse in an ubiquitous computing system, and context reasoning based on semantic web technologies [76] [175]. Up to now there were only limited "industrial driven" attempts to provide harmonised modelling of context under which data from software development are generated. The problem to be solved is how to extract context from the development of software service as well as the use of software services. Since it is planned to model context with ontology, context extraction mainly is issue of context reasoning and context provisioning: how to inference high level context information from low level raw context data [200] and [217]. Based on the formal description of context information, context can be processed with contextual reasoning mechanisms [70] [135]. There are three main categories of reasoning, Deductive reasoning, Event-Condition-Action reasoning and Statistical reasoning which can be distinguished.

Context sensitivity is of special importance in the ICT systems in software industry. The most challenging aspect of the application of the context sensitivity in the software industry is an effective acquisition/collection of data needed to extract a current context. Therefore, the advanced collection of data, is a basic prerequisite for an effective application of this approach in manufacturing industry. The

[189] https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE.OpenSpecification.Data.ContextBroker

key challenge is to identify costs effective ways to obtain data needed for context extraction, i.e. to find cost effective data sources. Context Awareness is especially useful for software products or Product Extension Services (PES) used under dynamically changing conditions and by various users [201]. An approach for context awareness in context sensitive embedded services was developed within Self-Learning project [217] and further developed in the SAFIRE project for process planning and optimisation [60]. Within the ProSEco project for context sensitivity services were adopted to PES [202].

5.3 Machine Learning in Software Quality Assessment

Artificial Intelligence and Machine Learning techniques, beyond their use for service creation, identification and orchestration, can also be leveraged to assess the quality of software, including software behind the provision of services and software consuming services.

Software quality is a multidisciplinary topic, in the sense that quality is about: (a) how well software meets users' needs, (b) how well software conforms to its specifications from the developers' point of view, (c) how well inherent, structural characteristics of the software are achieved from the product point of view, and (d) how much the end-user is willing to pay for it from the value point of view [116]. A significant portion of software quality research is nowadays performed through qualitative empirical studies. However, inherently qualitative studies are subject to bias, in the sense that they heavily rely on expert judgement. To alleviate such subjectivity, in traditional software quality research, researchers are nowadays exploiting the large amount of data that are available through software repositories. Such data enabled researchers to perform large-scale quantitative studies, and adopt modern techniques, such as Machine Learning to effectively carry out a specific task without relying on explicit instructions or rules. For example, Machine Learning can be applied for solving problems related to cost estimation, fault and vulnerability prediction, etc. Based on the aforementioned applicability of ML technologies, we believe that there is an opportunity to apply ML in quality management.

The goal of this section is to investigate how Machine Learning can be applied for software quality management, by studying existing literature. Since, the State-of-the-Art lacks a substantial amount of studies, for limiting this study in this research direction, we conducted a broader secondary study, i.e., on how Machine Learning approaches have been used in software engineering (SE) problems, by conducting a systematic literature review (SLR). Thus, the main outcome of this section is the provision of:

c1: The current status of research on combining ML and software engineering. In particular, we investigate which software engineering problems are approached through ML technologies.

c2: The opportunities of applying ML in software quality assessment. To achieve this goal, we map software engineering practices, in which ML has already been applied, to software quality management activities and concepts.

c3: The challenges for the adoption of ML in TDM research.

Therefore, the goal of this section can be described as follows: "*Analyze existing software engineering literature for the purpose of understanding the application of Machine Learning technologies for solving software engineering problems, with respect to: (a) the targeted software engineering problems; (b) the proposed Machine Learning solutions; and (c) the mapping between them*". To systematically explore the aforementioned goal, our study is built around three research questions:

RQ$_1$: Which software engineering problems are approached with Machine Learning technologies?

RQ$_2$: Which Machine Learning technologies have been used for approaching software engineering?

RQ$_3$: What is the mapping between software engineering problems and Machine Learning solutions?

Software engineering is a mature science field, which, however, strives for new solutions to its well-known problems. With the rise of Artificial Intelligence and the increment of the volume of data produced during software development, many researchers have tried to investigate how Artificial Intelligence (specifically Machine Learning) can aid in improving analysis and predictions problems. On the one hand, RQ_1 tries to catalogue the software engineering problems that are approached through Machine Learning, placing special emphasis on the practices that are attempted to be improved and the targeted quality attributes (QA) of interest. On the other hand, RQ_2 investigates Machine Learning technologies that aim at satisfactorily solving software engineering problems, compared to more traditional approaches. Special emphasis is placed on Machine Learning algorithms, learning styles, challenges, and success indicators. Finally, RQ_3 attempts to synthesize the findings of the previous research questions with the goal of mapping solutions to problems.

The search procedure aimed at the identification of candidate primary studies. The search plan involved automated search into five well-known and top-quality publication venues. In particular, we searched the articles identified in Information and Software Technology, IEEE Transactions on Software Engineering, ACM Transactions on Soft-ware Engineering and Methodology, Journal of Systems and Software, and Empirical Software Engineering. These journals were selected as top-quality venues in software engineering, based on their received citations and ranking. The publication venue selection was based on Karanatsiou et al. [103], and is acknowledged as well-known practice while conducting secondary studies in software engineering domain [117], for guaranteeing the quality and relevance of primary studies [8].

Since all publication venues are strictly on the software engineering field, the search string needed to be focused only on ML technologies. As keywords for the search string we have chosen to use simple and generic terms, which may yield as many meaningful results as possible without any bias or preference to a certain Machine Learning method or technique. The search string has been applied to the abstract and title of the manuscripts of all selected venues, without any time constraints. The search has been conducted automatically through the digital libraries of each venue. The final search string was:

"Machine Learning" OR "supervised learning" OR "unsupervised learning" OR "semi-supervised learning"

The papers that were selected as candidate primary studies in the re-view should be relevant to applications of Machine Learning in software engineering. In line with Dybå and Dingsøyr [59], an important element of the systematic mapping planning is to define the Inclusion Criteria (IC) and Exclusion Criteria (EC). A primary study is included if it satisfies one or more ICs, and it is excluded if it satisfies one or more ECs. The inclusion criteria of our systematic mapping are:

- The study applies one or more ML technologies to a SE problem.
- The study defines one or more ways to evaluate quality with ML.

The exclusion criteria of our systematic mapping are:

- Study is an editorial, keynote, opinion, tutorial, workshop summary report, poster, or panel.
- Study's full text is not available.
- Study mentions ML only in introduction or related work section.

The identified articles went through these inclusion/exclusion criteria, by taking into account the full text of the articles. Every article has been handled by the three researchers, including conflict management. During the data collection phase, we collected data on a set of variables that describe each primary study. Data collection has also been handled by two researchers. If both researchers assigned the same value to one variable, this value would be assigned to the variable without further discussion. In any other case, a discussion among the authors would result in consensus about the value to be assigned. We have not applied an agreement measure as the number of researchers involved in the review is not significantly large. However, all conflicts have been recorded. For every study, we have extracted the following data:

[V1] Year

[V2] Title

[V3] Publication Venue

[V4] SE practice (e.g., cost estimation, refactoring)

[V5] Targeted QA (business [107] or product qualities [95])

[V6] Learning Styles (i.e., un-, semi-, or supervised)

[V7] ML Algorithm

[V8] Challenges (challenges of applying ML to SE data)

[V9] Evaluation Metrics (for ML)

5.3.1 Software Engineering Applications

In Table 13 we present the frequency of software engineering problems that are approached with ML. Through the analysis, we have identified 9 high-level (HL) software engineering practices. For each HL practice, we present their frequency, and low-level (LL) practices from which they comprise.

Table 13: Software Engineering Practices Approached with ML

HL Practice	Freq.	LL Practices
Defect Management	21	Fault Proneness Prediction and Prioritization, Defect Prediction, Fault Localization
Cost/Effort Estimation	17	Development Cost/Effort Estimation, Software Maintenance Effort Prediction, Maintenance Type Classification
Design-time QAs	14	Change Proneness Prediction, User Interface Design, Software Product and Process Quality Assessment, Code Smells, Patterns and Tactics Detection, API Instability Detection, Refactoring of Test Suites, Refactoring Recommendations
Project Management	12	Bug Report and Change Requests Assignment Recommendations and Prioritization, Classification of Software Bugs, Commit Log Recommendations, Code Review Prioritization, Configuration Management Recommendation, Development Activity Detection, Software Upgrades Recommendation
Security	11	Malware, Malicious Code and Intrusion Classification/Detection, Fault Injection Detection, Software Vulnerabilities Detection

HL Practice	Freq.	LL Practices
Requirements Engineering	9	FR Recommendations, NFR Detection, Requirements Prioritization, Requirements Assessment, Software SPL Configurations Detection, Application Domain Classification
Run-time QAs	3	Performance Prediction, Energy Efficiency Recommendations
Reuse	2	API Usage Recommendation, Code Examples Prioritization for Reuse
Program Comprehension	2	Trace Recovery, Reverse Engineering

In Table 14, we provide an overview of the QAs that are targeted in each application of ML technologies. From the obtained results we can observe that: (a) maintainability and its sub-characteristics (namely: testability, reusability, modifiability and analysability) are a common target for ML technologies; and (b) business quality attributes are also targeted by ML [108].

Table 14: Targeted Quality Attributes by ML

HL QA	Freq.	LL QA
Maintainability	29	Testability, Reusability, Modifiability, Analysability
Functional Suitability	24	Functional Correctness
Security	12	-
Business Goals	10	Improve Market Position, Reduce Cost of Development
Performance Efficiency	5	Resource Utilization
Usability	1	-
Reliability	1	-

5.3.2 Machine Learning Technologies

To solve the aforementioned problems a variety of ML algorithms and learning styles have been used. The dominant learning style is supervised learning algorithms (89%), followed by unsupervised (6%) and semi-supervised learning (5%). Regarding specific algorithms, in Table 15, we present the most frequently used algorithms (i.e., used in more than 10 studies). Apart from the algorithm name and the frequency of its appearance, we also provide the HL category in which it can be classified. We note in cases when the authors have not specified a concrete algorithm (e.g., neural networks) the term Generic has been used as the ML algorithm.

Table 15: Machine Learning Algorithms

ML Algorithm	Freq.	HL Category
Bayesian Networks	35	Probabilistic Analysis

ML Algorithm	Freq.	HL Category
ID3, C4.5, CART	33	Decision Trees
SVM	31	Kernel Methods
Neural Networks	18	Biologically-inspired Computation
Random Forest	15	Ensemble Learner
Ripper	14	Rule System
Regression	13	Statistical Analysis
K-Means	13	Clustering
KNN	12	Nearest Neighbour

To evaluate the accuracy of the algorithms we identified: (a) those that aim at evaluating solutions to classification problems—i.e., using metrics such as precision, re-call, f-measure, etc.; and (b) those that aim at capturing the error rate of predictions—e.g., MMRE, pred(0.25), etc.

5.3.3 Mapping of SE Problems to ML Approaches

As a next step, having presented the results originating from each discipline independently; we present a classification schema, in which we map the most common HL software engineering problems to the ML algorithms that have been used for solving them (see Figure 18).

To investigate if a relation between specific ML algorithms and soft-ware engineering problems exists, we have performed a chi-square test. The results of the process suggested that the two variables are associated (alpha < 0.01). Therefore, specific algorithms appear to be more appropriate for specific problems and vice-versa.

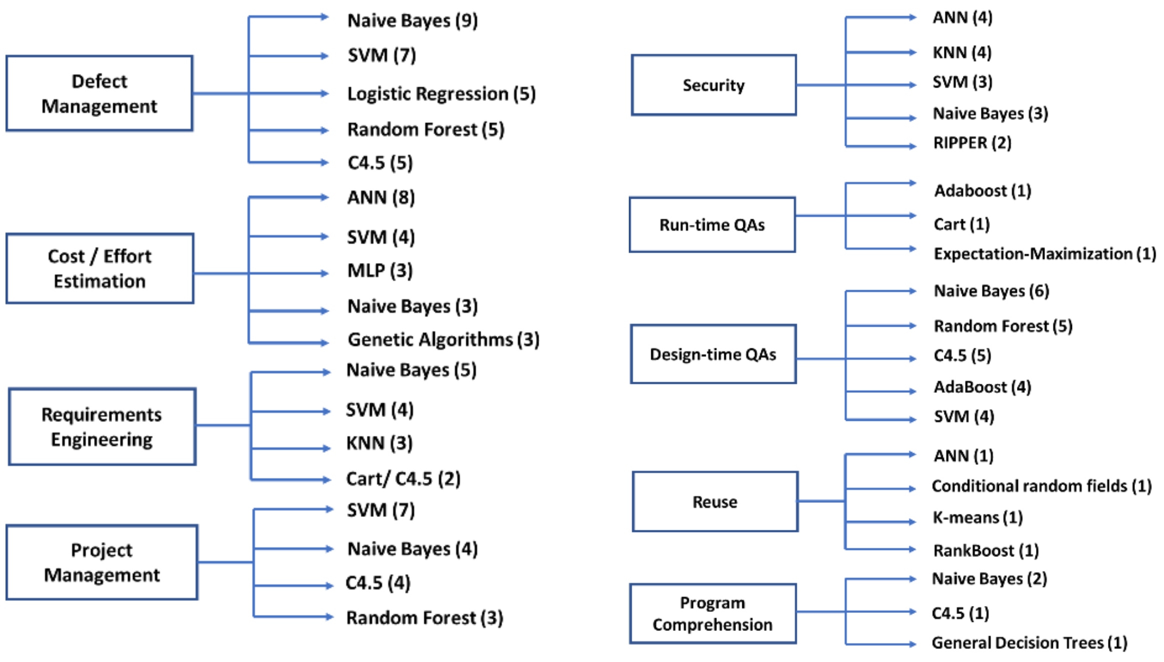

Figure 18: Mapping of ML to Software Engineering Problems

5.3.4 Quality Assessment through Machine Learning

In this section we discuss the main findings of this section, organized based on the expected contributions, i.e., the current status of research, the identified opportunities for the quality assessment community, and the challenges that might exist when applying ML in quality management research.

Current Status. We have observed that Machine Learning technologies have been applied to resolve multiple and quite diverse research problems; however, some of them appear to be prevalent. In particular, we observed that defect management, cost/effort estimation, management of design-time quality attributes, recommendations for efficient project management, and detection of security threats are the most common SE problems that have been investigated. We note that as management we refer to cases that we predict (future state), assess, classify, or detect a phenomenon of interest. In terms of quality attributes, the most relevant ones appeared to be the improvement of maintainability and functional suitability (i.e., correctness), followed by security and business quality attributes. In terms of ML algorithms, we suggest that Bayesian Networks, various Decision Trees, and SVM are the most frequently used ones. Finally, we identified that Neural Network Analysis appears to be fitting for Cost / Effort Estimation problems, Bayesian Networks for Defect and Project Management problems, and Random Forrest algorithms appear to be appropriate for Managing De-sign-Time QAs. On the other hand, Clustering and Decision Trees appear to be equally fitting for various SE problems.

Opportunities. Based on the above results, it is evident that many of the studied problems are related to software quality assessment. In particular, the following practices can be mapped to quality management:

- *Cost/Effort Estimation*: Monetization is a key concept in the quality management. To this end, any cost or effort estimation approach based on past data can be considered as relevant to predict the cost of applying refactoring or to predict the cost of future maintenance effort. In this category of SE problems special emphasis shall be placed to studies that deal with software maintenance effort prediction (e.g., [137]).
- *Management of Design-Time QAs and Defects*: In this high-level category, various related problems have been identified. First, many studies focused on change- [106] and fault-proneness [243]. These concepts are closely related to interest probability, in the sense that changes and faults lead to maintenance activities that can accumulate interest. Additionally, other studies focus on the detection of smell occurrences [69]. Finally, any method that is used for assessing or characterizing the levels of QAs (e.g., maintainability [82]) can be useful.
- *Requirements and Project Management Recommendations*: Many studies use ML to provide recommendations to developers related to which requirements shall be implemented first [182], or which reported bugs shall be prioritized [226]. Such recommendations could be useful for TD prioritization, by considering that artefacts that are not expected to change (due to bug fixing, or implementation of new requirements), shall not be prioritized for design-time quality improvements.

Similarly, by considering the targeted quality attributes, we can also identify some connection to design-time quality management. In many studies ML approaches are used to apply practices that aid in terms of the improvement of the market position of the product, or to reduce the development costs (e.g., by shrinking product time-to-market). In general, the satisfaction of business goals is roots of weakening internal product quality. Additionally, the improvement of the market position of a product can be considered as a by-product of quality management.

Challenges in Applying ML to Software Quality Management. As part of the analysis, we have identified specific challenges in applying ML to SE problems. Among the most important ones we acknowledge the following. First, there is a need for a substantial pre-processing in the used datasets, so as to eliminate cases of imbalanced datasets, handling duplicate values, etc. Additionally, specifically in quality management it is expected to face many difficulties in creating a solid dataset, since the methods for quantifying quality are highly diverse and no state-of-practice techniques exist. Furthermore, for supervised learning algorithms labelling of training data (e.g., software modules) can be challenging as no universal approach for measuring quality exists. In contrast to other fields (e.g., cost estimation) there is a lack of benchmarks that can be used for training and testing of algorithms (e.g., COCOMO or ISBSG). Furthermore, a common challenge in applying ML in software engineering is the curse of dimensionality, in which the researcher shall limit the variables that shall be fed into the model. This challenge is also highly relevant to quality management, in the sense that quality is a multi-dimensional concept, whose assessment requires the consideration of multiple aspects (e.g., code smell, improper architectural decisions, etc.) but also people’s habits and employed processes. Therefore, since the application of ML approaches requires a small subset of input variables to obtain a time-efficient, accurate, and noiseless model, it is of paramount importance to effectively perform data reduction.

6 Conclusions – Market Requirements

Given the above, we can conclude to the following baseline market requirements for the development of the SmartCLIDE solution:

SmartCLIDE **shall** support

1. user-friendly GUI even for non-technical users
2. visually intuitive interfaces to help users with model generation and training
3. implementation of coding-by-example principle
4. the provision of abstractions to minimize manual intervention that are required by the developers to the source code for implementing new features
5. the classification of services, learning from code or applying Machine Learning algorithms
6. user stories, features specification
7. specification of acceptance criteria for functional and non-functional requirements
8. the short iterations concept
9. CI/CD
10. automated testing in different flavours: ATDD / BDD / TDD.
11. static analysis
12. working code as a source of documentation
13. integration with run-time monitoring tools
14. version control of software
15. cloud native IDE for cloud native solutions.
16. Business Process Modelling capabilities
17. service discovery and search
18. service integration through the online dashboard
19. a wrapper which isolates user from DL complexity as far as possible, releasing developers from boilerplate code generation
20. the provision of coding blueprints which can serve as a base for more complex tasks, making code more reusable and easier to understand

SmartCLIDE **should** support

21. refactoring
22. easy configuration
23. the provision of metrics for maintainability / reusability at the service and the system level
24. the extension of existing tools for measuring maintainability and reusability to capture the metrics at the service level
25. the provision of solutions for facilitating the identification and elimination of critical vulnerabilities that reside in the source code of microservices from the early stages of their development
26. the provision of an easy non-coding implementation for Deep Learning usage (general problems) depending on input data
27. the provision of code blueprints (skeletons) based on Gherkin inputs for services implementation
28. the discovery and composition of basic services based on ontologies
29. scalability of processing capability
30. replicability of architecture to increase flexibility
31. fault tolerance and reliable
32. security through isolation / dependability
33. the monitoring of maintainability and reusability of the project under development
34. dynamic software configuration

SmartCLIDE **may** support

35. generation of automatic tests by natural language interpretation of acceptance tests
36. the provision of on-the-fly suggestions on how to improve the reusability and maintainability of the system

SmartCLIDE **may** support

37. agile tools such as a Kanban board
38. implementation of artefacts for Product and Sprint Backlog management (e.g. Kanban or Scrumban boards)
39. implementation of artefacts facilitating waterfall life cycles

References

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems", 2016

[2] Aceto, G., Botta, A., De Donato, W. and Pescapè, A. "Cloud monitoring: A survey", Computer Networks, 57 (9), pp. 2093-2115, June 2013.

[3] Agarap, A. F. "Deep Learning using Rectified Linear Units (ReLU)", Neural and Evolutionary Computing, (revised February 2019).

[4] Ahn J. H., Lee J. H., Cho K, and Park J. S., "Utilizing knowledge context in virtual collaborative work," Decision Support Systems, no. 39, pp. 563-582, 2005.

[5] Alahmari, S., Zaluska, E. and De Roure, D. C. "A Metrics Framework for Evaluating SOA Service Granularity", International Conference on Services Computing, Washington, DC, 2011, pp. 512-519, 2011.

[6] Ali, A. Q., Sultan, A. B. M., Ghani, A. A. A. and Zulzalil, H. "A Systematic Mapping Study on the Customization Solutions of Software as a Service Applications," IEEE Access, vol. 7, pp. 88196-88217, 2019.

[7] Alshuqayran, N., Ali, N. and Evans, R. "A Systematic Mapping Study in Microservice Architecture," 9th International Conference on Service-Oriented Computing and Applications (SOCA'16), Macau, 2016.

[8] Ampatzoglou, A., Bibi, S., Avgeriou, P., Verbeek, M. and Chatzigeorgiou, A. "Identifying, categorizing and mitigating threats to validity in software engineering secondary studies", Information and Software Technology, 106(2), pp. 201-230, 2019.

[9] Appleby, D. and VandeKopple, J.J. "Programming Languages: Paradigm and Practice", 2nd Edition, The McGraw-Hill Companies, Inc., New York, 1997.

[10] Arvanitou, E. M., Ampatzoglou, A., Chatzigeorgiou, A., Galster, M. and Avgeriou, P. "A Mapping Study on Design-Time Quality Attributes and Metrics", Journal of Systems and Software, Elsevier, 127 (5), pp. 52-77, May 2017.

[11] Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., Le Traon, Y., Octeau, D. and McDaniel, P. "FlowDroid: Precise Context, Flow, Field, Object-sensitive and Life cycle-aware Taint Analysis for Android Apps," Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and ImplementationJune 2014

[12] Bailin, S., Henderson, S. and Truszkowski, W. "Application of Machine Learning to the organization of institutional software repositories", Telematics and Informatics, Volume 10, Issue 3, pp. 283-299, 1993.

[13] Bakshi, A. and Bawa, S. "A Survey of Search and Retrieval of Components from Software Repositories", International Journal of Engineering Research & Technology (IJERT), Vol. 2 Issue 4, April 2013.

[14] Baldi, P. "Autoencoders, Unsupervised Learning, and Deep Architectures", Workshop on Unsupervised and Transfer Learning (UTLW'11), July 2011.

[15] Balog, M. Gaunt, A. L., Brockschmidt, M., Nowozin, S. and Tarlow, D. "DeepCoder: Learning to Write Programs", 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26 2017.

[16] Bansiya, J. and Davies, C. G. "A hierarchical model for object-oriented design quality assessment", Transactions on Software Engineering, IEEE Computer Society, 28 (1), pp.4-17, 2002.

[17] Bartocci, E., Falcone, T., Francalanza, A. and Reger, G. "Introduction to Runtime Verification", Lectures on Runtime Verification, Springer, pp. 1-33, 2018

[18] Bass, L., Clements, P., and Kazman, R. "Software Architecture in Practice", Addison-Wesley, Boston, USA, 2003.

[19] Beck, K. “Extreme Programming Explained: Embrace Change”, Addison-Wesley; 2nd Edition, 2004.

[20] Beck, K., Grenning, J., Martin, R. C., Beedle, M., Highsmith, J., Mellor, S., van Bennekum, A., Hunt, A., Schwaber, K., Cockburn, A., Jeffries, R., Sutherland, J., Cunningham, W., Kern, J., Thomas, D., Fowler, M., and Marick, B. "Manifesto for Agile Software Development", Agile Alliance, 2001

[21] Belhumeur, P. N., Hespanha, J. P. and Kriegman, D. J. "Eigenfaces vs. Fisherfaces: recognition using class specific linear projection," Transactions on Pattern Analysis and Machine Intelligence, 19 (7), pp. 711-720, July 1997.

[22] Bellavista P., Corradi A., Montanari R., and Toninelli A., "Context-aware semantic discovery for next generation mobile systems," IEEE Comm. Magazine, no. 44, pp. 62-71, 2006.

[23] Bholanath, R. “Analyzing the State of Static Analysis: A Large-Scale Evaluation in Open Source Software,” 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Suita, Japan, 14-18 March 2016.

[24] Bishop, M. "Computer Security: Art and Science", Addison-Wesley Professional, 1 Edition, 21 September 2015

[25] Bist, A. S. "Classification and identification of Malicious codes", Indian Journal of Computer Science and Engineering (IJCSE), 2012

[26] Boehm, B. “A Spiral model of software development and enhancement”, Computer, Volume: 21, Issue: 5, pp. 61-72, 1988.

[27] Boehm, B. “Get ready for agile methods, with care,” Computer (Long. Beach. Calif)., vol. 35, no. 1, pp. 64–69, 2002.

[28] Boehm, B., Brown, J. R., Kaspar, H., Lipow, M., MacLeod, G. J., and Merrit, M. J., “Characteristics of Software Quality”, North-Holland Publishing Company, New York, 1978.

[29] Bogner, J., Wagner, S. and Zimmermann, A. "Automatically Measuring the Maintainability of Service-and Microservice-based Systems – a Literature Review", International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement (IWSM/Mensura ’17), Gothenburg, Sweden

[30] Bosu, A., Carver, J. C., Hafiz, M., Hilley, P. and Janni, D. “Identifying the Characteristics of Vulnerable Code Changes: An Empirical Study,” 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, Hong Kong, November 2014

[31] Boyle, T. and Ravenscroft, A. "Context and Deep Learning design", Computers & Education, Volume 59, Issue 4, 2012

[32] Brézillon P., “Context in problem solving: A survey,” The Knowledge Engineering Review, 1999.

[33] Brooks, F. “No Silver Bullet. Essence and accident in software engineering”, 1986

[34] Buck, I. “GPU computing with NVIDIA CUDA”, Special Interest Group on Computer Graphics and Interactive Techniques Conference (SIGGRAPH '07), San Diego, California, 2007.

[35] Calinon, S. "Learning from demonstration (programming by demonstration)", Encyclopedia of Robotics, pp.1-8, 2018.

[36] Cardoso, J. and Sheth, A. "Semantic E-Workflow Composition", Journal of Intelligent Information Systems volume 21, pp. 191–225, 2003.

[37] Carvalho, M., Demott, J., Ford, R. and Wheeler, D. A. “Heartbleed 101”, IEEE Security Privacy, vol. 12, no. 4, pp. 63–67, 2014.

[38] Chaves, S. A., Uriarte, R. B. and Westphall, C. B. "Toward an Architecture for Monitoring Private Clouds," IEEE Communications Magazine Volume 49, Number 12, pp. 130-137, 2011.

[39] Chess, B. and McGraw, G. “Static analysis for security”, Security and Privacy, IEEE, vol. 2, pp. 76–79, 2004

[40] Choi, S. W. and Kim, S. D. "A Quality Model for Evaluating Reusability of Services in SOA", 10th Conference on E-Commerce Technology and the Fifth IEEE Conference on Enterprise Computing, E-Commerce and E-Services, Washington, DC, 21-24 July 2008.

[41] Chung, J., Gulcehre, C., Cho, K. and Bengio, Y. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling", Neural and Evolutionary Computing, December 2014.

[42] Chang K. and Kim Y., "Design and implementation of middleware and context server for context awareness," High Performance Computing and Communications, Proceedings, LNCS 4208, pp. 487-494, 2006.

[43] Clayman, S., Toffetti, G., Galis, A. and Chapman, C. "Monitoring services in a federated cloud: the reservoir experience," Achieving Federated and Self-Manageable Cloud Infrastructures: Theory and Practice, M. Villari, I. Brandic, and F. Tusa, Eds. IGI Global, 2012.

[44] Coad, P., de Luca, J. and Lefebvre, E. "Java Modeling in Color with UML", Prentice Hall, 1999.

[45] Cockburn, A. "Surviving Object-Oriented Projects", Addison-Wesley Professional, 1 Edition, 1998.

[46] Cong, Z., Fernandez, A., Billhardt, H. and Lujak, M. "Service Discovery Acceleration with Hierarchical Clustering», Information Systems Frontiers · August 2014.

[47] Cortes, C. and Vapnik, V. "Support-vector networks", Machine Learning, 20, pp. 273–297 September 1995.

[48] Cover, T. and Hart, P. "Nearest neighbor pattern classification", Transactions on Information Theory, 13 (1), pp. 21-27, January 1967.

[49] Crasso, M., Zunino, A. and Campo, M. "Combining query-by-example and query expansion for simplifying web service discovery", Information Systems Frontiers volume 13, pp. 407–428, 2011.

[50] Crasso, M., Zunino, A. and Campo, M. "Easy Web Service discovery: A Query-By-Example approach", Science of Computer Programming, 71(2), pp. 144-164, April 2008.

[51] Cui, Z., Xue, F., Cai, X., Cao, Y., Wang, G. and Chen, J "Detection of Malicious Code Variants Based on Deep Learning," in IEEE Transactions on Industrial Informatics, vol. 14, no. 7, pp. 3187-3196, July 2018.

[52] Cusumano-Towner, M. F., Saad, F. A., Lew, A. K. and Mansinghka, V. K. "Gen: A General-Purpose Probabilistic Programming System with Programmable Inference", 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '19), June 22– 26, 2019.

[53] Cypher, A. "Watch What I Do: Programming by Demonstration". The MIT Press, 1993 Debois P., "Agile Infrastructure and Operations: How Infragile are You?", AGILE' 08, Toronto, Canada, September 2008.

[54] Debois P., "Agile Infrastructure and Operations: How Infra-gile are You?", AGILE' 08, Toronto, Canada, September 2008.

[55] Dey A.K., Providing architectural support for building context-aware applications, Georgia Institute of Technology, 2000.

[56] Dey A.K., Abowd G. D., and Salber D., "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-aware Applications," Hum-Comput Interact, vol. 16, no. 2, pp. 97-166, December 2001.

[57] Di Francesco, P., Lago, P. and Malavolta, I. "Architecting with microservices: A systematic mapping study", Journal of Systems and Software, Volume 150, pp. 77-97, 2019.

[58] Doyle, J. W. M. "SAVI: Static-Analysis Vulnerability Indicator," IEEE Security and Privacy, 2012.

[59] Dybå, T. and Dingsøyr, T. "Empirical studies of agile software development: a systematic review", Information and Software Technology, 50(9–10), pp. 833–859, 2008.

[60] Dziurzanski P., Zhao S., Scholze S., Zilverberg A., Krone K., and Soares Indrusiak L., "Process Planning and Scheduling Optimisation with Alternative Recipes," at - Automatisierungstechnik, 05 12 2019.

[61] Edmonds, E. A. "A Process for the Development of Software for Nontechnical Users as an Adaptive System", General Systems, vol. 19, pp. 215–217, 1974.

[62] Ehrig, M. and Sure, Y. "Ontology Mapping – An Integrated Approach", European Semantic Web Symposium (ESWS'04), 2004.

[63] Epstein, J., Matsumoto, S. and McGraw, G. "Software security and SOA: danger, Will Robinson!", IEEE Security and Privacy, vol. 4, no. 1, pp. 80–83, 2006

[64] Erickson, B. J., Korfiatis, P., Akkus, Z., Kline, T. and Philbrick, K. "Toolkits and Libraries for Deep Learning», Journal of Digital Imaging volume 30, pp. 400–405, 2017.

[65] Falcone, Y., Havelund, K. and Reger, G. "A Tutorial on Runtime Verification", 2012.

[66] Felderer, M., Büchler, M., Johns, M., Brucker, A. D., Breu, R. and Pretschner, A. "Security Testing: A Survey," Advances in Computers, vol. 101, pp. 1–51, 2016.

[67] Feuerlicht, G. "Evaluation ofQuality of Design for Document-Centric Software Services," vol. 7759 LNCS, pp. 356-367,2013.

[68] Fernández, A., Cong, Z. and Balta, A. "Bridging the Gap Between Service Description Models in Service Matchmaking", Multiagent and Grid Systems, January 2012.

[69] Fontana, F. A., Mantyla, M. V., Zanoni, M. and Marino, A. "Comparing and experimenting Machine Learning techniques for code smell detection", Empirical Software Engineering, 21(3), 1143-1191, June 2016

[70] Forstadius J., Lassila O., and Seppänen, "RDF-Based Model for Context-Aware Reasoning in Rich Service Environment," in Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops, 2005.

[71] Fowler, M. "Microservices - new architecture", Online, 2014.

[72] Garousi, V., Felderer, M. and Mäntylä, M. "The need for multivocal literature reviews in software engineering: complementing systematic literature reviews with grey literature", 20th International Conference on Evaluation and Assessment in Software Engineering (EASE'16), Jun 2016

[73] Gegick, M., Williams, L., Osborne, J. and Vouk, M. "Prioritizing Software Security Fortification through Code-Level Metrics," 4th ACM workshop on Quality of protection (QOP'08), pp. 31–38, October 2008.

[74] Gene Kim, Humble, J., Debois, P. and Willis, J. "The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations", IT Revolution, 2016.

[75] Girardi, M. R. and Ibrahim, B. "Automatic indexing of software artifacts," Proceedings of 1994 3rd International Conference on Software Reuse, Rio de Janeiro, Brazil, 1994.

[76] Glassey R. et. al., "Towards a middleware for generalised context management.," First International Workshop on Middleware for Pervasive and Ad Hoc Computing, 2003.

[77] Grand, M. "Microservices Security Risks And Countermeasures," HCL

[78] Green, M. and Smith, M. "Developers are Not the Enemy!: The Need for Usable Security APIs", IEEE Security & Privacy, vol. 14, no. 5, pp. 40–46, September-October 2016

[79] Gulwani, S. and Jain, P., "Programming by Examples: PL meets ML". Microsoft Corporation, Redmond, USA 2017.

[80] Heckman, S. and Williams, L. "A model building process for identifying actionable static analysis alerts," 2nd International Conference on Software Testing, Verification, and Validation, ICST 2009, Denver, CO, USA, 1-4 April 2009.

[81] Heckman, S. and Williams, L. "A systematic literature review of actionable alert identification techniques for automated static code analysis," Information Software Technology, vol. 53, no. 4, pp. 363–387, 2011.

[82] Herbold, S., Grabowski, J. and Waack, S. "Calculation and optimization of thresholds for sets of software metrics", Empirical Software Engineering, 16 (6), pp. 812-841, December 2011

[83] Heun, V., Hobin, J. and Maes, P. "Reality Editor: Programming Smarter Objects",13th Conference on Pervasive and Ubiquitous Computing Adjunct Publication (UbiComp '13), Zurich, Switzerland, September 2013.

[84] Hinton, G. E. and Salakhutdinov, R. R. “Reducing the Dimensionality of Data with Neural Networks”, Science, 313 (5786), pp. 504-507, 28 July 2006.

[85] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. R. “Improving neural networks by preventing co-adaptation of feature detectors”, Neural and Evolutionary Computing, July 2012.

[86] Hirzalla, M., Cleland-Huang, J. and Arsanjani, A. "A Metrics Suite for Evaluating Flexibility and Complexity in Service Oriented Architectures”, Workshops Service-Oriented Computing (ICSOC'08), International Conference on Service-Oriented Computing, Springer, 2008

[87] Hofmeister, H. and Wirtz, G. "Supporting Service-Oriented Design with Metrics", 12th International IEEE Enterprise Distributed Object Computing Conference, Munich, Germany, 15-19 September 2008.

[88] Holzmann, G. J. “The Value of Doubt”, IEEE Software, vol. 34, no. 1, pp. 106–109, 2017.

[89] Hovemeyer, D. and Pugh, W. “Finding bugs is easy,” ACM SIGPLAN Not., vol. 39, no. 12, p. 92, 2004.

[90] Howard, M. "Writing secure code", Redmond, Wash: Microsoft Press, 2003

[91] Howard, M. and Lipner, S. "The Security Development Life cycle: SDL: A Process for Developing Demonstrably More Secure Software", Microsoft Press, 2006.

[92] Howard, M., LeBlanc, D. and Viega, J. "24 Deadly Sins of Software Security", 2010.

[93] Hsu, C. W. and Lin, C. J. “A comparison of methods for multiclass support vector machines”, Transactions on Neural Networks and Learning Systems, 13(2), pp. 415-25, 2002.

[94] Hutapea, R. C. A., Wahyudi, A. P. and Suhardi, "Design Quality Measurement for Service Oriented Software on Service Computing System: a Systematic Literature Review," International Conference on Information Technology Systems and Innovation (ICITSI), Bandung - Padang, Indonesia, 2018.

[95] ISO/IEC 25010:2011, Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models, Geneva, Switzerland, 2011.

[96] ISO/IEC 9126-1:2001, Software engineering - Product quality (Part 1: Quality model), Geneva, Switzerland, 2001

[97] Jacobson, I., Christerson, M., Jonsson, P. and Overgaard, G. “Object-Oriented Software Engineering: A Use Case Driven Approach”, Addison-Wesley Professional, 1 Edition, 1992.

[98] Jander, K., Braubach, L. and Pokahr, A. “Defense-in-depth and Role Authentication for Microservice Systems”, Procedia Computer Science, vol. 130, pp. 456–463, 2018.

[99] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S. and Darrell, T. "Caffe: Convolutional Architecture for Fast Feature Embedding", 22nd ACM international conference on Multimedia, November 2014

[100] Jimenez, M., Papadakis0 M., and Le Traon, Y. “Vulnerability Prediction Models: A case study on the Linux Kernel,” 16th International Working Conference on Source Code Analysis and Manipulation (SCAM), 2016, pp. 1–10.

[101] Johnson, B., Song, Y., Murphy-Hill, E. and Bowdidge, R. “Why don’t software developers use static analysis tools to find bugs?” International Conference on Software Engineering (ICSE'13), May 2013.

[102] Jovanovic, N., Kruegel, C. and Kirda, E. “Pixy: a static analysis tool for detecting Web application vulnerabilities,”Symposium on Security and Privacy (S&P’06), Berkeley/Oakland, CA, USA, 21-24 May 2006.

[103] Karanatsiou, D., Li, Y. Arvanitou, E. M., Misirlis, N. and Wong, W. E. “A bibliometric assessment of software engineering scholars and institutions (2010–2017)”, Journal of Systems and Software, 147 (1), pp. 246–261, 2019.

[104] Katsaros, G., Küandbert, R. and Gallizo, G. “Building a Service-Oriented monitoring framework with REST and nagios”, International Conference on Services Computing, Washington, DC, USA, 4-9 July 2011.

[105] Kawaguchi, S., Garg, P. K., Matsushita, M. and Inoue, K. "MUDABlue: An automatic categorization system for Open Source repositories", Journal of Systems and Software, Volume 79, Issue 7, pp. 939-953, 2006.

[106] Kaur, L. and Mishra, A. “Cognitive complexity as a quantifier of version to version Java-based source code change: An empirical probe”, Information and Software Technology, 102, pp. 31-48, February 2019.

[107] Kazman, R. and Bass, L. “Categorizing Business Goals for Software Architectures”, CMU/SEI-2005-TR-021, 2005.

[108] Kazman, R., Cai, Y., Mo, R., Feng, Q., Xiao, L., Haziyev, S., Fedak, V. and Shapochka, A. "A Case Study in Locating the Architectural Roots of Technical Debt", 37th International Conference on Software Engineering, Florence, pp. 179-188, 16-24 May 2015

[109] Ketkar, N. “Introduction to PyTorch”, Deep Learning with Python, pp. 195-208, October 2017.

[110] Ketkar, N. “Introduction to Keras”, Deep Learning with Python, pp. 97-111, October 2017.

[111] Khoshkbarforoushha, A., Jamshidi, P. and Shams, F. "A Metric for Composite Service Reusability Analysis,” Workshop on Emerging Trends in Software Metrics (WETSoM '10), Cape Town, South Africa, May 2010.

[112] Kim, W., Kim, S.D., Lee, E., and Lee, S. "Adoption issues for cloud computing", 7th International Conference on Advances in Mobile Computing and Multimedia (iiWAS 2009), Kuala Lumpur, December 2009.

[113] Kim, Y. “Convolutional Neural Networks for Sentence Classification”, Empirical Methods in Natural Language Processing (EMNLP’14), Doha, Qatar, October 2014.

[114] Kim S., Suh E., and Yoo K., "A study of context inference for Web based information systems," Electronic Commerce Research and Applications 6, pp. 146-158, 2007.

[115] Kitchenham, B. and Charters, S. "Guidelines for performing systematic literature reviews in software engineering", Technical Report EBSE 2007-001, Keele University and Durham University, 2007

[116] Kitchenham, B. and Pfleeger, S. L. "Software quality: the elusive target", IEEE Software, 13(1), pp. 12 - 21, 1996.

[117] Kitchenham, B., Brereton, P., Budgen, D., Turner, M., Bailey, J. and Linkman, S. "Systematic literature reviews in software engineering: A systematic literature review", Information and Software Technology, Elsevier, 51 (1), pp. 7-15, 2009.

[118] Kitchenham, B., Pickard, L. and Pfleeger, S.L. “Case Studies for Method and Tool Evaluation”, Software Magazine, IEEE Computer Society, 12(4), pp. 52-62, July 1995.

[119] Kochura, Y., Stirenko, S. and Gordienko, Y. "Comparative performance analysis of neural networks architectures on H2O platform for various activation functions," 2017 IEEE International Young Scientists Forum on Applied Physics and Engineering (YSF), Lviv, 2017,

[120] Kochura, Y., Stirenko, S., Alienin, O., Novotarskiy, M. and Gordienko, Y. "Performance Analysis of Open Source Machine Learning Frameworks for Various Parameters in Single-Threaded and Multi-threaded Modes", Conference on Computer Science and Information Technologies (CSIT'17), 2017

[121] Kolen, J. F. and Kremer, S. C. "Gradient Flow in Recurrent Nets: The Difficulty of Learning Long Term Dependencies," in A Field Guide to Dynamical Recurrent Networks, pp.237-243, 2001.

[122] Kotte O, Elorriaga A., Stokic D. and Scholze S., “Context Sensitive Solution for Collaborative Decision Making on Quality Assurance in Software Development Processes.,” in Intelligent Decision Technologies 2013, 2013.

[123] Kovalev, V., Kalinovsky, A. and Kovalev, S. “Deep Learning with Theano, Torch, Caffe, TensorFlow, and Deeplearning4J: Which One Is the Best in Speed and Accuracy?”, XIII International Conference on Pattern Recognition and Information Processing, Minsk, Belarus, October 2016.

[124] Krizhevsky, A., Sutskever, I. and Hinton, G. E. “ImageNet Classification with Deep Convolutional Neural Networks”, Communications of the ACM, May 2017.

[125] Krompass, D. and Spieckermann, S. "Lowering the Entrance Barrier of Deep Learning with High-Quality Source Code Generation", 2018

[126] Kruchten, P. “The Rational Unified Process:An Introduction (3rd Edition)”,ddison-Wesley Professional, 2003

[127] Lee, J. Y., Lee, J. W., Cheun, D. W. and Kim, S. D. "A Quality Model for Evaluating Software-as-a-Service in Cloud Computing," 7th ACIS International Conference on Software Engineering Research, Management and Applications, Haikou, China, 2-4 December 2009.

[128] Leitner, P., Inzinger, C., Hummer, W., Satzger, B. and Dustdar, S. "Application-level performance monitoring of cloud services based on the complex event processing paradigm," 5th International Conference on Service-Oriented Computing and Applications (SOCA'12), Taipei,Taiwan, 17-19 December 2012.

[129] Li, T. J. J., Li, Y., Chen, F., and Myers, B. A. "Programming IoT devices by demonstration using mobile apps", International Symposium on End User Development, Springer, Cham, pp. 3-17, June 2017.

[130] Lipton, Z. C. and Berkowitz, J “A Critical Review of Recurrent Neural Networks for Sequence Learning”, Computer Science, June 2015.

[131] Liu, L., Wang, B., Yu, B. and Zhong, Q. "Automatic malware classification and new malware detection using Machine Learning", Frontiers of Information Technology & Electronic Engineering 18(9), pp. 1336-1347, September 2017.

[132] Liu, Y. and Traore, I. "Complexity Measures for Secure Service-Oriented Software Architectures, “Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007), Minneapolis, 20-26 May 2007.

[133] Long, J., Shelhamer, E. and Darrell, T. "Fully convolutional networks for semantic segmentation", Computer Vision and Pattern Recognition (CVPR), Boston, MA, 2015.

[134] Luszcz, J. “Apache Struts 2: how technical and development gaps caused the Equifax Breach”, Networks Security, vol. 2018, no. 1, pp. 5–8, January 2018.

[135] Luther M., Mrohs B., Wagner M., Steglich S. and Kellerer W., Situational reasoning – a practical OWL use case, Chengdu, Jiuzhaigou: IEEE, 2005.

[136] Ma, Y., Fakhoury, S., Christensen, M., Arnaoudova, V., Zogaan, W. and Mirakhorli, M “Automatic Classification of Software Artifacts in Open-Source Applications”, 15th International Conference on Mining Software Repositories (MSR), 2018.

[137] Mair, C., Kadoda, G., Lefley, M., Phalp, K., Schofied, C., Shepperd, M. and Webster, S. “An investigation of Machine Learning based prediction systems”, Journal of Systems and Software, 53(1), pp. 23-29, 15 July 2000.

[138] Magana, E., Astorga, A., Serrat, J., and Valle, R., "Monitoring of a virtual infra-structure testbed”, Latin-American Conference on Communications, Medellin, Colombia, 10-11 September 2009.

[139] Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., Mcllraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N. and Sycara, K. "OWL-S: Semantic Markup for Web Services", 2004

[140] Martin, J. “Rapid Application Development”, Macmillan Publishing Company, 1991.

[141] Marvie, R. "An Introduction to Test-Driven Code Generation", EuroPython 2006 Refereed Paper Track, 2006

[142] Mastelic, T., Emeakaroha, V.C., Maurer, M. and Brandic, I. "M4Cloud - Generic application level monitoring for resource-shared cloud environments", 2nd International Conference on Cloud Computing and Services Science (CLOSER 2012), Porto, Portugal, April 2012.

[143] McAfee, “Net Losses: Estimating the Global Cost of Cybercrime”, Intel Security, 2014.

[144] McCall, J. A., Richards, P. K. and Walters, G. F. "Factors in Software Quality," National Technology Information Service, 1(2-3), 1977.

[145] McCulloch, W. S. and Pitts, W. “A logical calculus of the ideas immanent in nervous activity”, The bulletin of mathematical biophysics, 5, pp. 115–133, December 1943.

[146] McGraw, G. "Software Security: Building Security", Addison-Wesley Professional, 2006.

[147] McGraw, G. “Automated code review tools for security”, Computer, vol. 41, no. 12, 2008.

[148] McGraw, G. “Software Security,” Datenschutz und Datensicherheit - DuD, vol. 36, no. 9, pp. 662–665, 2012.

[149] McGuinness, D. and van Harmelen, F. "OWL Web Ontology Language Overview", 2004

[150] McNerney, T. S. “From turtles to Tangible Programming Bricks: explorations in physical language design”, Personal and Ubiquitous Computing, pp. 326– 337, 2004.

[151] McNerney, T. S. “Tangible programming bricks: An approach to making programming accessible to everyone”. PhD thesis. Massachusetts Institute of Technology, 1999.

[152] Medeiros, I., Neves, N. and Correia, M. “Detecting and Removing Web Application Vulnerabilities with Static Analysis and Data Mining,” IEEE Transactions on Reliability, vol. 65, no. 1, 2016.

[153] Miškuf, M. and Zolotová, I. "Comparison between multi-class classifiers and Deep Learning with focus on industry 4.0," 2016 Cybernetics & Informatics (K&I), Levoca, 2016

[154] Mohammed Elhag , A. A. and Mohamad, R. "Service-oriented design measurement and theoretical validation," J. Teknol., Volume 77, Number 9, pp. 1-14,2015.

[155] Mohammed, N. M., Niazi, M., Alshayeb, M. and Mahmood, S. “Exploring Software Security Approaches in Software Development Life cycle: A Systematic Mapping Study”, Computer Standards and Interfaces, vol. 50, pp. 107–115, February 2017.

[156] Molino, P., Dudin, Y. and Miryala, S. S. "Ludwig: a type-based declarative Deep Learning toolbox"

[157] Munaiah, N., Camilo, F., Wigham, W., Meneely, A. and Nagappan, M. “Do bugs foreshadow vulnerabilities? An in-depth study of the chromium project,” Empiral Software Engineering, vol. 22, no. 3, pp. 1305–1347, 2017.

[158] Murali, V., Qi, L., Chaudhuri, S. and Jermaine, C. "Neural Sketch Learning for Conditional Program Generation", 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3 2018.

[159] Muske, T. and Serebrenik, A. “Survey of Approaches for Handling Static Analysis Alarms,” 16th International Working Conference on Source Code Analysis and Manipulation (SCAM), 2016, pp. 157–166.

[160] Myers, B. A., Ko, A. J., LaToza, T. D. and Yoon, Y. "Programmers Are Users Too: Human-Centered Methods for Improving Programming Tools", Computer, vol. 49, no. 7, pp. 44-52, July 2016.

[161] Nair, M. K. and Gopalakrishna, V. "CloudCop: Putting network-admin on cloud nine towards Cloud Computing for Network Monitoring”, International Conference on Internet Multimedia Services Architecture and Applications (IMSAA;09), Bangalore, India, 9-11 December 2009.

[162] Ngan L. D., Tran, B. D., Tan, P. S., Soong Goh, A. E. and Lee, E. W. "MODiCo: A Multi-Ontology Web Service Discovery and Composition System", International Conference on Web Engineering (ICWE;09), 2009.

[163] Ngan, L. D. and Kanagasabai, R. "Semantic Web service discovery: State-of-the-Art and research challenges", Personal and Ubiquitous Computing volume 17, pp. 1741–1752, 2013.

[164] Ngan, L. D., Hang, T. M. and Soong Goh, A. E. "Semantic Similarity between Concepts from Different OWL Ontologies," 4th IEEE International Conference on Industrial Informatics, Singapore, 2006.

[165] Nguyen, A. T. and Nguyen, T. N. "Automatic Categorization with Deep Neural Network for Open-Source Java Projects," 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), Buenos Aires, 2017

[166] Nickolls, J., Buck, I., Garland, M. and Skadron, K. “Scalable Parallel Programming with CUDA”, Queue, March 2008.

[167] Nik Daud, N. M. and Kadir, W. M. N. W. "Systematic mapping study of quality attributes measurement in service-oriented architecture," 8th International Conference on Information Science and Digital Content Technology (ICIDT2012), Jeju, 2012.

[168] Nottamkandath, A. "High-precision Web Application Monitoring", Master thesis, Parallel and Distributed Computing, Advisor: C. Stratan, 2011

[169] Novikoff, A. “On Convergence Proofs for Perceptrons”, Computer Science, 1963.

[170] Nwankpa, C., Ijomah, W., Gachagan, A. and Marshall, S. “Activation Functions: Comparison of trends in Practice and Research for Deep Learning”, Machine Learning, (submitted on November 2018).

[171] Ochno, T. “Toyota Production System - Beyond Large-Scale Production”, CRC Press, 1988.

[172] Olovsson, T. "A Structured Approach to Computer Security", Technical Report, 1992.

[173] Osterweil, L. “Software processes are software too", 9th international conference on Software Engineering (ICSE '87), March 1987.

[174] Oundhakar, S., Verma, K., Sivashanugam, K., Sheth, A. and Miller, J. "Discovery of Web Services in a Multi-Ontology and Federated Registry Environment". International Journal of Web Services Research, 1 (3), 2005

[175] Paganelli F., Bianchi G., and Giuli D., “A Context Model for Context-Aware System Design Towards the Ambient Intelligence Vision: Experiences in the eTourism Domain.,” Universal Access in Ambient Intelligence Environments, 2007.

[176] Pahl, C and Jamshidi, P. “Microservices: A Systematic Mapping Study,” 6th International Conference on Cloud Computing and Services Science (CLOSER'16), Setubal, Portugal, 2016.

[177] Pascarella, L. and Bacchelli, A. "Classifying Code Comments in Java Open-Source Software Systems", 14th International Conference on Mining Software Repositories (MSR), Buenos Aires, 2017.

[178] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L. and Lerer, A. "Automatic differentiation in PyTorch", Workshop Autodiff Program Chairs (NIPS'17), 13 Nov 2017

[179] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J. and Chintala, S. "PyTorch: An Imperative Style, High-Performance Deep Learning Library",Curran Associates, Inc., 2019

[180] Perepletchikov, M., Ryan, C. and Frampton, K. "Cohesion Metrics for Predicting Maintainability of Service-Oriented Software”, Seventh International Conference on Quality Software (QSIC'07), Portland, OR, 11-12 Octomber 2007.

[181] Perepletchikov, M., Ryan, C., Frampton, K. and Tari, Z. "Coupling Metrics for Predicting Maintainability in Service-Oriented Designs", Australian Software Engineering Conference (ASWEC'07), Melbourne, 10-13 April 2007.

[182] Perini, A., Susi, A. and Avesani, P. "A Machine Learning Approach to Software Requirements Prioritization," Transactions on Software Engineering, 39 (4), pp. 445-461, April 2013.

[183] Petersen, K., Feldt, R., Mujtaba, S. and Mattsson, M. "Systematic mapping studies in software engineering", 12th International Conference on Evaluation and Assessment in Software Engineering (EASE'08), British Computer Society Swinton, Bari, Italy, pp. 68-77, 26 - 27 June 2008.

[184] Popek, G. and Goldberg, R."Formal requirements for virtualizable third generation architectures", Communications of the ACM, Volume 17, Number 7, pp. 412-421, July 1974.

[185] Poppendieck, M. and Poppendieck, T. “Lean Software Development: An Agile Toolkit (The Agile Software Development Series)”, Addison-Wesley Professional, 1 Edition, 2003.

[186] Qingqing, Z. and Xinke, L. "Complexity Metrics for Service-Oriented Systems", 2nd International Symposium on Knowledge Acquisition and Modeling, Wuhan, China, 30 November-1 December 2009.

[187] Rezende, E., Ruppert, G., Carvalho, T., Ramos, F. and de Geus, P. "Malicious Software Classification Using Transfer Learning of ResNet-50 Deep Neural Network," 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA), Cancun, 2017, pp. 1011-1014.

[188] Rezende, E., Ruppert, G., Carvalho, T., Theophilo, A., Ramos, F. and de Geus, P. "Malicious Software Classification using VGG16 Deep Neural Network's Bottleneck Features", Information Technology, January 2018

[189] Richardson, C. "Microservices patterns : with examples in Java", Shelter Island, New York, Manning Publications, 2019

[190] Rocco, D., Caverlee, J.,Liu, L. and Critchlow, T. "Domain-specific Web service discovery with service class descriptions," IEEE International Conference on Web Services (ICWS'05), Orlando, FL, 2005

[191] Rodriguez, M. A. and Egenhofer, M. J. "Determining semantic similarity among entity classes from different ontologies," in IEEE Transactions on Knowledge and Data Engineering, vol. 15, no. 2, pp. 442-456, March-April 2003.

[192] Rosenblatt, F. "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain", Psychological Review, 65 (6), 1958.

[193] Royce, W. W. "Managing the development of large software Systems", 9th international conference on Software Engineering (ICSE '87), March 1987.

[194] Rud, D., Schmietendorf, A. and Dumke, R. R. "Product Metrics for Service-Oriented Infrastructures", International Workshop on Software Metrics and DASMA Software Metrik Kongress (IWSM/MetriKon'06), January 2006.

[195] Rumelhart, D. E., Hinton, G. E. and Williams, R. J. "Learning representations by back-propagating errors", Neurocomputing: foundations of research, pp. 696–699, January 1988.

[196] Ruthruff, J. R., Penix, J., Morgenthaler, J. D., Elbaum, S. and Rothermel, G. "Predicting accurate and actionable static analysis warnings," 30th international conference on Software engineering, May 2008

[197] Scandariato, R., Walden, J., Hovsepyan, A. and Joosen, W. "Predicting vulnerable software components via text mining," IEEE Transactions on Software Engineering, vol. 40, no. 10, pp. 993–1006, 2014.

[198] Schmidhuber, J. "One Big Net for Everything", Technical Report, 24 February 2018.

[199] Scholze S., Siafaka R., Nagorny K., Zilverberg A., and Krone K., "Situation-aware Re-configuration of Production Processes," in International Workshop on Reconfigurable and Communicationcentric Cyber-Physical Systems ReCoCyPS 2019, 2019.

[200] Scholze S., Barata J. and Stokic D., "Holistic Context-Sensitivity for Run-Time Optimization of Flexible Manufacturing Systems," Journal Sensors, no. 17, p. 455, 2017.

[201] Scholze S., Nagorny K., Stöbener K. and Brückner D., "An Approach for Context Sensitive Product Extensions Services," INDIN2017, 2017.

[202] Scholze S., Correia A. T., and Nagorny K., "Services for development of Situational Aware Intelligent PSS," in ICE International Conference on Engineering, Technology and Innovation, Madeira, Portugal, 2017.

[203] Schölkopf, B. Herbrich, R. and Smola, A. J. "A Generalized Representer Theorem", International Conference on Computational Learning Theory (COLT'01), Amsterdam, Netherlands, 16-19 July 2001.

[204] Schuster, M. and Paliwal, K. K. "Bidirectional Recurrent Neural Networks", *Transactions on Signal Processing*, 45 (11), November 1997.

[205] Shabtai, A., Moskovitch, R., Feher, C., Dolev, S. and Elovici, Y. "Detecting unknown malicious code by applying classification techniques on OpCode patterns", Security Informatics volume 1, Article number: 1, 2012.

[206] Shao, J. and Wang, Q. "A performance guarantee approach for cloud applications based on monitoring", 35th Annual Computer Software and Applications Conference Workshops, Munich, Germany, 18-22 July 2011.

[207] Shim, B., Choue, S., Kim, S. and Park, S. "A Design Quality Model for Service-Oriented Architecture", 15th Asia-Pacific Software Engineering Conference, Beijing, China, 3-5 December 2008.

[208] Serafini L. and Bouquet P., "Comparing formal theories of context," AI. Artif. Intell., pp. 41-67, 1-2 05 2004.

[209] Sherstinsky, A. “Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network”, Physica D: Nonlinear Phenomena, Special Issue on Machine Learning and Dynamical Systems, 404, March 2020.

[210] Siavvas, M., Gelenbe, E., Kehagias, D. and Tzovaras, D. “Static Analysis-Based Approaches for Secure Software Development,” Euro-CYBERSEC 2018: Security in Computer and Information Sciences, pp 142-157, 2018.

[211] Siavvas, M., Kehagias, D. and Tzovaras, D. “A preliminary study on the relationship among software metrics and specific vulnerability types,” International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 14-16 Dec. 2017.

[212] Sindhgatta, R., Sengupta, B. and Ponnalagu, K. "Measuring the Quality of Service Oriented Design," Context, pp. 485-499, 2009.

[213] Singh, S. and Jindal, S “Designing Deep Learning Neural Networks using Caffe”, January 2015.

[214] Soldani, J., Tamburri, D. A. and Van Den Heuvel, W.-J. “The pains and gains of microservices: A Systematic grey literature review”, Journal of Systems and Software, vol. 146, pp. 215–232, December 2018.

[215] Sorli M. and Stokic D., Innovating in Product/Process Development, Heidelberg, London, New York: Springer-Verlag, 2009.

[216] Strang T. and Linnhoff-Popien C., “A Context Modeling Survey. in Workshop on Advanced Context Modelling, Reasoning and Management as part of the Conference on Ubiquitous Computing,” in The Sixth International Conference on Ubiquitous Computing, Nottingham, 2004.

[217] Stokic D., Scholze S., and Kotte O., “Generic Self-Learning Context Sensitive Solution for Adaptive Manufacturing and Decision-Making Systems,” in ICONS 2014, Nice, 2014.

[218] Suleiman, D. and Al-Naymat, G. "SMS Spam Detection using H2O Framework", Procedia Computer Science Volume 113, pp. 154-161, 2017.

[219] Sutherland, J. and Schwaber, K.” The Scrum Development Process”, Object-Oriented Programming, Systems, Languages & Applications (OOPSLA' 95), Austin, Texas, 1995.

[220] Sutskever, I., Vinyals, O. and Le, Q. V. “Sequence to Sequence Learning with Neural Networks”, Computation and Language, 2014.

[221] Tamilselvam, S., Panwar, N., Khare, S., Aralikatte, R., Sankaran, A. and Mani, S. “A Visual Programming Paradigm for Abstract Deep Learning Model Development”, 10th Indian Conference on Human-Computer Interaction (IndiaHCI '19), November 2019.

[222] Tang, X., Wang, Z., Qi, J. and Li, Z. "Improving Code Generation from Descriptive Text by Combining Deep Learning and Syntax Rules", 31st International Conference on Software Engineering and Knowledge Engineering, SEKE 2019, Hotel Tivoli, Lisbon, Portugal, July 10-12, 2019

[223] Tang, Y., Zhao, F., Yang, Y., Lu, H., Zhou, Y. and Xu, B. “Predicting Vulnerable Components via Text Mining or Software Metrics? An Effort-Aware Perspective,” International Conference on Software Quality, Reliability and Security, Vancouver, BC, Canada, 3-5 Aug. 2015.

[224] Tao Y., Tianyuan X. and Linxuan Z., "Context-centered design knowledge management," Computer Integrated Manufacturing Systems, no. 10, pp. 1541-1545, 2004.

[225] Tian, K., Revelle, M. and Poshyvanyk, D. "Using Latent Dirichlet Allocation for automatic categorization of software," 2009 6th IEEE International Working Conference on Mining Software Repositories, Vancouver, BC, 2009.

[226] Tian, Y., Lo, D., Xia, X. and Sun, C. “Automated prediction of bug report priority using multi-factor analysis”, Empirical Software Engineering, 20 (5), 1354-1383, 2015

[227] Trihinas, D., Pallis, G., and Dikaiakos, M. D. “JCatascopia: Monitoring Elastically Adaptive Appli-cations in the Cloud”, 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Chicago, IL, USA, 26-29 May 2014.

[228] Turing, A. M. “Computing Machinery and Intelligence”, Mind, 49, pp. 433-460, 1950.

[229] Theano Development Team, “Theano: A Python framework for fast computation of mathematical expressions”, CoRR, May 2016.

[230] van Vliet, H. “Software Engineering: Principles and Practice”, John Wiley & Sons, 2008.

[231] Viega, J., Bloch, J. T., Kohno,Y. and McGraw, G. “ITS4: A static vulnerability scanner for C and C++ code,” 16th Annual Computer Security Applications Conference (ACSAC'00), New Orleans, LA, USA, USA, 11-15 Dec. 2000.

[232] Vural, H., Koyuncu, M. and Guney, S. “A Systematic Literature Review on Microservices”, Computational Science and Its Applications (ICCSA'17), Trieste, Italy, 15 July 2017.

[233] Walden, J., Stuckman, J. and Scandariato, R. “Predicting vulnerable components: Software metrics vs text mining,” 25th International Symposium on Software Reliability Engineering, Naples, Italy, 3-6 Nov. 2014.

[234] Wang, Y., Cai, W. and Wei, P. "A Deep Learning approach for detecting malicious JavaScript code", Security and Communication Networks, Volume 9, Issue 11, 2016

[235] Widrow, B. and Hoff, M. E. “Adaptive switching circuits”, Neurocomputing: foundations of research, pp. 123–134, January 1988.

[236] Widrow, B. and Stearns, S. D. “Adaptive signal processing”, Prentice-Hall, May 1985.

[237] Wurster, G. and van Oorschot, P. C. “The developer is the enemy”, Proceedings of the 2008 New Security Paradigms Workshop (NSPW ’08), 2008.

[238] Yang, J., Ryu, D. and Baik, J. “Improving vulnerability prediction accuracy with Secure Coding Standard violation measures,” International Conference on Big Data and Smart Computing (BigComp), Hong Kong, China, 18-20 Jan. 2016.

[239] Yarygina, T. and Bagge, A. H. “Overcoming Security Challenges in Microservice Architectures,” Symposium on Service-Oriented System Engineering (SOSE), Bamberg, Germany, 26-29 March 2018.

[240] Younis, A. A., Malaiya, Y. K. and Ray, I. “Using attack surface entry points and reachability analysis to assess the risk of software vulnerability exploitability,” 15th International Symposium on High-Assurance Systems Engineering, Miami Beach, FL, USA, 9-11 Jan. 2014.

[241] Zhang, H., Shanshan, L., Jinfeng, S., Zijia J. and Zheng L. "Understanding Quality Attributes of Microservices Architecture", Technical Report, Januar 2018

[242] Zhang, Q., Cheng, L. and Boutaba, R. "Cloud computing: State-of-the-Art and research challenges", Journal of Internet Services and Applications, pp. 7-18, 2010.

[243] Zhou, Y. and Leung, H. "Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults," *Transactions on Software Engineering*, 32(10), pp. 771-789, Oct. 2006.