The Journal of Systems & Software 180 (2021) 111010

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

A sustainable-development approach for self-adaptive cyber–physical system's life cycle: A systematic mapping study☆

Luisa Restrepo [a], Jose Aguilar [a,b,c,*], Mauricio Toro [a], Elizabeth Suescún [a]

[a] RID on Information Technologies and Communications Research Group, Universidad EAFIT, Medellín, Colombia
[b] CEMISID Universidad de Los Andes, Mérida, Venezuela
[c] Universidad de Alcalá, Dpto. Automática, Alcalá de Henares, Spain

ARTICLE INFO

Article history:
Received 2 November 2020
Received in revised form 4 May 2021
Accepted 17 May 2021
Available online 8 June 2021

Keywords:
Self-adaptive systems
Sustainability
Cyber–physical systems
Systems-development life-cycle

ABSTRACT

Cyber–Physical Systems (CPS) refer to a new generation of systems where the cyber and physical layers are –strongly– interconnected. The development of these systems requires two fundamental parts. First, the design of sustainable architectures –centered on adaptation, throughout a System-Development Life-Cycle (SDLC)– to develop robust and economically profitable products. Second, the use of self-adaptive techniques to adjust CPSs to the evolving circumstances of their operation context. This work presents a systematic mapping study (SMS) that discusses different approaches used to develop *self-adaptive CPSs* (SA-CPSs) at each stage of the SDLC, focused on sustainability. The results show trends such as (i) Designs are not limited to particular application domains, (ii) Performance was the most commonly used attribute, and (iii) Monitor–Analyze–Plan–Execute over a shared Knowledge (MAPE-K) is the predominant feedback loop applied in the cyber layer. The results also raise challenges such as (i) How to design and evaluate sustainable SA-CPSs, (ii) How to apply unit and integration testing in the development of SA-CPSs, and (iii) How to develop feedback loops on SA-CPSs with the integration of machine-learning techniques.

1. Introduction

Cyber–Physical Systems (CPSs) are systems composed of collaborative computational elements to control physical entities. Also, CPSs can be defined as complex and complicated systems that require techniques of sophisticated design, which include interaction between the physical world and the cyber world. CPSs integrate (i) Mathematical modeling of physical systems, (ii) Formal computation models, (iii) Simulation of heterogeneous systems, (iv) Software engineering strategies, and (v) Verification and validation methods (Jensen et al., 2011).

A concept associated with CPSs is the *Internet of Things (IoT)*, where communication is very important (Marwedel, 2018), in which systems are interconnected and collaborate. Taken together, CPSs and IoT will conform to most of the future applications of *information technology*.

The design of CPSs is a task that has to be broken down into several sub-tasks to be tractable (Marwedel, 2018). Most CPSs are designed for specific types of requirements (Stankovic, 2014). Usually, these requirements concern both the physical and the cyber parts, and functional and non-functional aspects. In the physical part, actuators, sensors, and embedded-system processors are used for computer-controlled tasks. In turn, the physical part must interact with the cyber part, implemented through software systems, in order (i) to process data from the entire CPS, (ii) to diagnose all types of system failures, (iii) to make real-time decisions to prevent major failures, and (iv) to make data-based decisions that exhibit real-world behavior (Lee, 2008).

The use of self-adaptation techniques, in CPSs, is considered an effective approach to deal with changes in its environment and structure. Current challenges include the design and development of effective, energy-efficient, and sustainable self-adaptive CPSs (SA-CPSs) (Chantem et al., 2019).

According to Koziolek (2011), sustainability implies the development of technically–robust and economically–profitable products. Although sustainability has been more associated with the environmental context, it is becoming –increasingly– important in the context of engineering, in general, and software engineering, in particular (Pankowska, 2013). In software systems that are part of a CPS, sustainability is –strongly– linked to non-functional attributes such as maintainability. Koziolek et al. define that maintainability is divided in the following non-functional

☆ Editor: Heiko Koziolek.

* Corresponding author at: CEMISID Universidad de Los Andes, Mérida, Venezuela.
E-mail addresses: lrestr61@eafit.edu.co (L. Restrepo), jlaguilarc@eafit.edu.co, aguilar@ua.ve (J. Aguilar), mtorobe@eafit.edu.co (M. Toro), esuescu1@eafit.edu.co (E. Suescún).

https://doi.org/10.1016/j.jss.2021.111010
0164-1212/

attributes: (i) analysability, (ii) stability, (iii) testability, (iv) understandability, (v) modifiability, (vi) portability, and (vii) evolvability (Koziolek, 2011).

The design of CPSs –both the physical part and the cyber part– should include the design of their architecture and its sustainability. Additional to this, their design must consider issues related to self-adaptation to satisfy requirements in a dynamic environment (Zeadally et al., 2019a). The concept of architecture has several meanings (and definitions): The *International Organization for Standarization (ISO)* defines the architectural design as the "process of conceiving, defining, expressing, documenting, communicating, certifying, maintaining and improving an architecture throughout a system's life cycle"(International Organization for Standardization, 2011b). The design of an architecture is a key process in the *System-Development Life-Cycle (SDLC)*, and the quality of the architecture of a system –strongly– determines its sustainability (Koziolek, 2011; Chitchyan et al., 2017).

Understand and identifying sustainability strategies used at each stage of the SDLC of SA-CPSs, is important for the success of sustainable systems, and, particularly, to (i) improve practices; (ii) identify current opportunities, threats, trends; and, also, (iii) serve as an inspiration for the development of future sustainable autonomous systems. Nonetheless, making a system sustainable by adding attributes such as self-adaptation, increasing evolvability and increasing energy-efficiency, may increase its complexity and maintenance (by humans). The increase in complexity is both at the level of development and deployment. The first is related to the way the solution is implemented and the second is related to the context where will be used the solution (domain, process). Also, the maintainability plans allow establishing specific practices, as well as resources and relevant sequences of activities, which can be difficult to be followed/applied by humans. Thus, trade-offs should be taken into account when using sustainability strategies in SA-CPSs considering the different elements involved. Previous works do not carry out an analysis of sustainability strategies used at each stage of the SDLC of the SA-CPSs, based on the above ideas. Lin et al. (2009) point out that existing methods for designing and developing CPSs are usually limited to specific fields of application. Another problem that this work found is that some approaches are focused only on the physical part of the CPSs, ignoring the cyber part, or others only deal with the cyber part, resulting in a lack of integration. Finally, Lin and Panahi propose a framework, for the development of CPSs, with an emphasis on sustainability and predictability. However, they restrict the system architecture to *Service-Oriented Architecture* (SOA), without taking into account the use of other architectural patterns in the design of the CPSs architecture (Lin and Panahi, 2010).

1.1. Related works

In this section, three *Systematic Literature Reviews (SLRs)*, associated with SA-CPSs, were analyzed.

First, Muccini et al. (2016) investigated the role of self-adaptation within CPSs. In their SLR, 42 studies were included. In their analysis, Muccini et al. concluded that aspects –such as performance and reliability– are well covered, and CPS's most challenging aspects –such as interoperability and security– are still barely covered by the literature. Muccini et al. also concluded that *Monitor, Analyze, Plan, Execute, and Knowledge (MAPE-K)* model is the dominant adaptation mechanism, followed by agents and self-organization. The main challenges, defined by Muccini et al., were: (i) How to map these aspects to layers and adaptation mechanisms, (ii) How to integrate adaptation mechanisms within and across layers, and (iii) How to ensure system-wide consistency of adaptation.

Second, Musil et al. (2017) surveyed CPS studies that apply the promising design strategy of combining different self-adaptation mechanisms across the technology layers of the system (physical, proxy, communication, service and middleware, application, and social layers). In their research, Musil et al. identified performance as the dominant adaptation purpose. For the adaptation mechanisms, Musil et al. identified smart elements, multi-agent systems, and MAPE-K as the most applied.

Third, Zeadally et al. (2019a) established the state-of-the-art on CPSs from the self-adaptation perspective and evaluated the main self-adaptive approaches proposed, in the literature. Zedeally also evaluated the techniques to enable self-adaptation capabilities –within CPSs– at different architectural layers. An important conclusion is that adaptation should be implemented in all layers of a CPSs, and that researchers must adopt a holistic view of CPSs that includes (i) Self-adaptation, (ii) Autonomy, (iii) Efficiency, (iv) Functionality, (v) Reliability, (vi) Safety, (vii) Scalability, and (viii) Usability. Zedeally et al. defined as research opportunities the development of cost-effective self-adaptation cross-layer solutions, as well as run-time model-driven approaches that manage requirements.

1.2. Our contribution

In contrast to the SLRs of Zeadally et al. (2019a), Musil et al. (2017), and Muccini et al. (2016), this article discusses the designs of SA-CPSs from a perspective of the *SDLC* and identifies –at each stage of the SDLC–: (i) How the development was carried out and (ii) How sustainability –from both, the technical and economical perspectives– was taken into account. These two perspectives are the most relevant to CPSs according to Koziolek et al.'s SLR (Koziolek, 2011).

The contribution of this article is summarized as follows: A general overview of the strategies used for the development of SA-CPSs, gaps found in each stage of the SDLC, and finally, trends and future-research directions.

1.3. Organization

The present document is structured as follows. Section 2 presents the basis of sustainable and self-adaptive techniques, in CPS, and the SDLC. Section 3 presents the methodology to search and select relevant articles. Section 4 groups the articles found in the literature, for each phase of the SDLC. Section 5 analyzes the results of the reviewed articles, and presents the trends and limitations found. Section 6 presents the potential challenges. Finally, Section 7 outlines the conclusions of this research.

2. Background

This section is divided into three parts. First, it presents the SDLC. Second, it introduces the definition of sustainability of systems. Finally, it defines self-adaptability in CPSs.

2.1. System-development life-cycle (SDLC)

The SDLC is the framework to review the articles in this SMS. SDLC defines all the stages a system goes through. This life cycle is common to systems and software projects, and serves as a framework to understand how systems are built. The SDLC follows a set of four fundamental stages: (i) Planning, (ii) Analysis, (iv) Design, and (v) Implementation (Dennis et al., 2014). In software-engineering projects, it is common to have the testing and maintenance stages, separated from the implementation stage (Sommerville, 2015). Different projects may emphasize different parts of the SDLC, or approach the SDLC stages in different

ways; for instance, the work of Sánchez Aristizábal and Sarmiento Garavito (2019).

For this SMS, a SDLC is proposed in Fig. 1. This life cycle includes the planning, specification and analysis, design, implementation, integration, quality control, and maintenance stages, defined as follows.

Planning: In this stage of the development of a system, it is fundamental to (i) identify business needs to build a system, (ii) understand why the system should be built, (iii) identify the system's contribution to the organization or context, (iv) evaluate if the system is economically, technically and organizationally feasible, and (v) establish a work plan to control the project through the entire SDLC (Dennis et al., 2014).

Specification and analysis: This stage answers the questions of (i) who will use the system, (ii) what the system will do, and (iii) where and when the system will be used because the application domain determines –largely– how a project will be oriented and executed (Züllighoven, 2005). During this phase, the project team investigates existing systems, identifies improvement opportunities, and develops a concept for the new system (Dennis et al., 2014).

Design: Usually, at this stage, it is decided (i) how the system will operate in terms of hardware, software, and networking infrastructure that will be in place; (ii) how the user interface, forms, and reports will be used; and (iii) what specific programs, databases, and files will be needed. Although most of the strategic decisions about the system are made in the analysis phase. The design phase determines –exactly– how the system will operate. Finally, at this stage, the system architecture must be designed to guarantee the levels of the quality attributes specified (Dennis et al., 2014).

Implementation: This stage consists of building the system, to create the functionalities defined during the design stage (Sommerville, 2015). Many strategies and software tools can be used in this phase (e.g., open-source tools, *Integrated Development Environments [IDEs]* to develop programs, simulation platforms, *Commercial Off-The-shelf [COT]*, and micro-controllers).

Integration: This stage integrates information technologies (e.g., servers, databases, applications, and platforms) and physical objects (e.g., mechanic and electronic) using communication technologies (Dennis et al., 2014; Marwedel, 2018; Pahl et al., 2007).

Quality control: This stage aims to validate and verify that the system (i) is appropriate, (ii) meets all requirements, and (iii) will perform as expected. According to Marwedel (2018), this stage is –extremely– important for safety-critical embedded systems. System tests are conducted to ensure that all modules and programs meet business requirements, and acceptance tests are done to ensure that the system meets business needs such as usability, security, or performance (Dennis et al., 2014).

Maintenance: This stage is the process of refining a system where corrective, adaptive, perfective, and preventive maintenance are made. "Corrective maintenance is performed to fix errors, adaptive maintenance adds new capability and enhancements, perfective maintenance improves efficiency, and preventive maintenance reduces the possibility of future system failure" (Shelly and Rosenblatt, 2011).

Traditionally, to develop software products, methodologies are used. Such methodologies set up the framework that structures the phases described above –such as the waterfall model, iterative model, spiral model, and V-model (Sommerville, 2015)–; however, most of them lack the generality to be used in CPSs. For that reason, this research focuses on the SDLC proposed in Fig. 1 and described above.

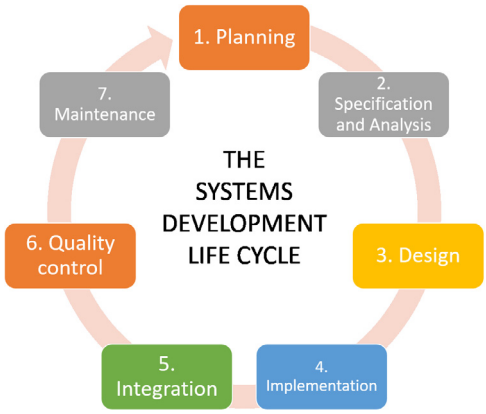

Fig. 1. System-Development Life-Cycle from Dennis et al. (2014), Sommerville (2015).

2.2. Sustainable development

Sustainable development is the practice of "meeting the needs of society today without compromising the ability of future generations to meet their own needs" (Stavros and Sprangel, 2008). In engineering, *sustainability* can be understood as the selection and implementation of iterative and incremental methodologies, which support the development of technologies in the long term, at low cost, and with reduced effort (Pankowska, 2013).

Becker et al. (2015) identified five sustainability dimensions: (i) environmental, (ii) social, (iii) economic, (iv) technical, and (v) individual. Nonetheless, Koziolek et al. in a previous SLR, found that the most relevant dimensions for CPSs are the economic and technical (Koziolek, 2011). This is the reason why our SMS focuses on the technical and economic dimensions, but future studies must consider the social and environmental dimensions. The economic dimension includes aspects such as capital, profitability, investment, income, and wealth creation. The technical dimension, according to Beckert et al. refers to the longevity of software systems and infrastructure, and their adequate evolution with changing surrounding conditions, including maintenance, innovation, obsolescence, and data integrity.

The main quality attributes of sustainable system architecture are Koziolek (2011): (i) maintainability, (ii) portability, and (iii) evolvability. In what follows, these three quality attributes are explained based on their sub-characteristics.

Maintainability: ISO/IEC 25010 (International Organization for Standardization, 2011a) defines this attribute as the capability of a product or system to facilitate maintenance activities –such as corrections, improvements, or adaptation to changes in the environment–, of requirements and functional specifications. Also, maintainability includes the installation of updates and upgrades. This attribute is subdivided into five sub-characteristics: (i) modularity, (ii) reusability, (iii) analysability, (iv) modifiability, and (v) testability. Maintainability is also related to evolvability.

Portability: According to ISO/IEC 25010, it is the "degree of effectiveness and efficiency with which a system, product or component can be transferred from one hardware, software or other operational or usage environment to another" (International Organization for Standardization, 2011a). This attribute is subdivided into three sub-characteristics : (i) adaptability, (ii) installability, and (iii) replaceability.

Evolvability: According to Rowe et al. (1994), it is an "attribute that bears on the ability of a system to accommodate changes in its requirements throughout the system's lifespan, with the least possible cost, while maintaining architectural integrity". Pei Breivold (2020) established that this attribute is similar to the

Table 1
Evolvability attribute's sub-characteristics.

Sub-characteristic	Definition
Maintainability (Analysability and testability)	Analysability - "Degree of effectiveness and efficiency with which it is possible to assess the impact on a product or system of an intended change to one or more of its parts, or to diagnose a product for deficiencies or causes of failures, or to identify parts to be modified" (International Organization for Standardization, 2011a) Testability - "Degree of effectiveness and efficiency with which test criteria can be established for a system, product or component, and tests can be performed to determine whether those criteria have been met" (International Organization for Standardization, 2011a).
Maintainability (Modifiability)	This attribute is a combination of changeability and stability (International Organization for Standardization, 2011a) and, according to our criteria is also associated with the ability to extend a system. This attribute is the degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality." (International Organization for Standardization, 2011a).
Security (Integrity)	"Degree to which a system, product or component prevents unauthorized access to, or modification of, computer programs or data" (International Organization for Standardization, 2011a).
Portability	This attribute has been defined previously
Domain-specific attributes	Additional quality sub-characteristics that are required by specific domains (Pei Breivold, 2020).

maintainability attribute, but in evolvability, one should consider unexpected changes. On the one hand, Rowe et al. (1994) defined (i) generality (accommodating change), (ii) adaptability, (iii) scalability, and (iv) extensibility as quality attributes that contribute to evolvability. On the other hand, Pei Breivold (2020) proposed that (i) analysability, (ii) integrity, (iii) changeability, (iv) extensibility, (v) portability, (vi) testability, and (vii) domain-specific attributes are sub-characteristics associated with the evolvability attribute. All these quality attributes can be mapped to the ISO/IEC 25010 model, as shown in Table 1, where these attributes are described.

A sustainable-system architecture must be able to evolve during its life cycle: This means in development and production environments, and this is achieved when the system is prepared for maintenance and evolution, an attribute that –indirectly– includes the concepts of longevity and cost-effectiveness (Koziolek, 2011).

This SMS focuses on the technical and economic perspectives of sustainability. On the technical, this article focuses on the maintainability attribute achieved through quality attributes established in the CPSs architecture (Hammoudi et al., 2018). This attribute improves the evolution of the systems, decreasing lifecycle costs and managing technical debt (Kruchten et al., 2012). From the economic perspective, this SMS focuses on the costs and incomes associated with the use and implementation of these quality attributes.

2.3. Self-adaptive cyber–physical systems

Self-adaptation is the ability of a system to modify its behavior and/or structure in response to changes in its environment and user requirements (De Lemos et al., 2013; Weyns and Georgeff, 2010). There are several feedback loops for the implementation of self-adaptive systems used in the design of CPSs. Typically, the MAPE-K loop is a dominant approach that allows systems to manage themselves given high-level objectives, which separates self-adaptation into the following components (see Fig. 2) (Vizcarrondo et al., 2017).

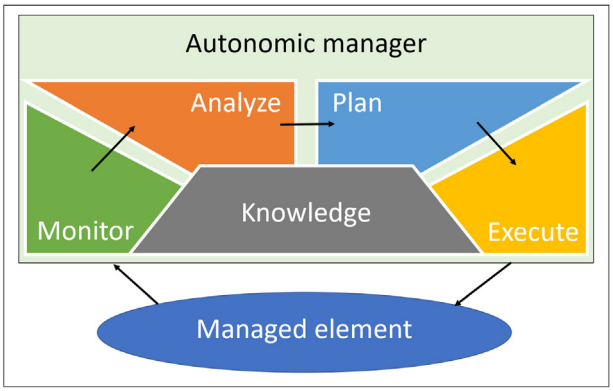

Fig. 2. MAPE-K feedback loop.
Source: Adapted from Kephart and Chess (2003).

Monitor: This component collects information by monitoring context data from sensors and other sources (Seiger et al., 2019), and –constantly– updates the knowledge component. This information serves as the basis of adaptation. The monitor component also supervises their suppliers and clients to ensure that they are receiving and not exceeding the agreed-on level of service, respectively (Kephart and Chess, 2003).

Analyzer: This component performs data analysis, stored on the knowledge component, to determine if a change is needed to satisfy the system goals. *Plan:* If an adaptation is needed, then the plan component creates a procedure to reach a new target condition that satisfies the goals (including the intermediate steps that occur when adapting from one state to another) (Jahan et al., 2020). In this component, strategies for translate-service agreements are needed (Kephart and Chess, 2003). *Execute:* The planned procedure recommended by the plan component is executed on the managed resources. *Knowledge:* This is a shared knowledge-base (Kephart and Chess, 2003) for the other components. The knowledge component comprises data that the MAPE-K loop uses during the adaptation strategies.

MAPE-K is a cyclic process, where context and goals are specified, observed, and managed. Another technique used is *multi-agents systems (MAS)*, which are autonomous approaches to solving problems from artificial intelligence (Weyns and Georgeff, 2010). A MAS provides a way to conceptualize adaptive systems and self-organization of systems defined by interacting autonomous agents, each acting, learning, or evolving –individually– in response to interactions with their environments (Aguilar et al., 2005; Dafflon et al., 2019; Perozo et al., 2008).

3. Methodology

The search strategy and the search process used for the articles are explained as follows.

3.1. Search strategy

This research is a SMS of the methods –currently available– for the design of the SA-CPSs systems, focused on the SDLC process. This research uses the methodology for SMSs proposed by

Table 2
Groups of terms and phrases.

Group	Terms and phrases
G1	(“Embedded Systems” OR “Cyber Physical Systems” OR “CPSs” OR “Cyberphysical Systems” OR “Cyber-physical Systems” OR “Internet of Things” OR ”IoT” OR “Connected things” OR “Autonomous systems” OR “Industrial internet of things” OR “Intelligent Systems” OR “Industry 4.0” OR “fourth industrial revolution”)
G2	(“Self-adaptive” OR “Self Adaptive” OR “Self-adaptiveness” OR “Adaptative” OR “self Adaptation” OR “self-adaptation”)
G3	(“Tools” OR “Instrument” OR “Device” OR “Strategies” OR “Methods” OR “Techniques” OR “Frameworks” OR “Structure” OR “Architecture” OR “Design”)

Kitchenham and Charters (2007). For this research, the following research questions were defined to provide adequate support for SDLC.

- (Q1) Which planning strategies were used for SA-CPS?
- (Q2.1) What was the application domain?, (Q2.2) Which were the specified quality attributes, and (Q2.3) Which specification techniques were used for SA-CPS?
- (Q3.1) What self-adaptive techniques were used for SA-CPS?, and (Q3.2) What architecture styles were used for SA-CPS?
- (Q4.1) How self-adaptation was implemented for SA-CPS?, (Q4.2) How SA-CPS was implemented in the cyber layer, (Q4.3) physical layer, and (Q4.4) network layer?
- (Q5) How SA-CPS components (e.g., sensors, actuators, servers, and databases) were integrated?
- (Q6.1) How many application domains were tested? (Q6.2) How SA-CPS solutions were validated, and (Q6.3) Which quality attributes were verified?
- (Q7) Which strategies were planned for the maintenance of SA-CPS?
- (Q8) How technical and economical sustainability was taken into account at each phase of the SDLC, to develop SA-CPS?

To answer the previous research questions, three groups of terms and phrases were defined. These terms and phrases were used in the search process, and are listed in Table 2.

The search string was generated combining the previous groups of terms and phrases. The search string is the concatenation of G1, G2 and G3, showed in Table 2. The boolean search string used in this research is “G1 AND G2 AND G3”.

The following restrictions to include/exclude publications were defined. These criteria were developed to find the most relevant articles to solve the research questions, and to exclude the articles that do not fit this research.

There are four *inclusion criteria (IC)*, numbered from IC1 to IC4, as follows. *IC1:* Journal articles, conference papers, and book chapters whose titles and abstract are related to frameworks or architectures for self-adaptive CPSs. *IC2:* Journal articles, conference papers, and book chapters published between January 2010 and September 2020. *IC3:* Journal articles, conference papers, and book chapters available in electronic form. Finally, *IC4:* Journal articles, conference papers, and book chapters in the English language.

There are two *exclusion criteria (EC)*, numbered from EC1 to EC2, as follows. *EC1:* Documents whose methods or techniques do not apply to frameworks or architectures for self-adaptive CPSs. *EC2:* Documents in the form of events, posters, unpublished works, and secondary studies.

3.2. Search process

The search process was composed of five stages, based on the study proposed by Li et al. (2015): (i) selection by title, (ii) snowballing, (iii) first-results merge (iv) selection by abstract, and (v) selection by full text. Each stage is detailed below and summarized in Fig. 3.

Selection by title: The search process used the search strings in Scopus and Web of Science, and the search was extended by looking at Google Scholar, as shown in Fig. 3; after, candidate documents were selected based on the title. Inclusion criteria IC1, IC2, IC3, and IC4 were applied in this step. At the end of this step, 120 articles remained.

Snowballing: The backward “snowballing” technique was performed to find other –potentially– relevant documents. Snowballing consists of checking the references of the previously selected documents (Wohlin, 2014). This process could be iterated as many as new documents are found; however, only the first iteration was applied. At the end of this step, 39 new articles were selected.

First-results merge: All candidate documents were merged for each research question; however, duplicated studies were found. A duplicated study is the one that is retrieved from different search sources (i.e., digital libraries) because of the overlapping between these sources. Duplicated documents were excluded at the first stage of scanning, keeping only one version of the document (the most complete, extended, or recent version). In the end, 27 duplicated documents were removed. The total of selected documents, at this stage, is illustrated in Fig. 3.

Selection by abstract: The candidate documents’ abstracts were analyzed to guarantee that they were related to the desired topic (i.e., SA-CPSs); at this point, 24 candidate documents remained.

Selection by full text: The previous documents’ full texts were analyzed and cross-checks were performed by the authors to validate the inclusion of each document, and as a result, 16 studies remained to build Table 3. Exclusion criteria EC1, and EC2 were applied at this step.

For the 16 documents selected, at the end of the search process, it was identified that there is a growing trend in the scientific production of SA-CPSs. Sweden and Italy are the countries with the largest number of articles. Bures, T. and Gerostathopoulos, I. are the most productive authors, and the most cited were do do Nascimento and de Lucena (2017), and Iftikhar et al. (2017). The subject area of the research is –mainly– in computer science and engineering. The word-cloud for the search process result is shown in Fig. 4. The size of each term indicates its frequency or importance. The most common terms were (i) IoT, (ii) adaptation, (iii) adaptive, (iv) software, (v) architecture, (vi) cyber, (vii) physical, and (viii) embedded, as they were the focus of research. Replication package is available in Mendeley repository (Restrepo, 2021).

3.3. Data items and extraction process

In this section, the methodology to review the articles is detailed, which follows the workflow proposed in Fig. 1. In Section 4, the articles related to each stage of the SDLC are reviewed. In general, the data of each paper was extracted and analyzed through a cross-check process among the authors. Particular characteristics are taken from each stage, in Fig. 1, through a cross-check process to validate the inclusion of each characteristic. These characteristics are detailed in Table 3, where the characteristics and sub-characteristics to be analyzed, for each stage, are listed. The characteristics represent some paths that the reviewed articles, commonly, follow for each stage of the SDLC.

In Table 3, the *Planning* stage divides the articles that implemented any planning strategy (Yes) and the articles that did not

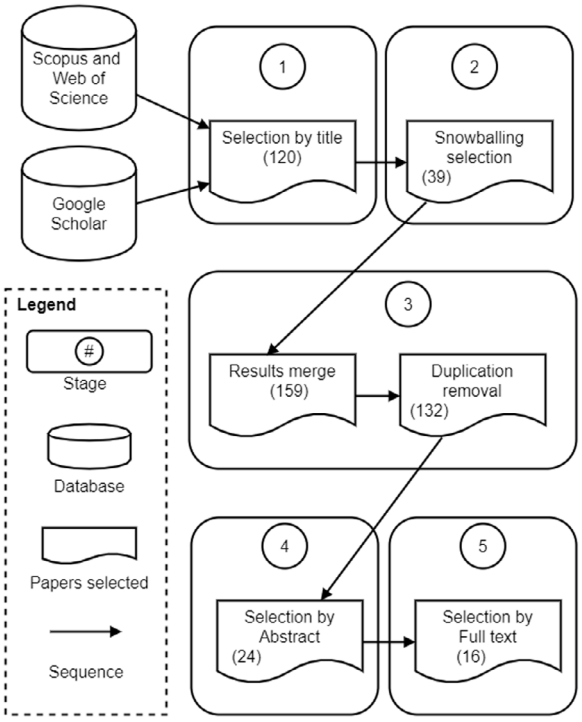

Fig. 3. Document search and selection stages.

Fig. 4. Word-cloud associated with the results of the search strategy.

make it or did not give information about it (No). The *Specification and analysis* stage defines the application domain (dependent or independent of the context), the quality attributes established in the designed architecture or framework, and identifies how the requirements of the proposed solution were specified. The *Design* stage identifies in which CPS architecture layer the articles are focusing (Cyber, Network, or Physical), the architecture style, and the self-adaptation technique used. The *Implementation* stage identifies the cyber and physical approaches used by the reviewed articles, differentiating the articles that used any of the listed layers and the articles that used another type of layer. The *Integration* stage differentiates the articles that integrated the physical and cyber layers (Yes) and the articles that did not make it or did not give information about it (No). The *Quality control* stage identifies how articles measure the quality of the system. In this sense, quality control differentiates the articles that implemented any unit test and integration test (Yes) and the articles that did not make it or did not give information about it (No). The quality control stage differentiates the strategies used to execute the system tests, and the attributes tested to achieve the acceptance tests. The *Maintenance* stage identifies the articles that implemented any maintenance strategy (Yes) and the articles that did not make it or did not give information about it (No).

3.4. Threats to validity

There are four threats to the validity of this research. The strategies to minimize such threats are explained in what follows. Table 4 lists some biases that were taken into account.

Construction validity: To limit construction threats, several measures were taken: (i) the search strategy and the search process used were guided by a well-known methodology proposed in the literature. (ii) in the construction of the search strings, different terms that could relate to CPSs were taken into consideration, and (iii) the research questions were answered according to a categorization scheme, defined in Table 3, through a cross-check process among the authors.

Internal validity: We searched three online digital libraries, including *Google Scholar*. These libraries cover the majority of high-quality publications in the field, but the lack of more libraries may lead to a bias in the identification of primary studies (Zhou et al., 2016). Furthermore, we used snowballing (Wohlin, 2014) as a complementary search strategy to reduce the possibility of missing relevant articles. Also, the search strategy was developed and reviewed by all the authors. Finally, an explicit statement of the methods used for the research review is described for the reader to make an informed assessment of the scientific rigor of the review and the strength of the review's inferences.

External validity: All the articles taken into consideration were selected based on their relevance to the SA-CPSs domain. The excluded papers may affect the generalizability of our results. This threat is minimized by the reliability of our research process because it is a systematic process that allows replication (proposed by Li et al. (2015)).

Conclusion validity: To avoid bias in the data extraction process, the data was extracted through a cross-check among the authors. A cross-check minimizes different interpretations of the data and subjective judgment.

4. Results

This section presents the results divided into each stage of the SDLC (see a summary in Table 7).

4.1. Planning

The articles do not show –in detail– how the planning stage was developed in their proposals because it is not usual to present this type of information in research articles. Nonetheless, it was identified the following information, for each planning activity, following the guidelines of Dennis et al. (2014).

Identification of the opportunity: In most of the articles, the identification of the opportunity to develop the system was made through a literature review, where challenges associated with the design of CPS and IoT systems were addressed, such as networking settings (Iftikhar et al., 2017), dynamic environment (Lee et al., 2019), domain dependency (Park and Park, 2019), and deployment issues (Alkhabbas et al., 2020).

Table 3
SDLC characteristics to be analyzed.

Stage	Characteristics	Sub-characteristic
Planning	Planning strategy	Yes No
Specification and analysis	Application domain	Dependent Independent
	Quality attributes	Performance Scalability Energy-efficiency Reliability Maintainability Security Interoperability Usability
	Requirement specification	System modeling (UML) Natural language Mathematical specification
Design	Architecture Layer	Cyber Network Physical
	Architecture style	Cloud-service Client–server Edge computing Layered N/A
	Self-adaptation technique	MAPE-K Agents Others
Implementation	Cyber layer	Web-application Web-service Component-based Other
	Physical layer	Physical components Model representation
Integration	Integration	Yes; No
Quality control	Unit testing	Yes; No
	Integration testing	Yes No
	System tests	Simulation Small-scale Real case
	Acceptance tests	Adaptability Scalability Performance Energy-efficiency Reliability
Maintenance	Maintenance strategy	Yes No

Table 4
Overview of potential biases in the SMS process, based on Cooper (2010), Felson (1992), Janssen (2018), Zhou et al. (2016).

Bias types	Description	Solutions
Publication bias	Tendency to selectively publish some articles over others (e.g. only significant effects or large studies).	Researchers cross-check the completeness of searches and validate the suitability of each study for inclusion.
Location bias	Tendency to select studies that are only indexed in electronic databases.	Google Scholar was used as a source of non-indexed articles.
Language bias	Tendency to exclusively select studies based on any language.	English was the dominant research language in the studies from all databases.
Citation bias	Tendency to select studies that may be relevant based in the citation results, this may produce a biased sample of studies.	Studies selection is not based on the citation number avoiding this type of bias. The selection was based on the inclusion and exclusion criteria.
Study selection bias	It means some errors (e.g., related studies are not chosen or irrelevant, poor quality studies or only positive papers are chosen), which may be found in the search process (Zhou et al., 2016).	A rigorous search strategy was defined and applied. Also, to mitigate misinterpretations title/abstract, introduction and conclusions were read before rejecting or accepting a paper.

Economic-feasibility evaluation: Articles, in general, do not discuss development or operational costs, such as consultant fees, hardware repair, software upgrades, user training, software-licensing fees, and return on investment (Dennis et al., 2014). An exception is the research of Trihinas et al. (2018) that defines that their framework is based on a low-cost adaptive and learning model. Intangible benefits the system will have were identified as a higher-quality of the product –directly– associated with quality attributes (International Organization for Standardization, 2011a) (also called nonfunctional requirements), defined in the system's architecture.

Organizational-feasibility evaluation: From an organizational perspective, a feasibility evaluation –such as system acceptance by the users, and incorporation in the organization or context– is not discussed in the revised articles.

Technical-feasibility evaluation: From a technical perspective, risks associated with the technology are not discussed.

In this stage, self-adaptation was defined as an opportunity to detect context changes from unintentional behaviors within the physical world to provide appropriate services, enabling a more reliable process execution. Sustainability –from the technical perspective– although was not explicitly mentioned, it is directly associated with the intangible benefits planned. Sustainability –from the economic perspective– was not mentioned, except Trihinas et al. (2018) that related strategic planning with the use of low-cost techniques to build inexpensive solutions, addressing economical sustainability.

4.2. Specification and analysis

The identification of the *application domain* is a part of the requirements elicitation in the specification and analysis stage (Sommerville, 2015). In most of the articles, the application domain is independent of the context, which means that the proposed SA-CPSs architectures or frameworks have the full potential to cover diverse domains, such as smart cities, smart agriculture, and smart homes. The exception is the work of Provoost et al.'s work (Provoost and Weyns, 2019), which presented the *Dingnet architecture* –where mobile embedded systems (i) move in a city area, (ii) adapt their network settings to ensure reliable and energy-efficient communication, and (iii) support the design and evaluation only for the smart-city application domain.

In the requirements elicitation, the functional and quality attributes were gathered from the literature, surveys, tools, or personal experience. Most articles included the *self-adaptability requirement*, one of the search criteria of this SMS. This requirement is understood in CPSs as the ability to modify their behavior and/or structure in response to changes in their environment and user requirements (De Lemos et al., 2013; Weyns and Georgeff, 2010). The quality attributes can be grouped into nine characteristics: (i) functional suitability, (ii) reliability, (iii) performance, (iv) efficiency, (v) usability, (vi) security, (vi) compatibility, (viii) maintainability, and (ix) portability, according to ISO/IEC 25010:2011 (International Organization for Standardization, 2011a). Each characteristic is composed of a set of related sub-characteristics. Quality attributes, in the reviewed articles, were very varied, as it is shown in Table 5 , "Addressed" column.

Eleven articles specified *performance*, where latency, throughput, overhead, and CPU-cycles consumed were established to measure it. Four articles specified the *energy efficiency*, which is part of the performance attribute, which –usually– is defined in *IoT devices* to expand battery life in devices with intense processing that increases energy consumption (Xiao et al., 2010).

Four articles specified the *scalability*, which is part of the adaptability attribute, and refers to the scalability of the internal capacity that can be vertical, where hardware and software capacity is increased by adding resources, or horizontal, where more nodes, such as servers or computers, are added to work as a single logical unit (Rouse, 2007).

Five articles specified the *reliability*, focusing on the number of packets lost, the capability to recover after a failure, automated error handling, and service accuracy.

Four articles specified *maintainability*, whose definition is presented in Section 2.2. For the reusability sub-characteristic of maintainability, articles focused –primarily– on the reusability of the device-level functionality and components.

Finally, two articles focused on *security* and one article in *interoperability*.

Quality attributes are documented in a process called requirements specification, where requirements can be represented in natural language, structured language, graphical notations such as *Unified Modeling Language (UML)* diagrams, and mathematical specifications (Sommerville, 2015). In the reviewed articles, three types of specifications of the architecture or framework were found: (i) Natural language, (ii) System modeling, and (iii) Mathematical specification.

Natural language: All articles used natural language to explain the requirements that the proposal will have, and some complemented with an overview figure, such as Park and Park (2019), Lee et al. (2019), Cui et al. (2013), Iftikhar et al. (2017), Alkhabbas et al. (2020), Bedhief et al. (2019), Camara et al. (2020).

System modeling: The articles (Park and Park, 2019; Seiger et al., 2019; do Nascimento and de Lucena, 2017; Gerostathopoulos et al., 2019; Provoost and Weyns, 2019; Torres et al., 2017; Alkhabbas et al., 2020; Ramesh Babu and Mohana Roopa, 2017) used UML diagrams, such as component diagrams, class diagrams, sequence diagrams and activity diagrams.

Mathematical specification: The articles (Trihinas et al., 2018; Lee et al., 2019) used mathematical specifications, such as mathematical concepts and finite-state machines.

In this stage, self-adaptation was specified as the main requirement of the proposed solutions. Sustainability –from the technical perspective– is implicit in the specification of maintainability, scalability, and security attributes. From the economic perspective, explicit information was not found, but economic sustainability can be implicitly associated with the performance and energy-efficient attributes since it assists the operation costs and energy costs, respectively.

4.3. Design

Reviewed articles focused on one or more layers of the CPSs architecture (cyber, network, physical) (Zeadally et al., 2019b). In most of the reviewed articles, the focus is on the cyber layer. In the articles (Trihinas et al., 2018; Cui et al., 2013), the framework or architecture can be applied to the physical layer, such as sensors, actuators, and controllers. In Iftikhar et al. (2017), Provoost and Weyns (2019), Torres et al. (2017), Bedhief et al. (2019), the network-layer design is reflected through the management of issues such as packet losses, delays, and network topology changes.

In Torres et al. (2017), Alkhabbas et al. (2020), a client–server architecture is defined where users or devices access servers to use services. Authors of Park and Park (2019), Alkhabbas et al. (2020), Camara et al. (2020) defined a cloud architecture where the software components and services are distributed across the cloud. In Trihinas et al. (2018), Iftikhar et al. (2017), Alkhabbas et al. (2020), Bedhief et al. (2019), an edge-computing architecture is used where software components and embedded systems are placed in networks, and processing and data dissemination are over the network. In Cui et al. (2013), Camara et al. (2020), a layered architecture is used to support scalability.

Table 5
Quality attributes addressed and tested in the reviewed articles.

Quality attributes	Addressed	Tested
Performance	Park and Park (2019), Seiger et al. (2019), D'Angelo et al. (2018), Trihinas et al. (2018), Lee et al. (2019), Gerostathopoulos et al. (2019), Cui et al. (2013), Alkhabbas et al. (2020), Bedhief et al. (2019) and Ramesh Babu and Mohana Roopa (2017)	Park and Park (2019), Seiger et al. (2019), Trihinas et al. (2018), Lee et al. (2019), Gerostathopoulos et al. (2019), Cui et al. (2013), Alkhabbas et al. (2020), Bedhief et al. (2019) and Ramesh Babu and Mohana Roopa (2017)
Energy efficiency	Trihinas et al. (2018), Iftikhar et al. (2017), Provoost and Weyns (2019) and Camara et al. (2020)	Trihinas et al. (2018), Iftikhar et al. (2017), Provoost and Weyns (2019) and Camara et al. (2020)
Scalability	Horizontal - Park and Park (2019), Trihinas et al. (2018), Alkhabbas et al. (2020), Bedhief et al. (2019) and Camara et al. (2020) Vertical -Camara et al. (2020)	Park and Park (2019), Trihinas et al. (2018), Alkhabbas et al. (2020) and Bedhief et al. (2019)
Reliability	Seiger et al. (2019), Iftikhar et al. (2017), Provoost and Weyns (2019), Bedhief et al. (2019), Camara et al. (2020) and D'Angelo et al. (2018)	Seiger et al. (2019), Iftikhar et al. (2017), Provoost and Weyns (2019), Bedhief et al. (2019) and Camara et al. (2020)
Maintainability	Park and Park (2019), Seiger et al. (2019), Cui et al. (2013) and Ramesh Babu and Mohana Roopa (2017)	
Security	Provoost and Weyns (2019) and Torres et al. (2017)	
Interoperability	do Nascimento and de Lucena (2017)	

In D'Angelo et al. (2018), do Nascimento and de Lucena (2017), Lee et al. (2019), Gerostathopoulos et al. (2019), Kit et al. (2015), Provoost and Weyns (2019), Seiger et al. (2019), Ramesh Babu and Mohana Roopa (2017,?), a specific architecture is not raised because they implement their solution as a software component or software project.

MAPE-K is the dominant self-adaptation technique in the design of CPSs. The articles (Seiger et al., 2019; D'Angelo et al., 2018; Gerostathopoulos et al., 2019; Kit et al., 2015; Iftikhar et al., 2017; Torres et al., 2017; Alkhabbas et al., 2020; Camara et al., 2020; Park and Park, 2019; Lee et al., 2019; Ramesh Babu and Mohana Roopa, 2017) used this technique. Provoost and Weyns (2019) used a simple feedback loop. Bedhief et al. (2019) used an autonomous manager to adapt a network to an on-demand application based on the available resources. do Nascimento and de Lucena (2017) used adaptive agents that make decisions on a controller, which could be a finite-state machine or a machine-learning technique. Trihinas et al. (2018) used probabilistic-learning algorithms to reduce data volume and network traffic between IoT devices and cloud services: The framework created is embeddable in the core software of IoT devices.

It was identified that most of the articles focused on the *autonomic-computing paradigm* use the MAPE-K feedback loop.

Three articles mentioned paradigms such as *context-aware*, which is a feature of self-adaptation where intelligent systems detect context changes and react based on their environment.

Three articles mentioned *machine-learning (ML) paradigm*, where ML techniques are used to perform adaptations. One article addressed the *process-aware paradigm*, where automated processes in CPSs require that the effects of the processes in the environment and the context are considered (Wombacher, 2011).

One article addressed the *model-driven engineering paradigm* (D'Angelo et al., 2018), where models are used to engineer SA-CPSs and exploited in all stages of the SDLC. One article addressed the *goal-driven paradigm*, where a set of devices with individual functionalities connect and cooperate temporally to achieve the user goal (Alkhabbas et al., 2020).

One article addressed the *MAS paradigm*, to model real-world systems and managing large and distributed-information systems.

One article addressed the *fog-computing paradigm*, to respond to the requirements in terms of reliability, delay, and scalability (Sanchez et al., 2017).

Table 6 presents a summary of paradigms used in the design stage, identified by the authors of this SMS.

At this stage, self-adaptation is achieved through adaptation techniques, mainly, the MAPE-K feedback loop. Sustainability –from the technical perspective– is associated with the design of software components that can be reused, aiming at the maintainability attribute. Technical sustainability is also associated with the use of layered and cloud-based architecture that allows scalability. Sustainability –from the economic perspective– is associated with the design of algorithms that reduce costs in analysis, data collection and energy consumption.

4.4. Implementation

This section focuses on how the cyber and physical layers were implemented for this type of system, and what technology decisions were taken.

In the *cyber layer*, most of the reviewed articles implemented their proposals as software components or software projects. The articles (D'Angelo et al., 2018; do Nascimento and de Lucena, 2017; Lee et al., 2019; Gerostathopoulos et al., 2019; Kit et al., 2015; Provoost and Weyns, 2019; Seiger et al., 2019) proposed solutions implemented as Java-projects, mostly using Eclipse IDE. Also, some of the software projects are available in the GitHub repository: A great advantage of these projects is that they can be downloaded and used to experiment in different application domains. Park and Park (2019) implemented a web-application where developers can easily (i) upload implemented components, (ii) search existing components, (iii) register participant systems/devices, and (iv) launch virtual machines. This implementation strategy allows developers to focus on implementing IoT collaboration services without caring about the type of participating devices. Also, Torres et al. (2017) implemented a web-application to monitor and request data from sensors as temperature and humidity. The works of Seiger et al. (2019), Iftikhar et al. (2017), Camara et al. (2020) implemented a web-service to be used by external entities, such as services and applications. In Trihinas et al. (2018), a monitoring framework was used in a server. In Alkhabbas et al. (2020), a simulator platform was used in a server. In Bedhief et al. (2019), a network emulator was used in virtual machines.

In the *physical layer*, the articles (Park and Park, 2019; Seiger et al., 2019; Trihinas et al., 2018; Lee et al., 2019; Cui et al., 2013; Iftikhar et al., 2017; Torres et al., 2017) used physical components (e.g., sensors, actuators, and controllers) in their implementations, such as Arduino, Raspberry, smartphones, temperature, and humidity sensors. In D'Angelo et al. (2018), do Nascimento and de Lucena (2017), Lee et al. (2019), Gerostathopoulos et al. (2019), Kit et al. (2015), Iftikhar et al. (2017), Provoost and Weyns (2019), Alkhabbas et al. (2020), Bedhief et al. (2019), Camara et al. (2020), Ramesh Babu and Mohana Roopa (2017), a model representation was used in platforms that simulate the behavior of physical components.

The communication between these layers was made through the network with the use of *hypertext transfer protocol (HTTP)* and

Table 6
Paradigms used in the design of self-adaptive CPSs.

Paradigm	Articles
Autonomic computing	Park and Park (2019), Seiger et al. (2019), D'Angelo et al. (2018), Lee et al. (2019), Gerostathopoulos et al. (2019), Kit et al. (2015), Iftikhar et al. (2017), Torres et al. (2017), Alkhabbas et al. (2020), Camara et al. (2020), Ramesh Babu and Mohana Roopa (2017)
Context-aware	Park and Park (2019), Ramesh Babu and Mohana Roopa (2017), Bedhief et al. (2019)
Machine learning	Camara et al. (2020), Trihinas et al. (2018), do Nascimento and de Lucena (2017)
Process-aware	Seiger et al. (2019)
Model-driven engineering	D'Angelo et al. (2018)
Goal-driven	Alkhabbas et al. (2020)
MAS	do Nascimento and de Lucena (2017)
Fog computing	Bedhief et al. (2019)

representational state transfer (REST) standards, but, especially, a publish–subscribe network protocol, the *message queue telemetry transport (MQTT)* broker was –widely– used.

In this stage, self-adaptation was implemented, mainly, in the cyber layer. Sustainability –from the technical and economical perspectives– complies with what is stated in the design stage.

4.5. Integration

In the reviewed articles, it was not explicit the integration process, but it was common to find the "testbed" concept where the authors showed how a solution design (or proof-of-concept) of the proposal would be. A testbed includes an environment with (i) tools, (ii) software components, (iii) servers, (iv) network components, (v) physical devices, and (vi) their communication. Therewith, the authors demonstrate, in a prototype, in some cases, the integration of the software layers and physical objects (Park and Park, 2019; Seiger et al., 2019; do Nascimento and de Lucena, 2017; Trihinas et al., 2018; Lee et al., 2019; Cui et al., 2013; Iftikhar et al., 2017; Torres et al., 2017). When physical devices are not used, the system integration is made by combining (i) software packages and libraries (Camara et al., 2020), (ii) simulation platforms for CPSs (Gerostathopoulos et al., 2019), (iii) network simulators (Kit et al., 2015), (iv) servers, and (v) virtual machines (Alkhabbas et al., 2020; Bedhief et al., 2019), so that they can be treated as a unit.

This stage did not provide much information about self-adaptation, nor sustainability from neither the technical nor economic perspectives.

4.6. Quality control

The authors of the reviewed articles tested their proposals in –usually– one specific application domain. do Nascimento and de Lucena (2017), Trihinas et al. (2018) tested their proposals in two or more application domains. The most used application domains for testing were smart cities, smart homes, smart public security, smart power grid, smart devices, streaming services, and smart greenhouses.

Unit-testing or component-testing involves verifying that each unit meets its specification. In the reviewed articles, there is no evidence of component tests in isolation from the rest of the system. This type of testing is considered important because (i) it allows verifying whether the functional and non-functional behaviors of the component are as designed and specified, (ii) it helps to reduce risks, (iii) it builds confidence in the component's quality, (iv) it finds defects in the component and (v) it prevents defects (ISTQB, 2018).

The components were combined to integrate systems and to ensure that the system works properly –focusing on the flow control and data exchanged among objects– (Dennis et al., 2014), and to detect defects in the interfaces and the interactions among them, known as integration testing. In the reviewed articles, *end-to-end (E2E)* integration testing –wherein it is verified that a defined set of interconnected systems will perform correctly (Tsai et al., 2001)– was used in Park and Park (2019), D'Angelo et al. (2018), do Nascimento and de Lucena (2017), Trihinas et al. (2018), Gerostathopoulos et al. (2019), Bedhief et al. (2019), Camara et al. (2020).

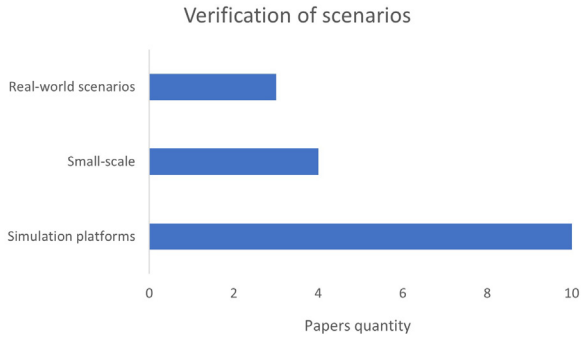

Fig. 5. Strategies used to execute and verify the scenarios.

Almost all articles defined at least one scenario to be tested, as in Park and Park (2019), Seiger et al. (2019), do Nascimento and de Lucena (2017), Trihinas et al. (2018), Lee et al. (2019), Gerostathopoulos et al. (2019), Provoost and Weyns (2019), Torres et al. (2017), Bedhief et al. (2019), Camara et al. (2020), which should be examined through system testing. System testing can be performed in different ways. In the reviewed articles, common strategies to execute and verify scenarios defined for CPSs are the following (see a summary in Fig. 5).

Simulation platforms are used for complex and complicated systems, such as CPSs, IoT, and multi-agent systems. In Park and Park (2019), D'Angelo et al. (2018), do Nascimento and de Lucena (2017), Lee et al. (2019), Gerostathopoulos et al. (2019), Kit et al. (2015), Iftikhar et al. (2017), Provoost and Weyns (2019), Alkhabbas et al. (2020), Camara et al. (2020), the validation of their design was made through simulations of physical models (representation of real devices), scenarios, and environments.

Small-scale or also called prototyping, is where the scenario to test is constructed in small size and limited in extent. For Park and Park (2019), do Nascimento and de Lucena (2017), Lee et al. (2019), Bedhief et al. (2019), system testing was made through a small-scale scenario.

Real-world scenarios, where real-world objects are used in real environments. In Seiger et al. (2019), Trihinas et al. (2018), Torres et al. (2017), system testing was executed in real-world scenarios.

Acceptance criteria is a kind of acceptance testing associated with the system's requirements –such as functional and quality attributes– defined in the specification and analysis stage, are verified. Acceptance testing is used to ensure that the behaviors of the system are as specified (ISTQB, 2018). In the reviewed articles, maintainability, security, and interoperability were requirements

–specified in the Specification stage– but it was not found evidence of the validation of these attributes in the quality-control stage.

Few articles, such as D'Angelo et al. (2018), Lee et al. (2019), Camara et al. (2020), did not test performance and scalability attributes due to the scope or limitations of the research. They reported that, in future investigations, these attributes will be evaluated. The quality attributes evaluated, in each article, for the quality-control stage, are listed in Table 5, "Tested" column.

In this stage, self-adaptation was the main objective and requirement tested, to demonstrate the feasibility of the proposals. This stage did not provide information about sustainability –from technical and economic perspectives.

4.7. Maintenance

In the reviewed articles, it was identified that the maintenance activities are not explicitly mentioned, but –implicitly– they were performed since they are presented as future research. Maintenance could be associated with adaptive maintenance-planning activities explained as follows.

Planning to improve the implementation of the system: As examples, Seiger et al. (2019) used alternative algorithms in the Analyze-and-Plan stages of the MAPE-K feedback loop to reduce modeling effort and increase autonomy. Lee et al. (2019) proposed to improve the decision-making method using ML techniques. Bedhief et al. (2019) proposed to use artificial intelligence and ML methodologies to improve the autonomous manager. Provoost and Weyns (2019) proposed to enhance the simulator using collected data from an experimental context, to bring it closer to a real setting.

Planning to add functionality and new features: Some works plan to extend their researches with new functionalities and features. As an example, Alkhabbas et al. (2020) proposed to enable scalability to support large-scale IoT environments, and to extend the approach to consider energy consumption, privacy, security, and cost reliable deployment topologies for *Goal-Driven IoT Systems (GDSs)*. D'Angelo et al. (2018) proposed to develop a validation technique to detect unintended interactions and to allow modularization and adaptation at run-time.

Planning to continue with the quality-control stage: Some articles plan to test the quality attributes –initially defined in the specification and analysis stage–, but did not reach the quality-control stage. As an example, D'Angelo et al. (2018) proposed to evaluate the performance and scalability of the tool. Camara et al. (2020) proposed to explore scalability and performance measures, such as response time, use, and throughput, as well as the trade-offs and robustness of their approach.

Planning to expand the application of the approach: Authors of Park and Park (2019), do Nascimento and de Lucena (2017) proposed to apply the proposed solution in various domains to increase domain coverage.

For self-adaptation, it is notable that the researchers plan to improve the implementation of the MAPE-K feedback loop with the use of alternative algorithms, such as ML techniques. For sustainability –from the technical and economic perspectives– explicit information was not found in this stage.

5. Discussion

This section presents the most important trends and limitations concerning the results from Section 4. Trends and limitations are divided into the stages of the SDLC (see a summary of limitations in Table 8 and a representation of trends in Fig. 6[1])

[1] An interactive demo of this figure can be downloaded from https://github.com/LuisaRestrepo/Sustainable-SA-CPSs, where there is more information about this study.

Planning

At the planning stage, all articles defined the motivation and identified the opportunity to be addressed. Everyone used at least one planning strategy, as shown in Fig. 6. No trends were identified at this stage.

A weakness observed at this stage is the *lack of a better specification about economic aspects such as development costs and revenues.* This specification would serve to help future researchers to (i) identify the economic feasibility to replicate or to use the proposed design, (ii) identify the viability in project planning and technology adoption, and (iii) know the relevance of the design to the business contexts. Since the focus of the reviewed articles is on intangible benefits, sustainability –from the technical and economic perspective– was not evaluated.

Specification and analysis

In the specification and analysis stage, almost all articles defined that their design can be applied to any application domain. The first trend is the creation of designs that are not limited to a particular application domain and the implementation of generic solutions that allow using diverse devices. Second, as it is shown in Fig. 6, performance is the most commonly used attribute when designing SA-CPSs, which coincides with the SLRs of Muccini et al. (2016) and Musil et al. (2017). Third, attributes such as energy-efficiency, scalability, and reliability are widely used. Finally, natural language is the most used technique to specify system requirements –sometimes– accompanied by UML diagrams (usually, component diagrams).

Self-adaptability is allowing systems to deal with uncertainties, resulting in a high capacity of maintenance aiming at the sustainability of the systems from the technical perspective. Sustainability has also been linked with the scalability attribute. From the environmental perspective, sustainability has been addressed in the energy-efficient attribute associated with performance, and to reduce the environmental impact by reducing the consumption of energy of the physical devices. Note that there are trade-offs between performance and energy consumption.

The main weakness identified in this stage is that the quality attributes associated with the self-adaption and sustainability –such as security, interoperability, usability, modularity, modifiability, compatibility, testability– are not being considered or specified in a detailed way. Furthermore, in some cases, it is not clear how the defined quality attributes will be measured.

Design

For the design stage, the most common trend is to use the MAPE-K feedback loop, which is consistent with the SLR of Muccini et al. (2016). MAPE-K is applied –mainly– to the cyber layer, with the implementation of software components. Besides, very few articles used ML approaches to enhance MAPE-K components (Camara et al., 2020; Trihinas et al., 2018; do Nascimento and de Lucena, 2017).

The weakness observed at this stage is that few articles used paradigms, such as cloud computing, edge computing and fog computing. As the focus of most articles is on the cyber layer, there is a *lack of designs that include the physical layer*, then special interfaces or protocols for certain devices have not been considered. This could imply that the cyber layer may not correctly accept inputs (e.g., data, control, and parameters) from the physical layer, or may send incorrect outputs to the physical layer.

Table 7
Summary of the results.

Stage	Q1 - Q7: Methods and strategies used	Q8 - Sustainability dimensions	
		Technical	Economical
Planning	(Q1) Identification of the opportunity in the literature. (Q1) Identification of intangible benefits associated with higher-quality attributes.	Implicitly associated with the identification of intangible benefits of the product.	Planning to use low-cost techniques.
Specification and analysis	(Q2.1) Application domain independent of the context (Q2.2) Performance, energy-efficiency, scalability, reliability, maintainability, security, and interoperability, as the specified quality attributes. (Q2.3) Natural language, system modeling, and mathematical specification as the most used specification techniques.	Implicit in the specification of maintainability, scalability, and security attributes.	Implicitly associated with the performance and energy-efficient attributes
Design	(Q3.1) Self-adaptation is achieved through adaptation techniques such as MAPE-k, adaptative agents, probabilistic-learning algorithms, and autonomous managers. (Q3.2) The use of architectural styles such as cloud, client–server, edge computing, and layered.	Design of software components that can be reused (Maintainability). The use of layered and cloud-based architecture (Scalability).	Design of algorithms that reduce costs in analysis, data collection, and energy consumption.
Implementation	(Q4.1) Self-adaptation was implemented, mainly, in the cyber layer as Java-projects. (Q4.2) The cyber layer was implemented through software components, web-application, web service, and the use of servers and simulator platforms. (Q4.3) The physical layer was implemented mostly as a model representation, a few with the use of physical components. (Q4.4) The network layer was implemented through the use of MQTT, REST, and HTTP protocols.	It complies with what is stated in the design stage.	It complies with what is stated in the design stage.
Integration	(Q5) System integration was made by combining software packages and libraries, simulation platforms for CPSs, network simulators, servers, virtual machines, and physical devices.	No information	No information
Quality control	(Q6.1) Most proposals were tested in one specific application domain. (Q6.2) Simulation platforms, small-scale, real-world scenarios were the validation techniques used to test the scenarios. (Q6.3) Performance, energy-efficiency, scalability, and reliability were the verified quality attributes.	No information	No information
Maintenance	(Q7) Plan to improve the implementation of the MAPE-K feedback loop. (Q7) Planning to add functionality and new features. (Q7) Planning to continue with the quality-control stage. (Q7) Planning to expand the application of the approach.	No information	No information

Implementation

In the implementation stage, the trend is to implement the proposed solutions as component-based and to use model representations for physical components.

The weaknesses observed at this stage are two. First, although the use of component-based approaches, attributes crucial for self-adaptation –such as modularity and reusability– were not directly mentioned and, therefore, it is not established how they will be measured. Second, the model-representation techniques have disadvantages associated with the use of physical-model representations, since physical aspects such as battery life, data transmission, and failures are not taken into account.

Integration

In the integration stage, it is identified that almost all articles integrated their solutions with software systems and a few with physical components. A weakness observed at this stage is the

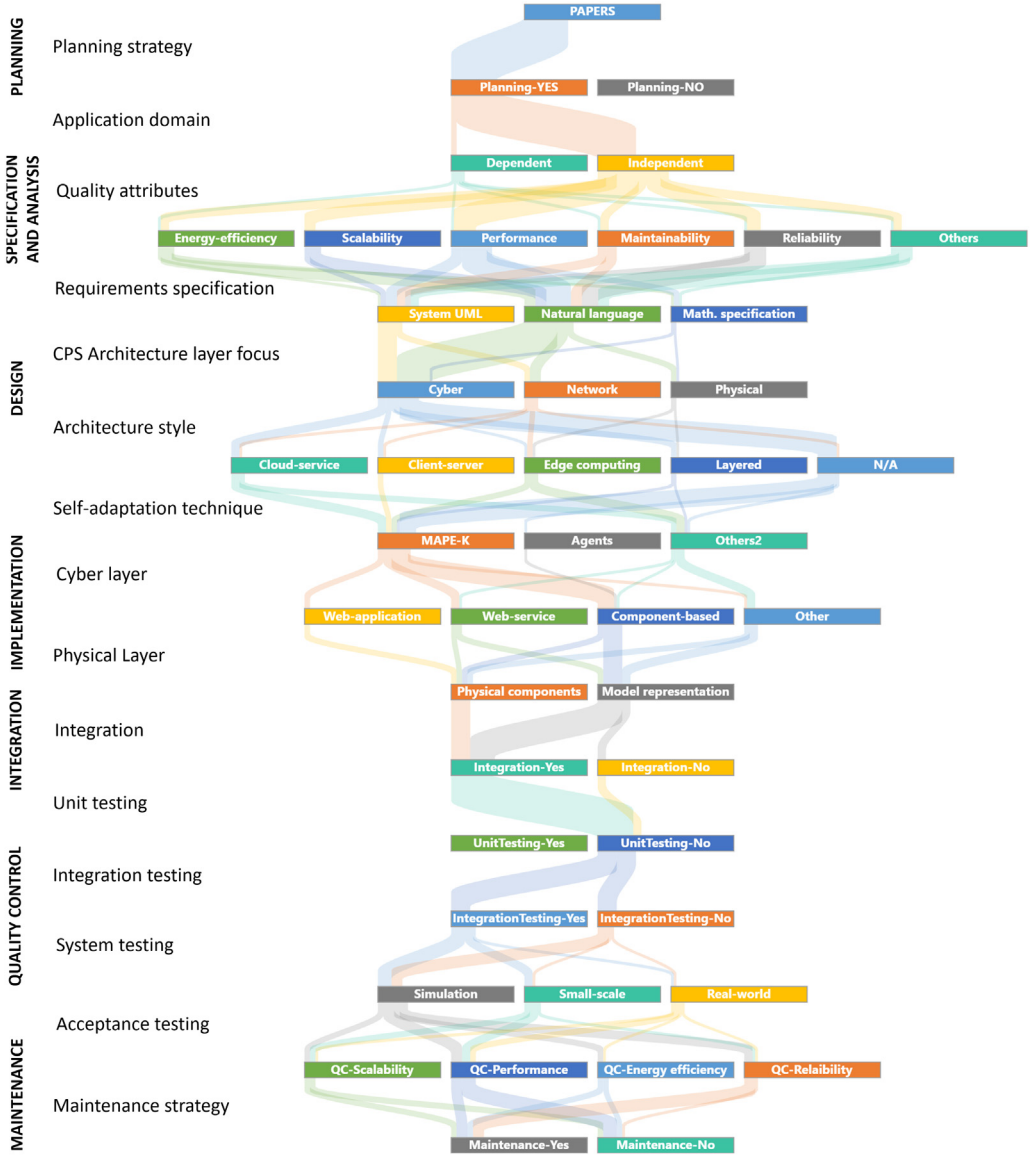

Fig. 6. Trends of the selected articles for designing self-adaptive CPSs.

lack of information on how the components were joined as one large system.

Quality control

In the quality-control stage, a trend –observed in Fig. 6– is to simulate the scenarios to evaluate and to achieve system testing. To achieve acceptance testing, the most tested quality attributes were scalability, performance, and energy-efficiency.

The main weaknesses observed at this stage are that there is a *lack of information on unit testing (or component testing)* and a *lack of integrated testing of the subsystems that together compose the system*. Only a general environment of tests was achieved. Frameworks and architectures have been tested in a few application domains; in most of them, they were simulated. There is a need to verify the real-world effects and the real-time behavior of the proposed solutions. Finally, more experimental tests are needed to measure quality attributes –crucial for self-adaptation– such as security, usability, testability, maintainability, and interoperability of the proposed designs.

Maintenance

In the maintenance stage, it is clear that –with the use of the autonomic paradigm– systems will maintain and adjust their operation for situations like failures in the software or hardware, or changes in the components at run-time. Nonetheless, no explicit information was found about the activities carried out to achieve technical maintenance to increase the useful life and reliability of the systems. It was identified maintenance activities –such as planning– to improve and enhance the proposed designs in future research. For this reason, a weakness observed at this stage is the *lack of information on how technical maintenance can be achieved*.

6. Challenges

Given the discussion of the trends and weaknesses, the following opportunities for future research were found.

Competitiveness on the market

Competitiveness on the market is an extremely crucial issue due to the high-volume of embedded systems in the market (Marwedel and Engel, 2016). To affront this demand, the creation

Table 8
Summary of limitations and weaknesses.

Stage	Limitation/Weakness
Planning	Lack of a better specification about economic aspects as development costs and revenues.
Specification and analysis	Quality attributes associated with the self-adaption and sustainability are not being considered or specified in a detailed way.
Design	Lack of designs that include the physical layer
Implementation	Attributes such as were not directly mentioned and it is not established how they will be measured. The use of physical-model representations techniques.
Integration	Lack of information on how the components were joined as one large system.
Quality control	Lack of information on unit testing (or component testing). Lack of integrated testing of the subsystems that together compose the system. Testing in few application domains. Simulated environments.
Maintenance	Lack of information on how technical maintenance can be achieved.

of low-cost CPSs should take into account the efficient use of hardware and software budget and a cost–benefit analysis. Further investigations are needed to identify these tangible benefits associated with the development of SA-CPSs to reach sustainability from an economic perspective, and strategies to implement low-cost SA-CPSs. Examples of such tangible benefits applied to software systems are (i) sales growth, and (ii) reduction in information-technology costs, staff, and inventory. A challenge is how to generalize aspects such as (i) sales growth, and (ii) reduction in information-technology costs, staff, and inventory (Dennis et al., 2014) to CPSs because there is no sale-and-cost history or staff-and-inventory records.

Hybrid approaches for feedback loops

The use of hybrid approaches that combine cloud computing, edge computing, and fog computing for deployment, with the MAPE-K model, adaptive agents, or other feedback loop mechanisms, may provide a suitable approach to mitigate the issues of scalability, fault-tolerance, performance, and flexibility of SA-CPS. Examples of articles in which such approaches mitigated these issues are Kumar and Hanumanthappa (2013), Kang and Yu (2018), Wang et al. (2019).

Maintenance of quality levels

The execution of self-adaptation activities –in SA-CPSs– can affect the levels of quality attributes established. Strategies are needed to allow maintaining the quality levels of the proposed designs. Examples of such strategies –in software systems– are a continuous quality-monitoring platform to understand when the quality is decreasing (Janes et al., 0000), test automation, and the establishment of metrics to continuously measure quality. A challenge is how to generalize these aspects for CPSs because software systems do not consider the physical layer.

Conceptualization of sustainability

System architectures are a major driver for sustainability (Koziolek, 2011), therefore, it is important to know *how to specify and evaluate sustainability in CPSs to construct robust and cost-effective systems*. A starting point would be to establish a unified vision of what sustainability is, by building an ontology, a soft-goal model, or a *non-functional requirement (NFR)* framework (Chung et al., 2000). Recent work on decision maps for sustainability provides such an (initial) framework; for instance, the work of Lago (2019).

Sustainable framework for CPSs

A framework that allows SA-CPSs to be sustainable from economical and technical perspectives is needed. Examples of similar frameworks are the *Insure framework* to incorporate sustainability –in the software-engineering process– (Saputri and Lee, 2020), and the *SustainPro framework* to implement sustainable designs (Carvalho et al., 2013). A challenge is how to generalize these frameworks for CPSs because existing frameworks do not consider the sustainability of the physical nor network layers.

Specification of heterogeneous devices

CPSs are composed of heterogeneous devices (Romero et al., 2015), so a challenge is how *to cope with the complexity and heterogeneity of requirements to fulfill various scenarios*. For this reason, it is important to identify the techniques to manage requirements and constraints from heterogeneous devices, and methodologies to implement them in SA-CPSs. This has been extensively controlled –for software development– with the use of systematic tools that supports requirement management (Hoffmann et al., 2004). A challenge is how to generalize that for CPSs because existing tools do not consider the different requirements of the physical devices.

Unit and integration testing

Unit and integration testing can be achieved in the development of SA-CPSs. In fact, testbeds were used in the development of SA-CPSs, but the use of methodologies to reduce the number of bugs, the time to find and fix bugs, and to improve the quality of tests were not mentioned. Methodologies used in software engineering –such as Attribute-Driven Design ADD, *test-driven-development (TDD)* (Janzen and Saiedian, 2005) and *continuous integration (CI) practices* (Zhao et al., 2017)– allow decreasing the amount of time it takes to find bugs and to reduce the cost to fix bugs. Thus, a challenge is how to generalize practices such as TDD and (CI) for CPSs because (i) testing methodologies do not consider physical devices and (ii) physical components cannot be continuously improved as software components.

Specification of quality attributes

The quality attributes allow addressing the architecture definition of the systems. A correct specification of quality attributes –such as security, interoperability, modularity, modifiability, compatibility, testability– are needed in the development of SA-CPSs to achieve a complete evaluation of the quality levels. In what follows, we explain challenges related to some of these attributes. *Security:* The enhancement of the security to guarantee the safety and privacy of the users (Hammoudi et al., 2018) is an important factor. The security of SA-CPSs is transformed into sustainability due to the ability to maintain the correct functioning under cyber-attacks. For that reason, it is important to identify and propose techniques that successfully allow maintaining the security of the developed designs.

Maintainability: The easy evolution and ability to change the systems decreases life-cycle costs and managing technical debt (Kruchten et al., 2012). For that reason, it is important to identify

and propose strategies to implement SA-CPSs that successfully allow maintainability and technical sustainability. *Interoperability:* SA-CPSs depend on integration (Song et al., 2016) due to the variety and heterogeneity of devices that have to operate in the environment. It is important to identify interoperability techniques and methodologies that allow the integration of diverse devices and systems (i) across the SDLC and (ii) with different paradigms (e.g., MAS and SOA) to guarantee the delivery of services.

7. Conclusions

This SMS presented general strategies used to design SA-CPSs, at each stage of the SDLC, introduced in Fig. 1. This SMS took into account the sustainability and self-adaptability of the SA-CPSs. Sixteen articles were selected for this SMS that presented a self-adaptive framework or an architecture for SA-CPSs.

This SMS unveiled several trends in the design of SA-CPSs. First, the designs are not limited to particular application domains. Second, performance was the most commonly used attribute. Third, MAPE-K is the predominant feedback loop applied to the cyber layer with the use of complement adaptation strategies. Fourth, the creation of component-based projects for the development of the designs and the simulation of these proposed designs. Fifth, sustainability, in SA-CPSs, has been addressed through self-adaptation, and the use of quality attributes such as adaptability, scalability, energy-efficiency, vaguely, modularity and reusability.

This SMS also identified the absence of information related to the stages of the SDLC. First, the lack of a good specification on economic aspects in the planning stage, especially, tangible benefits. Second, in the specification and analysis stage, the lack of inclusion of quality attributes, such as security, interoperability, modularity, modifiability, compatibility, and testability. Third, in the design and implementation stage, the lack of designs that include the physical layer. Fourth, in the integration stage, the lack of information on how the components were integrated. Fifth, in quality control, the lack of information on unit testing, and the lack of integrated testing of the subsystems. Sixth, in the maintenance stage, the lack of information on how maintenance can be achieved.

Finally, this SMS identified challenges such as (i) How to design and evaluate sustainable SA-CPSs, (ii) How to apply unit and integration testing in the development of SA-CPSs, and (iii) How to develop feedback loops on SA-CPSs with the integration of machine-learning techniques.

CRediT authorship contribution statement

Luisa Restrepo: Conception and design of study, Acquisition of data, Analysis and/or interpretation of data, Writing - original draft. **Jose Aguilar:** Conception and design of study, Analysis and/or interpretation of data, Writing - review & editing. **Mauricio Toro:** Conception and design of study, Analysis and/or interpretation of data, Writing - review & editing. **Elizabeth Suescún:** Conception and design of study, Analysis and/or interpretation of data, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors would like to thank *Vicerectoria de Descubrimiento y Creación* from Universidad EAFIT. This research was supported by Universidad EAFIT, Colombia. The authors would also like to thank David Velasquez for his early comments and suggestions on this research. All authors approved the version of the manuscript to be published.

References used in the review

Alkhabbas, F., Murturi, I., Spalazzese, R., Davidsson, P., Dustdar, S., 2020. A goal-driven approach for deploying self-adaptive IoT systems. In: Proceedings - IEEE 17th International Conference on Software Architecture, ICSA 2020. Institute of Electrical and Electronics Engineers Inc., pp. 146–156. http://dx.doi.org/10.1109/ICSA47634.2020.00022.

Bedhief, I., Foschini, L., Bellavista, P., Kassar, M., Aguili, T., 2019. Toward self-adaptive software defined fog networking architecture for iIoT and industry 4.0. In: IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks, CAMAD. http://dx.doi.org/10.1109/CAMAD.2019.8858499.

Camara, J., Muccini, H., Vaidhyanathan, K., 2020. Quantitative verification-aided machine learning: A tandem approach for architecting self-adaptive IoT systems. In: Proceedings - IEEE 17th International Conference on Software Architecture, ICSA 2020. Institute of Electrical and Electronics Engineers Inc., pp. 11–22. http://dx.doi.org/10.1109/ICSA47634.2020.00010.

Cui, Y., Voyles, R.M., Mahoor, M.H., 2013. ReFrESH: A self-adaptive architecture for autonomous embedded systems. In: IEEE International Conference on Automation Science and Engineering. pp. 850–855. http://dx.doi.org/10.1109/CoASE.2013.6654042.

D'Angelo, M., Napolitano, A., Caporuscio, M., 2018. Cyphef: A model-driven engineering framework for self-adaptive cyber-physical systems. In: Proceedings - International Conference on Software Engineering. pp. 101–104. http://dx.doi.org/10.1145/3183440.3183483.

do Nascimento, N., de Lucena, C., 2017. FIoT: An agent-based framework for self-adaptive and self-organizing applications based on the internet of things. Inform. Sci. 378, 161–176. http://dx.doi.org/10.1016/j.ins.2016.10.031.

Gerostathopoulos, I., Skoda, D., Plasil, F., Bures, T., Knauss, A., 2019. Tuning self-adaptation in cyber-physical systems through architectural homeostasis. J. Syst. Softw. 148, 37–55. http://dx.doi.org/10.1016/j.jss.2018.10.051.

Iftikhar, M.U., Ramachandran, G.S., Bollansée, P., Weyns, D., Hughes, D., 2017. DeltaIoT: A self-adaptive internet of things exemplar. In: Proceedings - 2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2017. Institute of Electrical and Electronics Engineers Inc., pp. 76–82. http://dx.doi.org/10.1109/SEAMS.2017.21, https://www.scopus.com/inward/record.uri?eid=2-s2.0-85025610351&doi=10.1109%2FSEAMS.2017.21&partnerID=40&md5=461a7294434c88ac3df9c02c491702aa.

Kit, M., Gerostathopoulos, I., Bures, T., Hnetynka, P., Plasil, F., 2015. An architecture framework for experimentations with self-adaptive cyber-physical systems. In: Proceedings - 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2015. Institute of Electrical and Electronics Engineers Inc., pp. 93–96. http://dx.doi.org/10.1109/SEAMS.2015.28, https://www.scopus.com/inward/record.uri?eid=2-s2.0-84953218659&doi=10.1109%2fSEAMS.2015.28&partnerID=40&md5=413c60f26d422f5eefa1d03d5d6e9200.

Lee, E., Seo, Y.-D., Kim, Y.-G., 2019. Self-adaptive framework based on MAPE loop for internet of things. Sensors (Switzerland) 19 (13), http://dx.doi.org/10.3390/s19132996.

Park, S., Park, S., 2019. A cloud-based middleware for self-adaptive IoT-collaboration services. Sensors (Switzerland) 19 (20), http://dx.doi.org/10.3390/s19204559.

Provoost, M., Weyns, D., 2019. Dingnet: A self-adaptive internet-of-things exemplar. In: ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems. pp. 195–201. http://dx.doi.org/10.1109/SEAMS.2019.00033.

Ramesh Babu, M., Mohana Roopa, Y., 2017. Component-based self-adaptive middleware architecture for networked embedded systems. Int. J. Appl. Eng. Res. 12 (12), 3029–3034.

Seiger, R., Huber, S., Heisig, P., Aßmann, U., 2019. Toward a framework for self-adaptive workflows in cyber-physical systems. Softw. Syst. Model. 18 (2), 1117–1134. http://dx.doi.org/10.1007/s10270-017-0639-0.

Torres, R., Aros, M., Calderón, J.F., 2017. Towards self-adaptation for cyber-physical systems using a distributed MAPE-k schema over XMPP. In: 2017 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies, CHILECON 2017 - Proceedings. Institute of Electrical and Electronics Engineers Inc., pp. 1–5. http://dx.doi.org/10.1109/CHILECON.2017.8229533, https://www.scopus.com/inward/record.uri?eid=2-s2.0-85043266216&doi=10.1109%2FCHILECON.2017.8229533&partnerID=40&md5=6c03fe10883fbcfd06c450255d1895ba.

Trihinas, D., Pallis, G., Dikaiakos, M., 2018. Low-cost adaptive monitoring techniques for the internet of things. IEEE Trans. Serv. Comput. http://dx.doi.org/10.1109/TSC.2018.2808956.

References

Aguilar, J., Cerrada, M., Mousalli, G., Rivas, F., Hidrobo, F., 2005. A multiagent model for intelligent distributed control systems. In: Khosla, R., Howlett, R.J., Jain, L.C. (Eds.), Knowledge-Based Intelligent Information and Engineering Systems. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 191–197.

Becker, C., Chitchyan, R., Duboc, L., Easterbrook, S., Penzenstadler, B., Seyff, N., Venters, C.C., 2015. Sustainability design and software: The karlskrona manifesto. In: Proceedings - International Conference on Software Engineering. Vol. 2, IEEE Computer Society, pp. 467–476. http://dx.doi.org/10.1109/ICSE.2015.179.

Carvalho, A., Matos, H.A., Gani, R., 2013. Sustainpro-a tool for systematic process analysis, generation and evaluation of sustainable design alternatives. Comput. Chem. Eng. 50, 8–27. http://dx.doi.org/10.1016/j.compchemeng.2012.11.007.

Chantem, T., Guan, N., Liu, D., 2019. Sustainable embedded software and systems. In: Sustainable Computing: Informatics and Systems. Vol. 22, Elsevier Inc., pp. 152–154. http://dx.doi.org/10.1016/j.suscom.2019.05.003.

Chitchyan, R., Groher, I., Noppen, J., 2017. Uncovering sustainability concerns in software product lines. J. Softw.: Evol. Process 29 (2), e1853. http://dx.doi.org/10.1002/smr.1853, http://doi.wiley.com/10.1002/smr.1853.

Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J., Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J., 2000. The NFR framework in action. In: Non-Functional Requirements in Software Engineering. Springer US, pp. 15–45. http://dx.doi.org/10.1007/978-1-4615-5269-7_2, https://link.springer.com/chapter/10.1007/978-1-4615-5269-7_2.

Cooper, H., 2010. Research Synthesis and Meta-Analysis: A Step-By-Step Approach, fourth ed. In: Applied Social Research Methods Series, Sage Publications, Inc, Thousand Oaks, CA, US.

Dafflon, B., Moalla, N., Ouzrout, Y., 2019. Cyber-physical systems network to support decision making for self-adaptive production system. In: International Conference on Software, Knowledge Information, Industrial Management and Applications, SKIMA. http://dx.doi.org/10.1109/SKIMA.2018.8631512.

De Lemos, R., Giese, H., Müller, H.A., Shaw, M., Andersson, J., Litoiu, M., Schmerl, B., Tamura, G., Villegas, N.M., Vogel, T., Weyns, D., Baresi, L., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Desmarais, R., Dustdar, S., Engels, G., Geihs, K., Göschka, K.M., Gorla, A., Grassi, V., Inverardi, P., Karsai, G., Kramer, J., Lopes, A., Magee, J., Malek, S., Mankovskii, S., Mirandola, R., Mylopoulos, J., Nierstrasz, O., Pezzè, M., Prehofer, C., Schäfer, W., Schlichting, R., Smith, D.B., Sousa, J.a.P., Tahvildari, L., Wong, K., Wuttke, J., 2013. Software engineering for self-adaptive systems: A second research roadmap. In: Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Vol. 7475 LNCS, Springer, Berlin, Heidelberg, pp. 1–32. http://dx.doi.org/10.1007/978-3-642-35813-5{_}1, https://link-springer-com.ezproxy.eafit.edu.co/chapter/10.1007/978-3-642-35813-5_1.

Dennis, A., Wixom, B.H., Roth, R.M., 2014. Systems analysis and design. In: Systems Analysis and Design, sixth ed. Wiley Publishing, p. 448. http://dx.doi.org/10.1201/9781420055948.pt2.

Felson, D.T., 1992. Bias in meta-analytic research. J. Clin. Epidemiol. 45 (8), 885–892. http://dx.doi.org/10.1016/0895-4356(92)90072-U.

Hammoudi, S., Aliouat, Z., Harous, S., 2018. Challenges and research directions for internet of things. Telecommun. Syst. 67 (2), 367–385. http://dx.doi.org/10.1007/s11235-017-0343-y, https://link.springer.com/article/10.1007/s11235-017-0343-y.

Hoffmann, M., Kühn, N., Weber, M., Bittner, M., 2004. Requirements for requirements management tools. In: Proceedings of the IEEE International Conference on Requirements Engineering. pp. 301–308. http://dx.doi.org/10.1109/ICRE.2004.1335687.

International Organization for Standardization, 2011a. ISO/IEC 25010:2011 - systems and software engineering — Systems and software quality requirements and evaluation (square) — System and software quality models. https://www.iso.org/standard/35733.html.

International Organization for Standardization, 2011b. ISO/IEC/IEEE 42010:2011 - systems and software engineering. architecture description. In: BSOL British Standards Online. https://bsol-bsigroup-com.ezproxy.eafit.edu.co/Bibliographic/BibliographicInfoData/000000000030216549.

ISTQB® International Software Testing Qualifications Board, 2018. Foundation level syllabus . https://www.istqb.org/downloads/syllabi/foundation-level-syllabus.html.

Jahan, S., Riley, I., Walter, C., Gamble, R.F., Pasco, M., McKinley, P.K., Cheng, B.H., 2020. MAPE-K/MAPE-SAC: An interaction framework for adaptive systems with security assurance cases. Future Gener. Comput. Syst. 109, 197–209. http://dx.doi.org/10.1016/j.future.2020.03.031.

Janes, A., Lenarduzzi, V., Cristian Stan, A., 2017. A continuous software quality monitoring approach for small and medium enterprises, in: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion - ICPE '17 Companion, ACM Press, New York, New York, USA http://dx.doi.org/10.1145/3053600.3053618.

Janssen, W., 2018. Bias in theory and practice: a literature review of bias types and a case study of bias views at the dutch safety board.

Janzen, D., Saiedian, H., 2005. Test-driven development: Concepts, taxonomy, and future direction. Computer 38 (9), 43–50. http://dx.doi.org/10.1109/MC.2005.314.

Jensen, J.C., Chang, D.H., Lee, E.A., 2011. A model-based design methodology for cyber-physical systems. In: 2011 7th International Wireless Communications and Mobile Computing Conference. pp. 1666–1671. http://dx.doi.org/10.1109/IWCMC.2011.5982785.

Kang, J., Yu, H., 2018. Mitigation technique for performance degradation of virtual machine owing to GPU pass-through in fog computing. J. Commun. Netw. 20 (3), 257–265. http://dx.doi.org/10.1109/JCN.2018.000038.

Kephart, J.O., Chess, D.M., 2003. The vision of autonomic computing. Computer 36 (1), http://dx.doi.org/10.1109/MC.2003.1160055.

Kitchenham, B., Charters, S., 2007. Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE 2007-001.

Koziolek, H., 2011. Sustainability evaluation of software architectures: A systematic review. In: CompArch'11 - Proceedings of the 2011 Federated Events on Component-Based Software Engineering and Software Architecture - QoSA+ISARCS'11. ACM Press, New York, New York, USA, pp. 3–12. http://dx.doi.org/10.1145/2000259.2000263, http://portal.acm.org/citation.cfm?doid=2000259.2000263.

Kruchten, P., Nord, R.L., Ozkaya, I., 2012. Technical debt: From metaphor to theory and practice. IEEE Softw. 29 (6), 18–21. http://dx.doi.org/10.1109/MS.2012.167.

Kumar, M., Hanumanthappa, M., 2013. Scalable intrusion detection systems log analysis using cloud computing infrastructure. In: 2013 IEEE International Conference on Computational Intelligence and Computing Research, IEEE ICCIC 2013. IEEE Computer Society, http://dx.doi.org/10.1109/ICCIC.2013.6724158.

Lago, P., 2019. Architecture design decision maps for software sustainability. In: Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Society, ICSE-SEIS 2019. pp. 61–64. http://dx.doi.org/10.1109/ICSE-SEIS.2019.00015.

Lee, E.A., 2008. Cyber physical systems: Design challenges. In: Proceedings - 11th IEEE Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, ISORC 2008. pp. 363–369. http://dx.doi.org/10.1109/ISORC.2008.25.

Li, Z., Avgeriou, P., Liang, P., 2015. A systematic mapping study on technical debt and its management. J. Syst. Softw. 101, 193–220. http://dx.doi.org/10.1016/j.jss.2014.12.027.

Lin, J., Sedigh, S., Miller, A., 2009. Toward integrated simulation of cyber-physical systems: A case study on intelligent water distribution. In: 8th IEEE International Symposium on Dependable, Autonomic and Secure Computing, DASC 2009. pp. 690–695. http://dx.doi.org/10.1109/DASC.2009.140.

Lin, K.J., Panahi, M., 2010. A real-time service-oriented framework to support sustainable cyber-physical systems. In: IEEE International Conference on Industrial Informatics (INDIN). pp. 15–21. http://dx.doi.org/10.1109/INDIN.2010.5549473.

Marwedel, P., 2018. Embedded System Design : Embedded Systems, Foundations of Cyber-Physical Systems, and the Internet of Things. Springer International Publishing, http://dx.doi.org/10.1007/978-3-319-56045-8, http://link.springer.com/10.1007/978-3-319-56045-8.

Marwedel, P., Engel, M., 2016. Cyber-physical systems: opportunities, challenges and (some) solutions. In: Management of Cyber Physical Objects in the Future Internet of Things. Springer, pp. 1–30.

Muccini, H., Sharaf, M., Weyns, D., 2016. Self-adaptation for cyber-physical systems: A systematic literature review. In: Proceedings - 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2016. Association for Computing Machinery, Inc, pp. 75–81. http://dx.doi.org/10.1145/2897053.2897069, https://www.scopus.com/inward/record.uri?eid=2-s2.0-84974536575&doi=10.1145%2f2897053.2897069&partnerID=40&md5=47dab41342a795ec3aab22756f79810f.

Musil, A., Musil, J., Weyns, D., Bures, T., Muccini, H., Sharaf, M., 2017. Patterns for self-adaptation in cyber-physical systems. Multi-Disciplinary Engineering for Cyber-Physical Production Systems: Data Models and Software Solutions for Handling Complex Engineering Projects. pp. 331–368. http://dx.doi.org/10.1007/978-3-319-56345-9{_}13.

Pahl, G., Beitz, W., Feldhusen, J., Grote, K.H., 2007. Engineering Design: A Systematic Approach. Springer London, pp. 1–617. http://dx.doi.org/10.1007/978-1-84628-319-2.

Pankowska, M., 2013. Sustainable software: A study of software product sustainable development. In: Mechanism Design for Sustainability: Techniques and Cases. Springer Netherlands, pp. 265–281. http://dx.doi.org/10.1007/978-94-007-5995-4{_}13.

Pei Breivold, H., 2020. Using software evolvability model for evolvability analysis.

Perozo, N., Aguilar, J., Terán, O., 2008. Proposal for a multiagent architecture for self-organizing systems (MA-SOS). In: Yang, C.C., Chen, H., Chau, M., Chang, K., Lang, S.-D., Chen, P.S., Hsieh, R., Zeng, D., Wang, F.-Y., Carley, K., Mao, W., Zhan, J. (Eds.), Intelligence and Security Informatics. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 434–439.

Restrepo, L., 2021. Replication package for: "a sustainable-development approach for self-adaptive cyber-physical systems life cycle: A systematic mapping study". 1, http://dx.doi.org/10.17632/GV66S3X56W.1.

Romero, D., Quinton, C., Duchien, L., Seinturier, L., Valdez, C., 2015. Smartyco: Managing cyber-physical systems for smart environments. In: Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Vol. 9278, Springer Verlag, pp. 294–302. http://dx.doi.org/10.1007/978-3-319-23727-5_25, https://link-springer-com.ezproxy.eafit.edu.co/chapter/10.1007/978-3-319-23727-5{_}25.

Rouse, M., 2007. What is vertical scalability (scaling up)? - definition from whatis.com. https://searchcio.techtarget.com/definition/vertical-scalability.

Rowe, D., Leaney, J., Lowe, D., 1994. Defining systems evolvability-a taxonomy of change. Change 94, 541–545.

Sanchez, M., Aguilar, J., Jerez, M., Mendonca, M., 2017. An extension of the misci middleware for smart cities based on fog computing. J. Inf. Technol. Res. 10 (4), 23–41.

Sánchez Aristizábal, A., Sarmiento Garavito, S., 2019. Diagnosis evaluation of the coffee leaf rust development stage in the colombian caturra variety integrating remote sensing, wireless sensor networks and deep learning (Ph.D. thesis). Universidad EAFIT, http://repository.eafit.edu.co/handle/10784/15427.

Saputri, T.R.D., Lee, S.W., 2020. Integrated framework for incorporating sustainability design in software engineering life-cycle: An empirical study. Inf. Softw. Technol. 106407. http://dx.doi.org/10.1016/j.infsof.2020.106407.

Shelly, G.B., Rosenblatt, H.J., 2011. Systems Analysis and Design. Cengage Learning.

Sommerville, I., 2015. Software engineering. 10th. In: Book Software Engineering. 10th, Series Software Engineering. Addison-Wesley.

Song, H., Rawat, D.B., Jeschke, S., Brecher, C., 2016. Cyber-Physical Systems: Foundations, Principles and Applications. Morgan Kaufmann.

Stankovic, J.A., 2014. Research directions for the internet of things. IEEE Internet Things J. 1 (1), 3–9. http://dx.doi.org/10.1109/JIOT.2014.2312291.

Stavros, J.M., Sprangel, J.R., 2008. "SOAR" from the mediocrity of status quo to the heights of global sustainability. In: Innovative Approaches To Global Sustainability. Palgrave Macmillan US, pp. 11–35. http://dx.doi.org/10.1057/9780230616646_2.

Tsai, W.T., Bai, X., Paul, R., Shao, W., Agarwal, V., 2001. End-to-end integration testing design. In: Proceedings - IEEE Computer Society's International Computer Software and Applications Conference. pp. 166–171. http://dx.doi.org/10.1109/CMPSAC.2001.960613.

Vizcarrondo, J., Aguilar, J., Exposito, E., Subias, A., 2017. MAPE-K as a service-oriented architecture. IEEE Lat. Am. Trans. 15 (6), 1163–1175. http://dx.doi.org/10.1109/TLA.2017.7932705.

Wang, C., Gill, C., Lu, C., 2019. FRAME: Fault tolerant and real-time messaging for edge computing. In: Proceedings - International Conference on Distributed Computing Systems. Institute of Electrical and Electronics Engineers Inc., pp. 976–985. http://dx.doi.org/10.1109/ICDCS.2019.00101.

Weyns, D., Georgeff, M., 2010. Self-adaptation using multiagent systems. IEEE Softw. 27 (1), 86–91. http://dx.doi.org/10.1109/MS.2010.18.

Wohlin, C., 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering - EASE '14. pp. 1–10. http://dx.doi.org/10.1145/2601248.2601268.

Wombacher, A., 2011. How physical objects and business workflows can be correlated. In: Proceedings - 2011 IEEE International Conference on Services Computing, SCC 2011. pp. 226–233. http://dx.doi.org/10.1109/SCC.2011.24.

Xiao, Y., Bhaumik, R., Yang, Z., Siekkinen, M., Savolainen, P., Ylä-Jääski, A., 2010. A system-level model for runtime power estimation on mobile devices. In: Proceedings - 2010 IEEE/ACM International Conference on Green Computing and Communications, GreenCom 2010, 2010 IEEE/ACM International Conference on Cyber, Physical and Social Computing, CPSCom 2010. pp. 27–34. http://dx.doi.org/10.1109/GreenCom-CPSCom.2010.114.

Zeadally, S., Sanislav, T., Mois, G., 2019a. Self-adaptation techniques in cyber-physical systems (CPSs). IEEE Access 7, 171126–171139. http://dx.doi.org/10.1109/ACCESS.2019.2956124.

Zeadally, S., Sanislav, T., Mois, G.D., 2019b. Self-adaptation techniques in cyber-physical systems (CPSs). IEEE Access 7, 171126–171139. http://dx.doi.org/10.1109/ACCESS.2019.2956124, https://www.scopus.com/inward/record.uri?eid=2-s2.0-85078403794&doi=10.1109%2fACCESS.2019.2956124&partnerID=40&md5=ac0ad13602e57ff214eb6867f0bbfc4a.

Zhao, Y., Serebrenik, A., Zhou, Y., Filkov, V., Vasilescu, B., 2017. The impact of continuous integration on other software development practices: A large-scale empirical study. In: ASE 2017 - Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering. Institute of Electrical and Electronics Engineers Inc., pp. 60–71. http://dx.doi.org/10.1109/ASE.2017.8115619.

Zhou, X., Jin, Y., Zhang, H., Li, S., Huang, X., 2016. A map of threats to validity of systematic literature reviews in software engineering. In: Proceedings - Asia-Pacific Software Engineering Conference, APSEC. IEEE Computer Society, pp. 153–160. http://dx.doi.org/10.1109/APSEC.2016.031.

Züllighoven, H., 2005. In: Züllighoven, H.B.T.O.-O.C.H. (Ed.), 12 - The Development Process. Morgan Kaufmann, San Francisco, pp. 393–457. http://dx.doi.org/10.1016/B978-155860687-6/50012-8, http://www.sciencedirect.com/science/article/pii/B9781558606876500128.

Luisa Restrepo received a B.Sc. degree in Computer Science in 2015 and a M.Sc. degree in Engineering from Universidad EAFIT, Colombia with emphasis on Software engineering, in 2019. Since 2020, Luisa works as Adjunct Professor at the Department of Systems and Informatics Engineering at Universidad EAFIT. Her research interests include requirements engineering, assessment of software applications, software reuse, cyber–physical systems, and data quality.

Professor Jose Aguilar received the B. S. degree in System Engineering in 1987 (Universidad de Los Andes-Venezuela), the M. Sc. degree in Computer Sciences in 1991 (Universite Paul Sabatier-France), and the Ph.D degree in Computer Sciences in 1995 (Universite Rene Descartes-France). He was a Postdoctoral Research Fellow in the Department of Computer Sciences at the University of Houston (1999-2000) and in the Laboratoire d'Analyse et d'Architecture des Systems of Toulouse, France (2010-2011). He is a Titular Professor in the Department of Computer Science at the Universidad de los Andes, Mérida, Venezuela, and contracted professor of the Department of Systems Engineering of the EAFIT University, Medellin, Colombia. His research interests include artificial intelligence, industry 4.0, IoT, cyber–physical and autonomic systems.

Mauricio Toro received a B.Sc. degree in Computer Science and Engineering from Pontificia Universidad Javeriana, Colombia, in 2009. Mauricio got a PhD degree in Computer Science from Université de Bordeux, France with emphasis on Artificial Intelligence, in 2012. Mauricio was a postdoctoral fellow at the Computer-Science department at University of Cyprus, during 2013. Since 2014, Mauricio works as Assistant Professor at the Department of Systems and Informatics Engineering and as a researcher of the GIDITIC Group at Universidad EAFIT. His research interests include artificial intelligence, industry 4.0, machine learning, computer vision, and agricultural applications.

Elizabeth Suescún Monsalve received a B.Sc. degree in Computer Science from Politecnico Colombiano JIC, Colombia, in 2004. Elizabeth got a Master and PhD degree in Computer Science from Pontifical Catholic University of Rio de Janeiro - PUC-Rio, Brazil with emphasis on Software Engineering, from 2010 to 2014. Since 2015, Elizabeth works as Assistant Professor at the Department of Systems and Informatics Engineering and as a researcher of the GIDITIC Group at Universidad EAFIT. Her research interests include Software Engineering, DevOps, industry 4.0, Software Transparency, Intentional Modeling, cyber–physical systems and its applications.