sensors

MDPI

Review

Agile Methodologies Applied to the Development of Internet of Things (IoT)-Based Systems: A Review

Gleiston Guerrero-Ulloa [1,2], Carlos Rodríguez-Domínguez [2,*] and Miguel J. Hornos [2]

[1] Faculty of Engineering Science, State Technical University of Quevedo, Quevedo 120301, Ecuador; gleiston@correo.ugr.es or gguerrero@uteq.edu.ec
[2] Software Engineering Department, Higher Technical School of Computer and Telecommunications Engineering, Aynadamar Campus, University of Granada, 18071 Granada, Spain; mhornos@ugr.es
* Correspondence: carlosrodriguez@ugr.es

Abstract: Throughout the evolution of software systems, empirical methodologies have been used in their development process, even in the Internet of Things (IoT) paradigm, to develop IoT-based systems (IoTS). In this paper, we review the fundamentals included in the manifesto for agile software development, especially in the Scrum methodology, to determine its use and role in IoTS development. Initially, 4303 documents were retrieved, a number that was reduced to 186 after applying automatic filters and by the relevance of their titles. After analysing their contents, only 60 documents were considered. Of these, 38 documents present the development of an IoTS using some methodology, 8 present methodologies focused on the construction of IoTS software, and 14 present methodologies close to the systems life cycle (SLC). Finally, only one methodology can be considered SLC-compliant. Out of 38 papers presenting the development of some IoTS following a methodology for traditional information systems (ISs), 42.1% have used Scrum as the only methodology, while 10.5% have used Scrum combined with other methodologies, such as eXtreme Programming (XP), Kanban and Rapid Prototyping. In the analysis presented herein, the existing methodologies for developing IoTSs have been grouped according to the different approaches on which they are based, such as agile, modelling, and service oriented. This study also analyses whether the different proposals consider the standard stages of the development process or not: planning and requirements gathering, solution analysis, solution design, solution coding and unit testing (construction), integration and testing (implementation), and operation and maintenance. In addition, we include a review of the automated frameworks, platforms, and tools used in the methodologies analysed to improve the development of IoTSs and the design of their underlying architectures. To conclude, the main contribution of this work is a review for IoTS researchers and developers regarding existing methodologies, frameworks, platforms, tools, and guidelines for the development of IoTSs, with a deep analysis framed within international standards dictated for this purpose.

Keywords: Internet of Things (IoT); development methodologies; agile methodologies; software engineering; Model-Based Engineering; Model-Driven Engineering

Citation: Guerrero-Ulloa, G.; Rodríguez-Domínguez, C.; Hornos, M.J. Agile Methodologies Applied to the Development of Internet of Things (IoT)-Based Systems: A Review. *Sensors* **2023**, *23*, 790. https://doi.org/10.3390/s23020790

Academic Editor: Anfeng Liu

Received: 21 November 2022
Revised: 5 January 2023
Accepted: 7 January 2023
Published: 10 January 2023

1. Introduction

Several methodologies have been proposed to develop traditional information systems (ISs), some of them being universally known. The first methodology for the development of ISs was the waterfall methodology, presented by Winston Royce [1]. Then, other well-known and used methodologies emerged, such as Spiral [2], Rapid Prototyping (RP) [3,4], and agile methodologies [5,6]. Among the latter, some popular ones are Scrum [7–9] and Extreme Programming (XP) [9–11]. These methodologies have been widely used for the development of ISs, but with the emergence of Web(-based) Information Systems (WISs), the need for new methodologies also arose. Some examples of such methodologies are the Object-Oriented Hypermedia Design Method (OOHDM) [12,13], Hypermedia Data Bases

(HDM) [14,15], Enhanced Object-Relationship Model (EORM) [16–18], and Relationship Management Methodology (RMM) [19,20].

Nowadays, a new type of systems is currently emerging, due to the advancement and popularisation of technologies related to the Internet of Things (IoT). In this paper, we name those systems as IoT-based Systems (IoTSs). IoTSs are intended to monitor and control the environment through the deployment of sensors and actuators that can interact with each other and with Internet services. IoTSs allow us to remotely monitor the current state of any physical object or "thing", modify the conditions of the environment, obtain data to predict or infer events and make decisions in real time [21–24]. To achieve those goals, physical objects become digital objects that can be manipulated from anywhere and connected to the Internet [22,25–28]. IoTSs have been applied to multiple fields, such us improving people's lifestyles, health, work productivity, entertainment, etc. [22,24,25,29–33].

IoTSs are systems that are changing the world and will change it even more in the future. However, to fully exploit the potential of IoT, well-defined development methodologies are required, so as to improve the success rate of the development process and the quality of the resulting system.

Faced with this recent paradigm of systems, the Software Engineering (SE) research field has been tasked to propose methodologies that can specifically cover the development lifecycle of IoTS, since such systems have notable differences with traditional ISs and WISs. It should be noted that IoTSs are composed not only of software applications but have two additional components: hardware (sensors/actuators), and communication mechanisms to allow the interaction between that hardware and the Internet [34,35]. Moreover, the interaction between things and people through well-defined interfaces should also be considered too [29,36–38]. In short, there is a need to formulate and validate methodologies for the development of IoTSs.

Therefore, scientists and developers of this new type of systems, namely IoTSs, need a methodology to ensure the quality of their work. At the time of writing, there is no universally adopted methodology for the development of IoTSs. This is evidenced by the many IoTS development methodologies presented in research papers found in ScDBs (see Section 3), and by previous state-of-the-art review papers on IoTS development methodologies [39,40]. Furthermore, among the methodologies found in the literature, only one of them complies with international standards on system and/or software development lifecycles (Sections 3 and 4).

The following subsections present a background on software engineering, the objectives that guided the writing of this paper, and a state-of-the-art review of previously published works on IoTS development methodologies.

1.1. Background

As it was previously mentioned, the first software development methodology was the waterfall methodology, presented by Winston Royce [1]. This methodology is commonly used for the elaboration, manufacture, or construction of any physical product, with the particularity that the original waterfall methodology considers that the developer could return to any previous stages when necessary, even if it could be difficult or unfeasible in many cases [41]. In contrast, hardware construction methodologies do not consider that possibility, since it would involve a great economic impact.

Given the delays and limited achievements in the implementation of the waterfall methodology, it is necessary to identify the reasons for its low impact on the software industry. One of those reasons is that the waterfall methodology encompasses the development of large information systems as a whole [5,42–44]. In contrast, others, such as the spiral methodology, help to detect when it is not possible to develop the planned system and, consequently, abandon its development before investing resources in it [10,45].

Another reason is that in the analysis stage of a software system, usually end users (customers) do not have a very clear picture of the functionalities and quality properties that need to be covered. Therefore, the prototyping methodology [46] arose to formalize

the presentation of iterative versions of the product to the end users for their evaluation. Prototyping is currently used as part of other methodologies, especially agile methodologies.

Agile methodologies emerged to reduce the risk of not completing the development of large systems due to budget, technological or resource constraints, among other reasons [47,48]. The emergence of agile methodologies made it possible to improve the success rate of software development projects and reduce the budget consumption of projects that are finally abandoned [49,50].

Agile methodologies divide the overall system into deliverables (modules or subsystems), so that the customer uses or checks those deliverables before the entire system is completed. Agile methodologies are based on 4 values and 12 principles included in the Manifesto for Agile Software Development [42]. These methodologies focus only on the development of a part of the overall system, deploying fully functional products for that part of the system in a short period of time (maximum of 4 to 6 weeks).

In addition, some of the characteristics of agile software development methodologies, in contrast with traditional methodologies, are: teams must be small, the deliverables to be developed must be negotiated with the client and changes can be introduced in the project at any time [43,44]. However, in some cases, those features could turn into disadvantages [51]. For example, in an enterprise software development project, a small team of developers, as suggested by agile methodologies (in Scrum, 8 developers), is not sufficient when time is pressing [52]. Additionally, whenever an unexpected change needs to be performed to software, there could be budget issues, since the development team could have to deal with it using the original budget [53]. Moreover, if the decision for the development priority of each deliverable primarily relies on the customer [52,54], this may not help the development team to be productive.

Regarding IoTSs, we found in the analysis presented herein that the authors have usually completed their developments using ad hoc methodologies or without any explicitly mentioned methodology at all. One example is the work of Gea et al. [55], which presents a system developed to integrate existing sensor networks and an intelligent front-end application built with the technology of the moment, but without mentioning the development methodology used. Another example is the work done by Yelamarthi et al. [56] which describes several IoTSs developed for different purposes, including healthcare, structural health monitoring, agriculture, and tourist guidance. However, the research process suggests that no definite methodology has been followed in the development process. Additionally, some authors have developed IoTSs following methodologies not specifically designed for this type of system. A couple of examples of this are the combination of Scrum with XP [57,58] and the combination of Scrum with RP [59,60].

1.2. Objectives

This paper aims to guide researchers and developers on methodologies that

- Have been proposed for IoTS development.
- Comply with the life cycle of software systems according to standards issued jointly by the International Organization for Standardisation (ISO), the International Electrotechnical Commission (IEC), and the Institute of Electrical and Electronics Engineers (IEEE). The application of ISO/IEC/IEEE 15289:2019 [61] to the development of IoTSs contributes to the delivery of a quality product on time and within budget [62].
- Consider the specific developmental aspects of IoTSs.

Therefore, we present an in-depth analysis of the state-of-the-art IoTS development methodologies, so that researchers and developers could choose the most appropriate methodology for the development of their IoTSs. Moreover, researchers could work on this analysis to define a methodology specifically designed for the development of IoTSs, and that complies with the ISO/IEC/IEEE 15289:2019 standards and that covers all aspects of the life cycle of such systems [61].

1.3. State of the Art in Methodologies to Develop IoTSs

SE is responsible for providing an adequate methodology for the development of all types of computer systems [63,64]. Therefore, researchers in this field have been working to provide optimal methodologies to develop the different types of systems that have recently emerged [64].

In the literature, there are several methodologies for the development of IoTSs. For example, INTER-METH, which was presented by Fortino et al. [65], is a methodology for the development of IoTSs based on the waterfall methodology, making it iterative. Likewise, Test-Driven Development Methodology for IoTSs (TDDM4IoTS), presented by Guerrero-Ulloa et al. [37], is based on the manifesto for agile software development and complies with the stages of the system development lifecycle [66,67]. However, none of them is universally accepted [39,40]. This fact is evident in the state-of-the-art reviews of methodologies for IoTS development found in the ScDBs considered, which are presented below. However, those works pursue different objectives from those set out in this article or follow substantially different methodologies for their study.

Bouanaka et al. [40] present a state-of-the-art review of IoTS development methodologies applied to smart traffic lights. They consider that there is a limited number of methodologies that can be used to develop these types of IoTSs and clearly represent their specific characteristics. The authors provide information to support the decision-making of a small group of developers when deciding on which methodology to follow.

Fortino et al. [39] have also addressed the analysis of existing methodologies for the development of IoTSs. However, the products analysed in the study are based on third-party surveys, rather than on scientific publications. Some aspects that are analysed in the revised methodologies match with those of the present study, such as the development life cycle stages. To unify the terminology found (methodology, framework, platform, tool) during their review process, Fortino et al. [39] refer to ISO/IEC/IEEE 24765 [68], SEBoK (Systems Engineering Body of Knowledge) [69], and PMBoK (Project Management Body of Knowledge) [70,71]. However, none of the previous state-of-the-art review works mention the ISO/IEC/IEEE standards on which they base the analysis.

The present document is proposed as a result of not having found in the literature an exhaustive review of the state-of-the-art IoTSs concluding whether there is a universally adopted methodology for developing IoTSs. This work also aims to present a review of the minimum conditions of exclusion and a thorough analysis of the existing literature. Moreover, this work analysis if the existing proposals adhere to international standards, so as to guarantee the quality of the IoTSs developed.

So far, no IoTS development methodology has been set in ISO/IEC/IEEE 15289:2019 [61] as a standard that unifies the processes specified in ISO/IEC/IEEE 12207:2017 [72] and ISO/IEC/IEEE 15288:2015 [67]. Although none of the proposed methodologies covers all the technical aspects of the software systems life cycle set out in ISO/IEC/IEEE 15289:2019, findings from extant literature suggest that TDDM4IoTS [37] is the only methodology whose IoTS development life cycle follows that standard.

The remainder of this document is organised as follows: Section 2 presents IoTSs developed with traditional software development methodologies. Section 3 discusses existing methodologies for developing IoTSs. Section 4 provides a study of the automated tools and frameworks used for the development of IoTSs. Section 5 is dedicated to architectures for IoTSs, but from a dual point of view: those incorporated into the developed IoTSs and those used in the development methodologies. Finally, Section 6 presents the main conclusions of the analysis carried out.

2. Methodologies Designed for the Development of IoTSs

A system development methodology can be defined as "a series of stages of a software or hardware creation following a pattern based on experience and theory of program design" [73]. On the other hand, a possible definition of software development methodology would be "a process of dividing software development work into smaller, parallel, or sequential steps or sub-processes to improve design, product management" [74]. Therefore,

both systems and software development methodology may include the predefinition of specific deliverables and artifacts that a project team creates and completes to develop or maintain an application or system [75].

Developing IoTSs using methodologies that are designed for the development of traditional ISs has the great disadvantage that they do not cover specific aspects of IoTSs, such as the design and deployment of hardware (e.g., sensors, actuators, processors, and so on) in the environment to be controlled. Moreover, other aspects that, although not unique to IoTSs, are essential in this type of system, such as the incorporation of artificial intelligence (AI) techniques to help the system in decision-making and to be context-aware, i.e., it reacts appropriately according to the context or the existing conditions in the environment [39,42]. Therefore, and if we also consider the heterogeneity of the components and application domains of IoTSs [76,77], it is obvious to conclude that professionals from different areas should be part of their development team in order to carry out the activities included in the different stages of its development, either to act throughout the whole life cycle or to carry out specific tasks [78].

2.1. Stages or Processes of the Software System Development Life Cycle

To make an analysis of system development methodologies, we could go back to the 40s, when the digital or general-purpose computer called EDVAC was created [79], or to the 50s and 60s, when many programming languages appeared, such as Fortran, Cobol or Basic, to name a few [80]. The first software development methodologies for desktop systems were defined according to the paradigm on which the programming language to implement the system was based. For example, to develop using structured programming languages, structured development methodologies appeared [37,81], while for development using object-oriented programming languages, object-oriented methodologies appeared [37,82]. Subsequently, the methodologies were reoriented according to the type of system to be developed, such as desktop or web [37].

The *waterfall* methodology comprised the following phases or stages: (a) system requirements analysis, (b) software requirements analysis, (c) preliminary program design, (d) system analysis, (e) software design, (f) coding, (g) testing, and (h) operation.

Subsequently, agile methodologies, among which some popular ones are XP, Kanban, and the Scrum framework [83], consider that the values and principles of the agile manifesto should be present in the development methodology of any software product that can be broken down and developed into parts. In the development of any IoTS, it is necessary to consider: the deployment of different hardware components (e.g., sensors, actuators, processors, and so on) in the environment to be controlled, communication and interaction between objects or " things" connected, and the development of the means for user interaction with the system (e.g., web application, mobile application, and so on) [22,84–86], among other aspects, which will eventually become deliverables. These deliverables will be developed by carrying out different activities. Therefore, we can conclude that this type of system can be developed using agile methodologies.

2.2. Standards That Define the Stages and Processes of the Software Systems Life Cycle

ISO/IEC 12207:2017 [72] addresses software lifecycle processes, while ISO/IEC/IEEE 15288:2015 [67] addresses systems lifecycle processes. In turn, the ISO/IEC/IEEE 15289:2019 standard [61] proposes a common life cycle for systems and software engineering, based on the life cycle processes specified in the two standards. Similarly, the purpose of ISO/IEC/IEEE 24748-1:2018 [87] is to facilitate the joint use of the content of ISO/IEC/IEEE 15288 and ISO/IEC/IEEE 12207, providing unified and consolidated guidance on systems and software lifecycle management. On the other hand, ISO/IEC/IEEE 24748-3:2020 [88] provides guidance on the application of the software lifecycle process standard, i.e., ISO/IEC/IEEE 12207:2017. In addition, ISO/IEC/IEEE 24748-4:2016 [89] provides detailed requirements and guidance on the application of system lifecycle processes, i.e., ISO/IEC/IEEE 15288. Figure 1 has been

prepared after reviewing the standards to show the life cycles and processes that these standards involve.

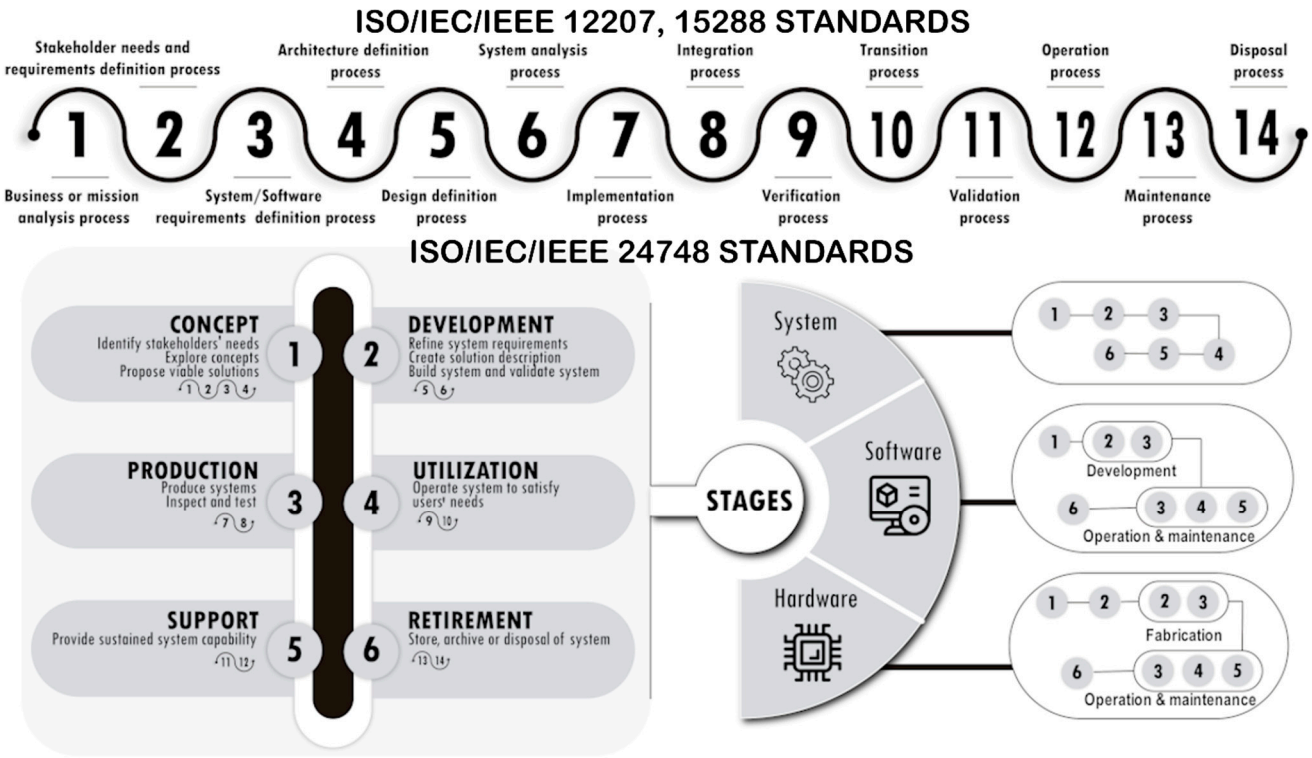

Figure 1. Summary of the stages and processes of the life cycle of software systems considered in the different reviewed ISO/IEC/IEEE standards [61,67,72,87,88].

Methodologies are free to specify how to execute the stages and technical processes that the ISO/IEC/IEEE standards propose, their execution order, and even select which processes will be executed [67]. However, presumably, at least the following processes must be present in the lifecycle: (1) planning and requirements gathering, (2) solution analysis, (3) solution design, (4) solution coding and unit testing (construction), (5) integration and testing (implementation), and (6) operation and maintenance [64,90–92]. Additionally, all the work done in each of the activities carried out should be well documented. Therefore, the difference between the methodologies must be in the processes to be considered, in the way they are executed, and in their execution order [61,67,72,87–89,91].

2.3. IoTS Development Methodologies Based on the Agile Manifesto

These methodologies split the system to be developed into deliverable products, which in turn are organised into tasks that last from 2 to 4 weeks (called *sprint* in Scrum) [92,93]. This way of organising the work contrasts with other development methodologies, such as Waterfall, Spiral, RP, etc., that take the problem to be solved in its entirety.

2.3.1. Guidelines for Project Risk Management

Agile methodologies treat risks indirectly, by establishing the characteristics of work teams. Thus, for example, Pico-Valencia et al. [94] present a methodology for small teams based on the principles defined in Scrum [95]. In this way, they try to make up for the lack of clear guidelines on the agile software development manifesto to specify stages or activities that deal with risks during the development of the systems. The authors consider that building small development teams and holding daily face-to-face meetings are two characteristics that can lessen the risks inherent when people are involved. The gradual delivery of deliverables can also reduce the risks inherent to the technology itself

(its non-availability in the market, degree of dominance on the part of developers, etc.), and the budget (acquisition of components, staff training, and so on).

According to Abrahamsson et al. [5], the agile manifesto (and, consequently, Scrum) does not specify anything regarding software development processes. Consequently, Scrum could be considered a project management methodology rather than a software development methodology, since it does not specify the activities that must be carried out during the development life cycle, while it specifies how to manage the project lifecycle [44,70,71]. Muntés-Mulero et al. [96] incorporate risk analysis as an integral part of each sprint in the Scrum methodology.

In agile methodologies, which are the focus of this work, the processes that are specified will be carried out more than once, depending on the number of deliverables into which the system has been divided. The order in which they are executed may be different, depending on the design of the methodology. For example, tests in Scrum are performed at the end [92], while in XP they are written at the beginning of the development cycle, as stipulated by TDD (Test-Driven Development) [97].

2.3.2. End-User Needs and Requirements Definition Process

One of the reasons behind the success of agile methodologies is the distinguishing feature of being able to make changes during the development of the system at any stage. Ideally, all requirements should be known and clear from the beginning of development, although this is not always the case. The elicitation of requirements is a problem known to developers [98] and causes developers to take the changes suggested by customers or end users throughout the whole development process.

For agile methodologies, ISO/IEC/IEEE 26515:2018 defines user stories, scenarios, and characters as tools for obtaining and analysing requirements, while use cases are proposed as a design technique [99]. For example, Scrum proposes to use user stories as a tool for acquiring requirements [7,8,92,100]. User stories are simple narratives that illustrate a user requirement from the perspective of a person or actor [98]. User stories must be thoroughly understood by developers to express system and software requirements. User stories are written in natural language. Therefore, they are unstructured tools and could be misinterpreted by developers [101].

It is imperative that there is a consensus among developers on how to properly obtain system requirements, as they are necessary to define the initial system architecture. Obtaining the requirements is complicated, and it is unlikely to obtain all of them at the beginning of the development of the system [102]. Therefore, in this scenario, it is foreseeable that there will be changes throughout the development of the system. Undoubtedly, the possibility of adapting to such changes, like agile methodologies can do, would be an important point to include in the guidelines of a methodology specifically designed for the development of IoTSs.

2.3.3. Non-Functional Requirements

It is very important to reflect on non-functional requirements (NFRs) in the development of an IoTS. The security and privacy of data captured by sensors [103,104], the durability of power supplies (batteries) [91,105], the resilience of communications [105,106], and the intrusiveness of sensors [103,107], among other NFRs, must be considered at each stage of an IoTS development. Therefore, for example, there are research lines on different methods, protocols, and guidelines to guarantee data security and privacy [30,108–110], low energy consumption [110–113], or to integrate sensors with different levels of intrusiveness [114,115], among others.

Consequently, the importance given to NFRs in IoTSs could make agile methodologies inappropriate for their development, since requirements are elicited through user stories, due to the problems mentioned in Section 2.3.2. User stories could be used to identify NFRs when they are user needs [98], but according to Sachdeva and Chung [116], Scrum has no clear way to check whether NFRs are met or not. This risk can be reduced or even

avoided if acceptance criteria are defined in detail, and NFRs are clearly described in user stories from the outset, relying on additional information and subsequent activities to this end, as advised by Pecchia et al. [98]. To ensure success in obtaining requirements, the writing of user stories (if such a tool is used to obtain requirements) should be carried out by both parties, i.e., developers and end users, who should also need to be involved in the development of the IoTS [96], to ensure that it will finally meet their needs and expectations.

2.3.4. Number of Development Team Members

PMBoK [70,71] provides guidelines for managing work teams from small to large. However, popular agile methodologies have been thought to work with small teams of a maximum of 15 people, including the product owner [117]. For example, Kanban indicates a maximum of 14 developers [118], XP and Scrum, and a maximum of 11 members, with the scrum master and the owner of the product [117,118]. Regarding the roles of the team in Scrum, Kettunen, and Laanti [119] raise the need to create other roles additional to those defined in Scrum for larger software companies. Along the same lines, Morais dos Santos et al. [120] also raise the need to adapt Scrum for large software projects. In fact, they adopt the Scrum of Scrums (SoS) technique and add two new roles, called General Product Owner (GPO) and General Scrum Master (GSM). These new roles will serve to coordinate and assist other roles.

2.4. Modeling as a Key in IoTS Development Methodologies

From the perspective of software developers, IoTSs are mainly characterized by the heterogeneity of their components and the technologies they involve, in addition to the scarce processing capacity of each component [121]. For example, due to the lack of well-established standards at the hardware level, manufacturers often provide different implementations and/or operational features, which often leads to a heterogeneous software and communications platform (i.e., with different communication technologies and protocols) [34,122]. In addition, developers typically must deploy a shared set of functionalities across multiple and different devices [123–125]. Consequently, in that scenario, hardware heterogeneity leads to different source codes of the same software design, just to be able to support the different features of the underlying hardware.

Model-driven development methodologies can help address hardware heterogeneity. In fact, these methodologies were introduced with the aim of focusing on the design of a system's functionalities, mainly at a platform-independent level of abstraction. Its main objective is to be able to obtain final implementations of the system with as little platform-dependent code as possible written by the developers.

The most important model-driven development methodologies known so far are Model-Based Engineering (MBE) [126,127], Model-Driven Engineering (MDE) [127], Model-Driven Development (MDD) [127], and Model-Driven Architecture (MDA) [127,128]. Although the similarity between them is easy to recognise, their difference lies in the importance they place on the models themselves, how they conceptually define "what is" a model, and how they are used to ultimately obtain an implementation of the system. In addition, they differ in the stages of development involved and the tools they propose to enable modelling [129].

Both MDD and MDA are guidelines to follow during the software development stages. The difference between them is that MDD can be written in any modelling language, while MDA is a standard specification that clarifies that UML (Unified Modelling Language) should be used as the main modelling language, while any transformation should be specified using the Query/View/Transform (QVT) language. MDA also specifies that models should be transformed following a descending abstraction level order, that is, from computationally independent models (CIMs) to platform-independent models (PIMs), and these two platform-specific models (PSMs) [127,128]. As expressed in the standard specification, MDA is an approach to software design, development, and implementation spearheaded by the Object Management Group (OMG) [130]. MDA provides guidelines

for structuring software specifications that are expressed as models [131]. Thus, MDA aims to set aside the technical particularities of implementations, and instead focuses on "modelling" software solutions. On the other hand, it separates itself from requirement elicitation, assuming it as a preliminary work already completed at an earlier stage.

In MBE, neither model definition nor automatic code generation constitute key aspects of the development process. In addition, it considers models as plans that must be understood by programmers to write program code in a target programming language. Instead, MDE is governed by models, as models are expected to be defined to generate (semi-)automatically at least one partial codebase, or even other models from them [132]. This process is commonly referred to as model-to-text (M2T) or model-to-model (M2M) transformation. In addition, in MDE, models and transformations can be defined in any modelling language and can refer to different levels of abstraction.

3. Methodologies for Traditional IS Development Applied to IoTS Development

As mentioned above, methodologies devised for the development of conventional I'S have been used, and even some of them have been adapted for the development of IoTSs. To review the literature on this subject and analyse the existing proposals, the related articles have been retrieved and reviewed, consulting several scientific databases (ScDBs), namely, Web of Science (WoS) (https://www.webofscience.com/wos/alldb/advanced-search, accessed on 21 November 2022) and Scopus (https://www.scopus.com/search/form.uri?display=advanced, accessed on 22 November 2022), IEEE (https://ieeexplore.ieee.org/search/advanced, accessed on 23 November 2022), and ACM (https://dl.acm.org/search/advanced, accessed on 24 November 2022), to ensure that we consider the largest number of articles published on this research topic.

Figure 2 shows the flowchart of the procedure carried out for the retrieval of documents for the review of the state-of-the-art that is being presented in this article. It shows the processes that the authors had to perform manually and the processes that were executed automatically with the help of the tools provided in the ScDBs consulted. The objective of our search was to retrieve the documents in which some Methodology, Framework, Platform, Tool, or Guidelines for IoTS Development (MFPTG4IoTSD) is proposed, in addition to those in which the development of an IoTS is described, provided that the methodology applied to develop it is specified.

The search carried out was very extensive, as can be seen by consulting Table 1, which shows the search statements entered in each of the ScDBs consulted, as well as the number of documents initially retrieved and the number of documents resulting after applying each of the filters indicated in the procedure described in Figure 2. The execution of the queries was carried out on 14 October 2022, and alerts were registered in the databases so that the data of the new documents that meet the search criteria established for this work are sent to the email of one of the researchers (last revision on 22 December 2022). At the time of submitting this article, no new matches have been received.

Making use of the analysis tools of the WoS platform, Figure 3 shows the distribution in time of the jobs retrieved with our search sentence (3201 records) since the beginning of the popularisation of the term IoT in 2009 [133], having obtained only 2 works published in that year. The appearance of articles related to IoT is constantly growing, although in 2020 there was a decrease in the number of publications of research papers, probably due to the COVID-19 pandemic, which led to health problems and restrictions on the mobility of the population [134,135]. However, this growth resumed in 2021. It should be noted that from 2022 onwards, papers published up to the date of submission for publication of this article have been considered.

Of the filters applied (see Table 1 and Figure 2), the first two, that is, by title and language, were automatic filters. The *Type* column in Table 1 indicates the number of papers that meet to be peer-reviewed articles and published in standard publication sources, such as journals, conferences, book chapters, and books. The *Language* column shows the number of articles that have been written in English or Spanish (only one of the finally

selected is written in Spanish). However, the filter by title was carried out manually. The *Title* column shows the number of articles whose title was considered significant for the present investigation.

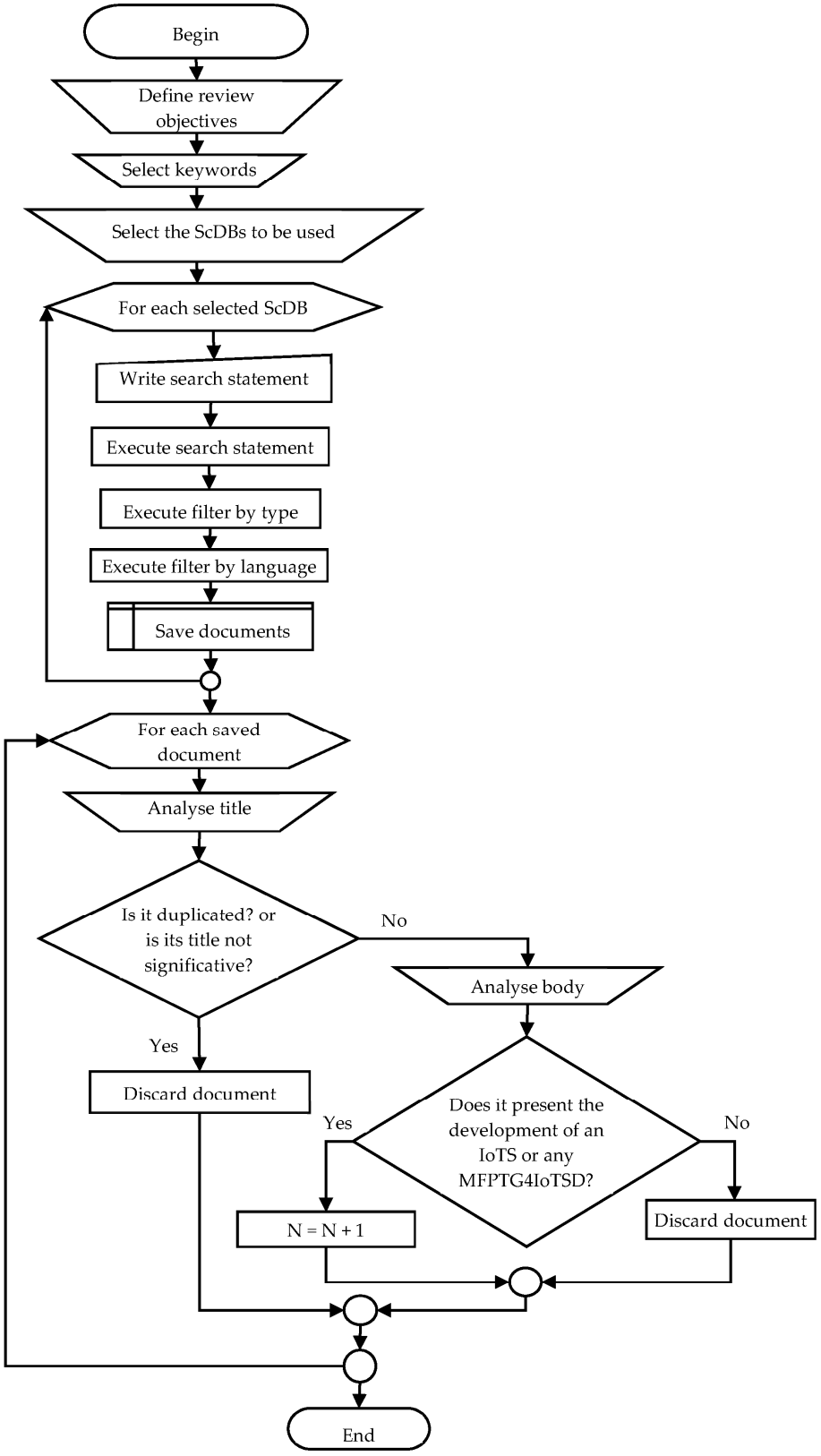

Figure 2. Flowchart of the state-of-the-art review process.

Table 1. Search statements for each ScDB were consulted and a number of results were consulted after executing them and after applying the corresponding filters. As usual, the wildcard character "*" is used to indicate that it could be substituted for any string (0 or more characters) at that place in the query.

ScDB	Search Sentence	Results	Filters		
			Type	Language	Title
ACM	Keyword:((IoT OR "Internet of Things") AND ("develop* method*" OR "design* method*" OR "construct* method*" OR "implement* method" OR "develop* framework" OR "design* framework" OR "construct* framework" OR "implement* framework" OR "develop* tool*" OR "design* tool*" OR "construct* tool*" OR "implement* tool*" OR "develop* guidelines" OR "design* guidelines" OR "construct* guidelines" OR "implement* guidelines" OR "develop* lifecycle" OR "design* lifecycle" OR "construct* lifecycle" OR "implement* lifecycle" OR "develop* platform*" OR "design* platform*" OR "construct* platform*" OR "implement* platform*"))	3	3	3	3
IEEE	("Index Terms":IoT OR "Index Terms":"Internet of Things") AND ("Index Terms":"develop* method*" OR "Index Terms":"design* method*" OR "Index Terms":"construct* method*" OR "Index Terms":"implement* method" OR "Index Terms":"develop* framework" OR "Index Terms":"design* framework" OR "Index Terms":"construct* framework" OR "Index Terms":"implement* framework" OR "Index Terms":"develop* tool*" OR "Index Terms":"design* tool*" OR "Index Terms":"construct* tool*" OR "Index Terms": "implement* tool*" OR "Index Terms":"develop* guidelines" OR "Index Terms":"design* guidelines" OR "Index Terms":"construct* guidelines" OR "Index Terms":"implement* guidelines" OR "Index Terms":"develop* lifecycle" OR "Index Terms":"design* lifecycle" OR "Index Terms":"construct* lifecycle" OR "Index Terms":"implement* lifecycle" OR "Index Terms":"develop* platform*" OR "Index Terms":"design* platform*" OR "Index Terms":"construct* platform*" OR "Index Terms": "implement* platform*")	452	429	429	40
WoS	TS=(IoT OR "Internet of Things") AND TS=("develop* method*" OR "design* method*" OR "construct* method*" OR "implement* method" OR "develop* framework" OR "design* framework" OR "construct* framework" OR "implement* framework" OR "develop* tool*" OR "design* tool*" OR "construct* tool*" OR "implement* tool*" OR "develop* guidelines" OR "design* guidelines" OR "construct* guidelines" OR "implement* guidelines" OR "develop* lifecycle" OR "design* lifecycle" OR "construct* lifecycle" OR "implement* lifecycle" OR "develop* platform*" OR "design* platform*" OR "construct* plat-form*" OR "implement* platform*")	3201	2475	2344	83
Scopus	KEY ((IoT OR "Internet of Things") AND ("develop* method*" OR "design* method*" OR "construct* method*" OR "implement* method" OR "develop* framework" OR "design* framework" OR "construct* framework" OR "implement* framework" OR "develop* tool*" OR "design* tool*" OR "construct* tool*" OR "implement* tool*" OR "develop* guidelines" OR "design* guidelines" OR "construct* guidelines" OR "implement* guidelines" OR "develop* lifecycle" OR "design* lifecycle" OR "construct* lifecycle" OR "implement* lifecycle" OR "develop* platform*" OR "design* platform*" OR "construct* platform*" OR "implement* platform*"))	647	646	636	60

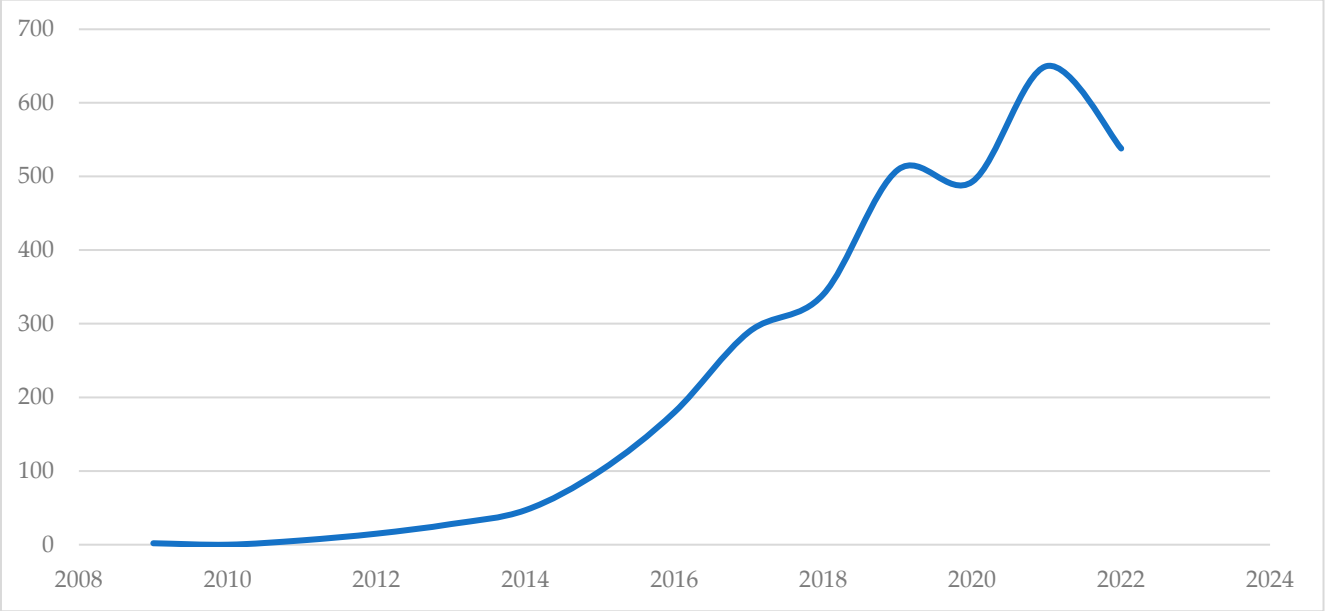

Figure 3. Year of publication of the documents selected for analysis.

Another of the manual filters, and the most important, was the filter for the information contained in the body of the document. Thus, to be considered a document, it must present: (1) The development of any IoTS, application or device, provided that its authors present evidence of having used any MFPTG4IoTSD; or (2) A development methodology, so that (2.1) the main objective of the authors of that MFPTD4IoTSD has been the design and construction stages of the corresponding system, or (2.2) the work presents some broader MFPTD4IoTSD, that is, it does not only specify the design and construction phases of the system.

After the application of this last manual filter, 60 documents of interest for the present research were found, of which 38 documents present the development of some IoTS following a development methodology for traditional IS (see Figure 4), 8 papers present MFPTG4IoTSDs that address the design and construction phases of the software for IoTSs, i.e., they differ greatly from the life cycle presented in the ISO/IEC/IEEE 15289:2019 standards, and 14 documents present MFPTG4IoTSDs that can be considered to be within the ISO/IEC/IEEE standards mentioned. Documents presenting the development of any IoTS where the development methodology used is not clearly specified were not considered.

Figure 4 shows the frequency or number of times that methodologies devised for the development of conventional ISs have been applied to the development of IoTSs in the set of documents analysed in the study we have carried out.

As can be seen in Figure 4, most of these IoTSs have been developed using the Scrum methodology [60,116,120,136–148], specifically in 42.11% of the IoTSs considered, with RP [149–163] being the other most used methodology, with 39.47%. In addition, Scrum has been used in combination with other methodologies in 10.53% of cases. So, for example, there are developments by combining Scrum with XP [57,58], or using a combination of Scrum, XP, and Kanban [164], or combining Scrum with RP [59]. The least used methodologies have been Rapid Application Development (RAD), V-Model [165], and SDLC (System Development Lifecycle) [166], representing 2.63% each, while Kanban has only been used for the development of IoTSs in combination with Scrum and XP.

In the works represented in Figure 4, the areas or fields of application in which the IoTSs proposed them have been developed have also been identified. The results of this analysis can be seen in Figure 5, where it can be seen that the domain in which the development of IoTSs has been most formalised is Health Care, with 18.42%, followed by those of Smart Car and Air Quality, with 10.53% in each of them with respect to the total of the works considered for this analysis.

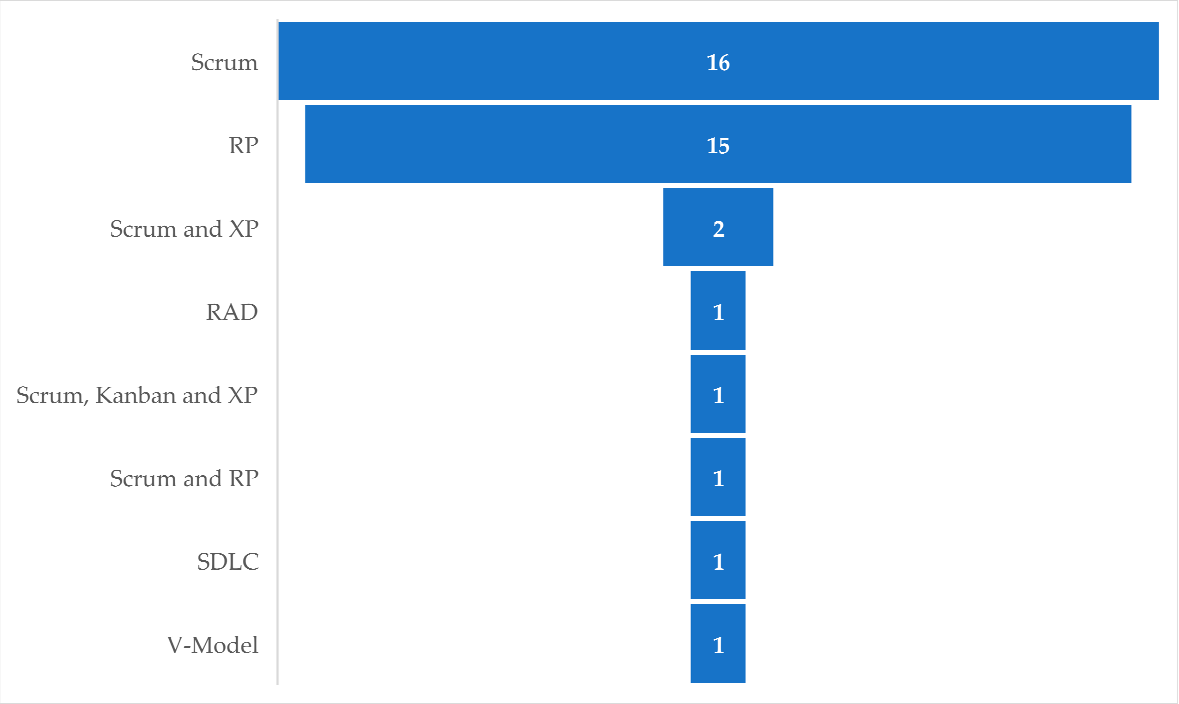

Figure 4. Number of IoTSs developed with methodologies designed for the development of ISs.

Although the growth of IoT and IoTSs has been dizzying in recent years, not everyone has access to the Internet yet. According to the World Bank [167], only 49. 72% of the population has access to the Internet. In fact, there are countries whose population with Internet access is below 25%. An even smaller percentage of people probably know what IoT is and how IoT can benefit them. One of the methodologies that has been successful when the client is not clear about their requirements, either because they do not know the use they can give to the existing technology, or because they are not very clear about the requirements of the system to be developed, is the prototyping methodology. Therefore, these data suggest that the use of prototyping for the development of this type of system will be a success and will be well-valued by customers. Moreover, the possibility of not continuing with the development of the system, whether due to lack of budget, technological issues (such as unavailable technology), or of any other nature, is another characteristic that supports the success of agile development.

3.1. Methodologies, Tools, and Frameworks Focused on the Design and Construction of Software for IoTSs

For this classification of documents, those with an MFPTG4IoTSD that do not specify the stages of the system life cycle were considered. These methodologies focus on the design and construction stages of IoTSs. In these MFPTG4IoTSDs, the design is mostly based on models and metamodels. For software construction, technologies are presented that are capable of automatically generating code in various languages. Most of them focus on generating code in C, C++, or variants, as they are the most popular type of languages on computer boards (controllers) used in the development of IoTSs. Another programming language in which they generate code is Java, and only one of the references found mentions the generation of code for the Node.js environment. Table 2 shows the main characteristics of non-traditional methodologies that address the design and construction stages of IoTSs. In addition, it shows the tools they have used or proposed to carry out their objectives.

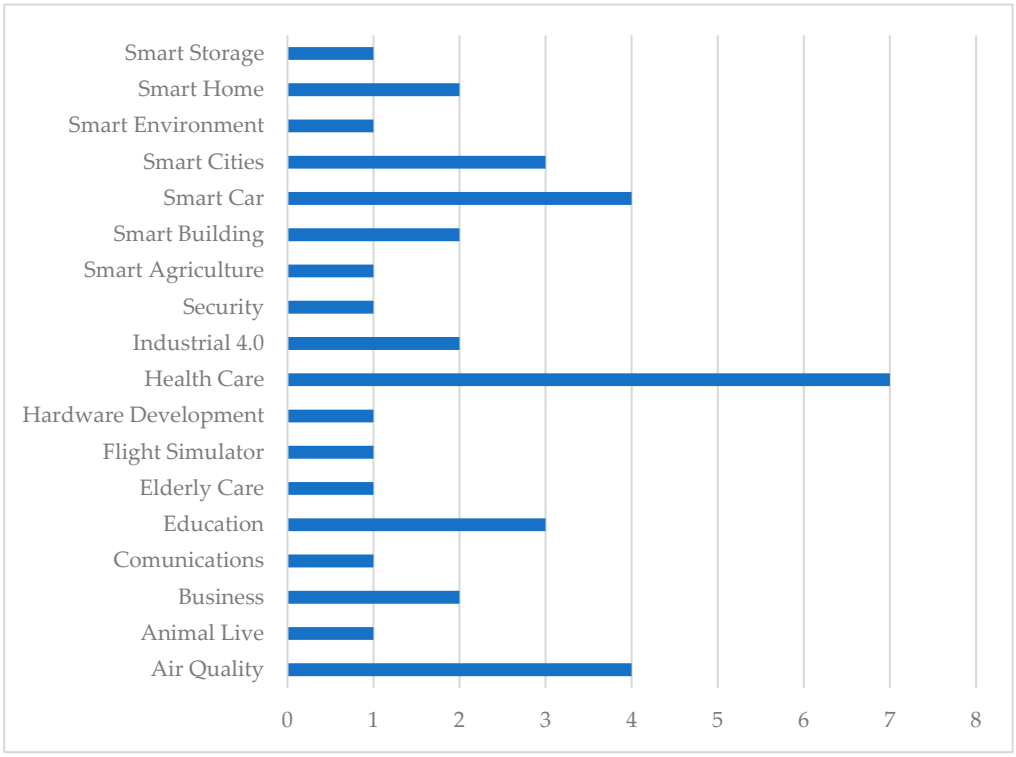

Figure 5. IoTSs application areas are developed with methodologies designed for ISs.

To start the software design and construction stages, you must first go through the stages of analysing the needs of the stakeholders and the elicitation of system requirements, and then move on to the stage of analysing both system and software requirements. These stages are considered as very important stages in some works [168,169] (those marked with ☑ in the Requirements column of Table 2), being these stages, the providers of the information needed to continue with the design and construction. However, other works [170,171] (those marked with ~ in the Requirements column of Table 2) take them as resolved, putting a lower emphasis on them than in the previous works, and without specifying any analysis method or tools to be used. Moreover, the methodologies presented in other works [172,173] do not mention the requirements.

Table 2 shows other aspects of the methodologies and tools reviewed in this study. More specifically, it shows whether the requirements analysis is considered, or it is only mentioned, or it is not specified in the corresponding methodology, as well as their respective modelling languages, and the artifacts used to obtain and analyse the requirements, model and/or generate the code.

The contribution of Lekidis et al. [168] consists of an IoTS design flow based on MDE and SOA. These authors focus on models for IoT Wireless Personal Area Network (WPAN) systems. This proposal also supports the modelling and implementation of the application functions to their deployment in the IoT system. The steps specified by the flow are (1) translation for the construction of the application model, (2) translation for the synthesis of the OS/kernel model, (3) transformation for the construction of the system model, (4) code generation, (5) space state exploration, (6) calibration, (7) verification of statistical models, and (8) injection of failures. Design activities are supported by requirements verification and validation processes, facilitating system model refinement. This ensures compliance with NFRs related to application performance and efficiency, as well as functional requirements (FRs). Their work focuses on the Contiki platform, which uses a proprietary DSL (Domain Specific Language) that serves to write the REST services that run on that platform.

Table 2. Methodologies focused on IoTS development, and the artifacts used and/or recommended to achieve their objectives.

Year	Reference	Requirements *	Artifacts for			Approach	Application
			Modelling Language	Analysis and Modelling	Code Generation		
2018	Lekidis et al. [168]	☑	✗	DSML [a]	C/C++	MDE, SOA [b]	Smart Buildings
2017	MDE4IoT [c] [172]	✗	UML, DSML	✗	Java, C/C++	MDE	Smart Cities
2017	IOPT [d] [174]	✗	Petri net	IOPT networks	ANSI C	Petri Net	Smart Car
2017	Brambilla et al. [170]	~	mobile IFML [e]	~	NS [f]	Components and patterns	Several domains
2017	Harbouche et al. [169]	☑	UML	AD [g], SD [h]	NesC, Java	MDE	Health Care
2016	Chauhan et al. [175]	DL [i]	Holder	AL [j], UIL [k]	Node.js	MDD	Smart Home
2012	ROOD [176]	NDE [l]	SsML [m]	SOM [n]	J2ME [o]	MDA	Smart Gym
2012	ELDAMeth [171]	~	✗	ELDA [p], MMM [q]	Java	Agents	Mobile Agents

[a] Domain-Specific Modelling Language; [b] Service-Oriented Architecture; [c] MDE for IoT; [d] Input-Output Place-Transition; [e] Interaction Flow Modelling Language; [f] NOT specified; [g] Activity Diagrams; [h] Sequence Diagrams; [i] Domain Language; [j] Architecture Language; [k] User Interaction Language; [l] Environmental Context Model; [m] Smart Space Modelling Language; [n] Smart Object Model; [o] Java 2 Micro ©Edition; [p] ELDA (Event-driven Lightweight Distilled state charts-based Agents) [177]; [q] MAS (Multi-Agent System) Meta-Model. ☑ Very important; ~ Mentioned; ✗ They don't mention them; * Collection and analysis of requirements.

MDE4IoT [172] is a methodology based on MDE that is focused on modelling and generating the final product. This methodology does not mention the stages of planning, obtaining, and analysis of requirements, operation, maintenance, or deployment. The elicitation of system requirements is also not addressed by its authors. To achieve the transformation of models to executable artifacts, MDE4IoT leverages a combination of Domain-Specific Modelling Languages (DSMLs). Modelling is done from three points of view: (1) specific software application domain, (2) physical devices, and (3) both software and hardware of a specific application domain.

IOPT-Tools [174] allows control modelling for embedded systems through a class of Petri nets called "Input-Output Place-Transition nets". IOPT works through a Web interface and allows you to obtain the control code from a single graphical model. However, if you want to communicate different controllers with each other, then you must manually write and adapt the generated code. Therefore, although it is a tool for IoT devices in general, it does not consider how to implement the interaction with other components of an IoTS or with the users themselves.

Brambilla et al. [170] present an approach for building mobile applications for IoTSs. This approach is based on an extension of UML, known as Interaction Flow Modelling Language (IFML), designed to express the content, user interaction, and control behaviour of the front-end of mobile applications. To model both the events and actions associated with IoT devices, new elements have been added to this set of graphical notations (IFML) to create models that visually represent the behaviour of systems in the face of user interactions. To define patterns that encompass the most common IoT use cases, the authors define content class models and interaction class models. The patterns addressed are specific to IoT, user interaction, and data synchronization. It is not specified in which programming languages the software code is generated. In addition, this proposal covers some limited areas of application, and even in those areas it is limited to only certain types of systems.

Harbouche et al. [169] propose an MDE approach that allows developers to derive a system design from the overall specification of their requirements. Their design methodol-

ogy follows the top-down paradigm, and they bet on automatic processes for the derivation of the behaviours of the global requirements of the system towards a set of collaborative components to eliminate possible errors. Each level of abstraction is described using a specific metamodel. Therefore, the application of an MDE approach requires the definition of the appropriate metamodels and the corresponding model transformations.

Chauhan et al. [175] present a development framework that encompasses the following: the domain specification, application architecture design specification, architecture framework generation, domain framework generation, definition of a set of abstract user interactions, generation of user interfaces, and description of the implementation specification. Another of its considerations is the generation of the code of the programs that can be deployed in the devices themselves. For each of these aspects of the methodology, a language is defined that will be used by the different members of the development team.

ROOD (Resource-Oriented and Ontology-Driven Development) [176] is a methodology based on an MDA approach, although it is supported by MDE-based tools. ROOD is oriented to the development of intelligent spaces from two points of view: (1) of the contextual activities or behaviour of the resources, being these the sensors and actuators, and (2) of the intelligent object. It includes Environmental Context Models (ECMs) and Smart Object Models (SOMs). Among MDE-based tools, ROOD incorporates a UML profile known as Smart Space Modelling Language (SsML). This methodology presents 3 main stages, namely, the first stage, ECM, which is related to the MDA CIM. The second stage, SOM is represented by the MDA PIM, and finally, the PSM. At each stage, Corredor et al. [176] address the verification of model consistency and semantic consistency from the point of view of ECM or SOM with the respective viewpoints, and according to domain concepts on the knowledge base. Although ROOD has been considered a methodology that is not in accordance with ISO standards [61], it clearly exposes the work that each professional must do along the development process, including analysis, modelling and implementation.

Finally, ELDAMeth [171] is a simulation-based methodology for Distributed Agent Systems (DASs), which allows rapid prototyping based on visual programming, validation, and automatic code generation for DASs based on the Java Agent Development (JADE) framework. ELDAMeth is an iterative development process for DASs that encompasses the stages of low-level design, simulation-based validation, and JADE-oriented implementation. However, it does not present information on how the authors approach the stages of planning, elicitation, and analysis of requirements, in addition to the integration, operation, and maintenance stages.

All the development methodologies presented in this section are focused on the design and construction (automatic code generation) stages, based on a model transformation approach to obtain the IoTS software code. However, some of them [168,169,172,174–176] are specifically based on the transformation of models, while others [170,175] are based on patterns, and another [171] is based on agents.

3.2. Methodologies Designed for IoTS Development in Accordance with ISO/IEC/IEEE Standards

Fifteen documents were found presenting some development methodology for IoTSs, out of the several documents obtained from the ScDBs consulted. Table 3 shows the methodologies found. The analysis carried out on these documents was oriented to the formality of the methodologies, that is, they present the stages or processes of systems and software engineering, as well as the tools that they recommend using to obtain the deliverable product at each stage. In a quick review of the selected documents, it has been possible to determine that most of them comply below 50% of the stages that ISO/IEC/IEEE standards establish in the life cycle of systems/software [61,67,72,87–89]. For this reason, we have reviewed the literature to determine which stages should have an appropriate life cycle and if there is a consensus in the literature on this matter. Common stages in systems life cycles are (1) planning, (2) analysis (of requirements and software/system), (3) solution design, (4) solution coding and testing (construction), (5) integration and testing

(implementation), and (6) operation and maintenance [64,90,178,179]. Therefore, we have considered that these stages are the ones that a methodology should specify.

Table 3. Methodologies specifying the stages of the life cycle.

Year	Name	Bases (Methodology/Approach)	(1)	(2)	(3)	(4)	(5)	(6)	Other
2022	RASPSS [a] [180]	DDD [b]	✗	☑	✓	☑	☑	☑	☑
2021	Schauer and Falas [181]	AS [c]	✗	✓	✓	~	~	✗	✗
2020	TDDM4IoTS [d] [37]	Agile	✓	✓	✓	✓	✓	✓	✓
2019	Pico-Valencia et al. [94]	Agile (SCRUM)	±	☑	☑	✓	✓	☑	☑
2019	Gogineni et al. [86]	V Model XT	☑	✓	✓	✓	✓	✗	✗
2018	INTER-METH [65]	Iterative waterfall	~	✓	✓	✓	✓	✓	✗
2018	SERVUS [102]	SOA	✗	✓	✓	✓	✓	✗	✗
2018	Sosa-Reyna et al. [123–125]	MDD and SOA	✗	✓	✓	✓	✓	✗	✗
2017	SEM [e] [182]	Metamodel	✗	✓	✓	✓	☑	✗	✗
2017	Arrowhead [183]	SOA [f]	✗	☑	✓	✓	✓	✗	✓
2016	IDeA [g] [184]	MBSE [h], OOSEM [i]	✗	✓	✓	±	±	±	±
2015	Patel and Cassou [78]	Concerns-Oriented	☑	☑	☑	✓	✓	✓	☑
2015	Fortino et al. [185]	Metamodel	~	✓	✓	✓	✗	✗	✗
2013	AMG [38]	Model Transformation	✗	~	✓	✓	✗	✗	✗

[a] Rehabilitation Assistive Smart Product-Service System; [b] Data-Driven Development; [c] Atomic Services; [d] Test-Driven Development Methodology for IoTSs; [e] Smart Environment Metamodel; [f] Service-Oriented Architecture; [g] IoT DevProcess and AppFramework; [h] Model-Based Systems Engineering; [i] Object-Oriented Systems Engineering Method.

The analysis of the documents presenting these methodologies was able to determine the degree of compliance (expressed with an adequate clarity in the corresponding paper) of these 6 stages, whose score was as follows:

- Proper compliance. The authors have addressed all stages of the software/system life cycle, recording evidence of this, such as explaining what it consists of, and the tools to be used, among other aspects (✓).
- Incomplete compliance. The authors, although not specifically naming some stages, name some activities, tools to be used, or other aspects of those missing stages. For example, they mention the use of use case diagrams, class diagrams, or software generation from models, among others (☑).
- Legacy compliance. The authors do not explicitly name some stages because they are part of or already present in the approaches on which they are based. For example, in some cases, they mention that they are based on the fundamentals of SCRUM, which suggests that the planning stage is carried out (±).
- Inadequate compliance. Works in which the importance of some stage is mentioned, but without giving more detail on how to carry it out (~).
- Non-existent compliance. Works in which their authors do not mention activities of some stages or even do not mention certain stages (✗).

The *Other* column refers to other additional stages included in the analysed methodology, even as a consequence of having divided one or more of the 6 stages considered, as long as this is in accordance with the life cycle of the software/system presented in the ISO/IEC/IEEE standards [61,67,72,87–89] The results of this analysis are summarised in Table 2 and detailed below.

RASPSS [180] is a methodology for the design and implementation (construction) of intelligent health service processes in the context of industrial interconnection and iterative design of rehabilitation assistance devices. This methodology, in addition to being directed to a particular type of system, guides directly to the construction stage of this type of system. The architecture it presents clearly shows the importance of interaction with the user. It addresses in detail the creation of IoT devices, including providing them with the artificial intelligence techniques necessary to achieve their objectives.

The work of Schauer and Falas [181] focuses on the development of atomic services. Services are deployed in a service repository according to the types of computing resources that are available. That repository has been designed to provide two functionalities: (1) the list of services with the appropriate descriptions; and (2) pluggable Docker images ready to use in different architectures based on atomic services. Its architecture is based on a repository of Docker image forms and controlled by the service orchestration process. The atomic services are developed following a life cycle with the stages of design, development (a stage of this methodology), deployment, and execution. In addition, IoTSs are treated as complex systems and are carried out through service composition. A third issue addressed refers to the components related to computational resources (processing, storage, and so on), and communications (protocols, technologies, and so on). This methodology is focused on the stage of IoTSs construction and is closed to the technologies. These may be the reasons why the planning, integration, and maintenance stages are not mentioned. Other activities are assumed to take place in what they call the development stage.

TDDM4IoTS [37] is a methodology based on the 4 values and 12 principles of the Agile Manifesto. It considers 11 stages and specifies, in an acceptable way, the resources and tools to be used in each of the stages. It is the most attached to the software/systems life cycle raised in the ISO/IEC/IEEE standards [61,67,72,87–89]. TDDM4IoTS raises the stages in a distinctive way, involving those aspects of IoTSs that differentiate them from the traditional ISs, except that it does not address those activities that involve the provision of AI techniques to an IoTS, as well as the withdrawal of the IoTS once it cannot fulfil the functions for which it was designed, or its maintenance is unfeasible.

Another methodology based on the principles and values of agile methodologies is the one presented by Pico-Valencia et al. [94]. This methodology is focused on the construction of IoTSs. It presents the requirements elicitation stage by collecting the global requirements, being the work team who collects these requirements from users, clients, and other stakeholders. The design stage consists of, once these requirements have been defined, carrying out a set of tasks linked to very specific agents. More specifically, an exploration process of the IoT infrastructure is carried out, where the objects connected to the network are modelled as Linked Open Agents (LOAs). In addition, tasks related to the infrastructure itself are carried out (interception of messages, adding new messages, workflow execution, and agent discovery). The construction and integration stages are presented as a macroscopic level stage in which the microscopic level LOAs are integrated and coordinated within a single network. Moreover, its authors have designed this stage to meet the requirements of LOA interaction, collaboration, coordination, data processing, user interaction and intelligent behaviour. It should be noted that this methodology mentions the use of tools that the developer can use in the analysis and design stages. However, there is no evidence of the planning, implementation, operation, and maintenance stages.

Among the contributions of Gogineni et al. [86] is a methodology for the development of IoTSs that follows the V-model XT. In it, the verification and validation of the requirements, functionalities, and principles governing the system is the focus of its considerations. Other important stages are the requirements elicitation, design (in some respects), integration, and testing stages. However, they do not present evidence to address the operation and maintenance stage.

INTER-METH [65] can be considered an iterative methodology that defines the six sequential stages of development: analysis, design, implementation, deployment, testing, and maintenance. Each iteration can be geared towards improving individual stages, a set of successive stages, or the entire process, thus improving adaptability to new requirements. INTER-METH is a methodology adapted from the traditional waterfall methodology, differing in that it divides the problem into subproblems to ensure successful development. INTER-METH, being based on the waterfall methodology, is supposed to meet each of the characteristics of its base methodology. However, its authors do not present tools, guidelines, activities, or tasks typical of IoTSs, such as hardware deployment.

Usländer and Batz present SERVUS [102] as an IoTS development methodology aimed at solving interoperability challenges by adopting a service-oriented architecture based on the Industrial Internet Reference Architecture (IIRA 4.0). SERVUS addresses requirements elicitation and analysis, as well as the analysis and design stages. To obtain the requirements and their analysis, they recommend user stories and use cases as the main artifacts for this stage. However, there is no supporting evidence for the software construction, deployment, or operation and maintenance stages.

Sosa-Reyna et al. present a methodology with two approaches, i.e., with the MDD [124,125] and MDE [123] approaches. This methodology establishes the following stages of development: (1) Analysis of business requirements, (2) definition of the business logic, (3) design of the integrated services solution, (4) generation of the technological solution, and (5) model transformation methods. They leverage the MDE guidelines and use languages such as Unified Modelling Language (UML) to specify business requirements in stage (1), and Business Process Model and Notation (BPMN) for the definition of the business logic in stage (2). The result of stages (1) and (2) is a PIM. Subsequently, a more refined model is obtained following an SOA approach in stage (3). To obtain a platform-specific code in stage (4), two steps are performed: First, a PSM is derived from the PMI, and second, the PSM is transformed into a code. This is one of the IoTS development methodologies that presents and defines all the basic stages for the development of this type of system. However, its authors do not consider how to develop user interfaces. Moreover, they present a set of tools for capturing and analysing FRs of IoTSs, but NFRs are not considered.

The SEM methodology [182] is based on metamodels. The modelling is done from two points of view: from the functions that the system must fulfil and from the data with which it works. It presents 3 stages: the Requirements Analysis stage, in which the provided metamodel is used to obtain the model in the Design stage, and finally, the System Implementation stage. SEM is another of the methodologies focused on the construction of IoTSs, so it focuses on the necessary aspects to obtain the final product with the requirements demanded by the users. However, it only mentions important stages, such as planning, requirements elicitation, implementation, and operation and maintenance of the IoTS. Moreover, it is a specific methodology for a specific domain.

Arrowhead [183] is presented as a framework. The design, development, and verification methodology for each service, system, and system of systems within the Arrowhead framework supports that these can be implemented, verified, deployed, and executed in an interoperable manner. Arrowhead helps in the construction of IoTSs from the perspective that the system can be built by integrating other systems. Although its authors mention the stages of the software/system life cycle a lot, they do not present evidence to guide developers to carry out their activities.

Costa et al. present IDeA [184] as a methodology for the development of IoTSs, based on model-based systems engineering (MBSE). The method provided is based on existing standards to which activities considered the most relevant for the design of IoT applications are added. IDeA provides high-level abstractions through metamodeling as a possible solution to the (hardware and software) heterogeneity problem. In addition, their IoT application design method is a multidisciplinary method, in which all the stakeholders involved in the process participate. They apply the ISO/IEC/IEEE 15288:2015 standard, from which they implement the systems' life cycle processes, although they do not specify all their phases. Although IDeA refers to the ISO standard, it does not provide guidelines for its application. In addition, there is no evidence that deals with the planning stage and, being a methodology based on metamodels, forces developers to stick to them.

The methodology proposed by Cicirelli et al. [182] is based on both functional (functions and services) and data (data sources, attributes, and relationships) metamodels, and is focused on the design of IoTSs. Its authors assume the requirements elicitation as a preliminary stage and focus directly on the system modelling. By being based on metamodels, it raises stereotypes for: (1) the environment in which the system is going to work, (2) the functionalities that an intelligent environment can offer, which can be atomic or composite.

In the metamodels, this methodology specifies the processes that will make the environment an intelligent environment. To use a metamodel, the problem must be aligned to the metamodel to be applied in the solution. The methodology is clearly oriented towards the design stage, but there is no evidence that this methodology addresses requirements elicitation and analysis. It also does not address the IoTS development planning or final stages, such as implementation, operation, and maintenance. In addition, the stage of construction or obtaining the final product is not clear.

Patel and Cassou [78] focused on the roles of team members to try to solve the problems of heterogeneity of technologies with which IoT applications can be implemented. Domain experts and software designers oversee the system analysis and design stage activities. Application programmers and device developers are engaged in the system construction and testing stage. Finally, network administrators install the application on the system in question, which fits with the implementation and deployment stages of other methodologies. This methodology does not address the planning, operation, and maintenance stages.

Fortino et al. [185] propose a metamodel-based engineering approach for the systematic development of smart objects (SO). The analysis stage deals with the modelling of relevant aspects of SOs using a metamodel. The design stage tries to model the functional components of the system, their relationships, and interactions using the smart object model, based on an ELDA framework [177] and an ACOSO platform [186]. To support the implementation stage, the metamodel of smart objects based on ACOSO has been specialised with respect to the JADE platform [187], resulting in the JACOSO metamodel [188]. This metamodel highlights the components of the JADE platform (people, their behaviour, and messages). Other elements of the metamodel represent the tasks to be respectively carried out by the person responsible for configuring the system, the communication manager, and the device manager, as well as the user-defined tasks that are included in the inference rules, which control the behaviour of smart objects. To use this metamodel, the developer must master the frameworks and platforms named above.

AMG [38] is an IoT application development methodology based on SOA consisting of three steps: (1) definition of abstractions, (2) modelling, and (3) code generation. AMG is based on a bottom-up approach since it starts from concrete models (concrete services) to obtain abstract services. The abstraction process starts from the descriptions of the services, in such a way that first the necessary graphic representations are obtained and then the source code is obtained.

Some authors, such as Gogineni et al. [86], Wang et al. [180], and Fortino et al. [185], consider requirements analysis as part of the system analysis stage, which constitutes the reason why they do not address the requirements elicitation. However, other authors, such as Guerrero-Ulloa et al. [37], Usländer and Batz [102], and Sosa-Reyna et al. [123–125], separate these stages very well and give them the importance they require, since the system requirements must be clear enough from the very beginning so that the development of the IoTS is not delayed. It could be concluded that the lack of consideration of the planning stage is widespread since only Guerrero-Ulloa et al. [37] expressly specify it, while Gogineni et al. [86] only give hints of its consideration, and Fortino et al. [65] only let us guess its consideration, since its proposal is based on waterfall methodology, which does contemplate this stage.

Another stage not mentioned by most researchers who have designed development methodologies is the joined operation and maintenance stage (i.e., a unique stage combining operation and maintenance activities). Varga et al. [183] and Guerrero-Ulloa et al. [37] present the maintenance stage as part of their respective methodologies. However, Wang et al. [180] and Pico-Valencia et al. [94] address this stage briefly. It can be assumed that Costa et al. [184] address this stage due to the approach its methodology (IDeA) relies on. Although Schauer and Falas' [181] proposal is considered to address design, the work is focused on presenting an adaptive architecture for IoTSs.

Among the methodologies that have addressed the obtaining of software code is AMG [38], Schauer and Falas' [181] one, and SEM [182], particularly in the modelling and/or obtaining of software for the IoT device, ignoring the applications that will serve for the interaction between the user and the IoTS [38,174]. Although Wang et al. [180], Pico-Valencia et al. [94], Schauer and Falas [181], and Fortino et al. [185] address and mention user-centred design, they do not present evidence of addressing the construction of end-user applications. The authors mentioned focus on obtaining the hardware component of the IoTS and the software for its configuration.

4. Other Proposals for the Development of IoTSs

There are many contributions oriented to facilitate the development of IoTSs. Some of them were already analysed in previous sections. Table 4 presents the main features of additional platforms, frameworks, and tools for the development of IoTSs.

Table 4. Main features of platforms, frameworks, and tools for the development of IoTSs.

Year	Reference Name	Approach	Artifacts	Final Product
2021	Autolink [189]	Components, Templates, MDD	Code, Device design diagram	IoT device: software and design
2020	TinyLink [190]	Components, Templates, MDD	Code, Device design diagram	IoT device: software and design
2018	PrIoT [154]	Components	PrIoT-core, PrIoT-API, Prior-Test, Prior-DB, PrIoT-UI	Conceptual framework
2018	Cai et al. [191]	MDA, patterns, and ontologies	BPMN, CD, AD	Software
2017	COMFIT [192]	MDA	Components	Code in NesC and C languages
2017	EDG [193]	Components (APIs), MDD	Code, BD [a]	IoT device: software and design

[a] Block Diagram.

Pawar et al. [154] propose PrIoT, a development framework that proposes to group third-party solutions to meet certain goals. The framework is composed of three modules to try to integrate the different components that make up an IoTS. PrIoT-core makes the work of the developers independent of the devices and the details of the communication protocols. PrIoT-Lang deploys a device-independent programming interface. PrIoT-API captures the most practical IoT scenarios in a limited way. PrIoT-Test allows to specify metrics to test the validity of the results obtained from the prototype. PrIoT-DB allows us to select the necessary hardware either from a specific or generic supplier and include libraries in the project, for example, to enable information exchange with other devices. The components are configured through a file. PrIoT-UI introduces a high-level graphical interface to support developers.

Cai et al. [191] propose a mobile application deployment framework based on MDA. Development patterns based on semantic reasoning are provided to target the development of Cloud of Things applications (CoT) [194] in a configurable and adaptable way. They also provide a metamodel with multi-view business components and service components for exchanging models. They follow the MVC (Model-View-Controller) pattern to transform business models into a service component to configure cloud services.

COMFIT [192] is an Integrated Development Environment (IDE) based on MDD and Cloud Computing. COMFIT consists of two modules: (1) a development module for designing IoTSs using adaptive models of high abstraction; and (2) a set of web-based management and execution applications, capable of generating code aimed at the Contiki and TinyOS platforms from the models obtained in the design stage. It is worth mentioning that COMFIT does not address how to develop applications for the end user.

EDG [193] is a methodology for the generation of integrated designs. To do this, the user is required to specify with simple requirements their embedded software. From that specification (hardware-independent code), a synthesised final circuit diagram is produced, a list of required materials and the firmware necessary for the device to meet the requirements. Finally, the hardware configuration diagram allows us to develop the physical device and write the hardware-dependent code.

TinyLink [190] is a tool that follows the EDG methodology [193]. TinyLink is aimed at developers with no experience in embedded systems, which may have both hardware and user constraints. TinyLink seeks to optimise hardware use through a set of APIs to allow developers to carry out a bottom-up development process, in contrast with traditional tools, which usually follow a top-down approach. In addition, TinyLink can generate a hardware-dependent code. Similarly, Autolink [189] is a tool based on TinyLink [190]. In addition to generating TinyLink's functional solutions, Autolink addresses NFRs such as estimating the useful lifetime of the device's hardware components, execution time, and optimising battery use.

All these works support developers when it comes to having the software code for the configuration of the IoT hardware. However, there is no evidence that it has been addressed how to obtain the software code for applications that will serve as a means of interacting with the end user. In addition, the presented tools need previously elicited requirements as inputs for the design stage and subsequent code generation.

5. Architectures for IoTSs

The architecture of any system serves as a guide in its development process. Thus, it is important to know the architectures that the researchers have proposed to develop IoTSs, as well as if the methodology they present for the development of IoTSs is based on some architecture, and which is the most frequent. Among the most frequent architectures are layered and service-oriented architectures. In addition, some authors [123–125,170] present a combination of both types, while other authors [78,94,139,169,195,196] present their own architectures to give solutions to different types of IoTSs.

5.1. Layered Architecture

Some existing IoTSs have been developed following a layered architecture, such as the one presented by Vashi et al. [197], which consists of five layers (see Figure 6a). Guerrero-Ulloa et al. [22] present a very similar architecture that also consists of five layers, as shown in Figure 6b. As can be observed, there are similarities between both architectures. Other similar architectures only have four layers [65,151,198]. The difference between these last architectures is the order of their layers. In some of them [65,151], their layers are (1) device, (2) network, (3) middleware, and (4) application, while in the other [198], its layers are: (1) perception, (2) middleware, (3) services, and (4) applications. Consequently, the lower layers (1) and the upper layers (4) have the same objectives, while the second and third layers are exchanged regarding the ones of the other two architectures.

Unlike the architectures of the works, the RapIoT toolkit [162,163] and the Mobile Health Platform [160] are based on a three-layer architecture, where two of the layers fulfill similar functions (device management and applications) in both architectures, only being different in the third layer (communications). In the former [162,163], this layer fulfils the functions of a cloud service, which allows data storage and integration with third-party services, while in the latter [160], it consists of the middleware (application server) and the web application that interconnects the various objects of the physical layer with other actors (health professionals, hospitals, and other systems) [162,163]. RapIoT layers [162,163] are named as an *embedded layer*, *gateway layer,* and *server layer*, and in the Mobile Health Platform [160] as a physical-objects layer, network layer, and health portal. The third layer performs very similar functions to the middleware layer of Fortino et al. [65] and Sharma et al.'s [151] architectures, the services layer of Qiang et al.'s [198] architecture, and the cloud processing layer of Guerrero-Ulloa et al.'s [22] architecture.

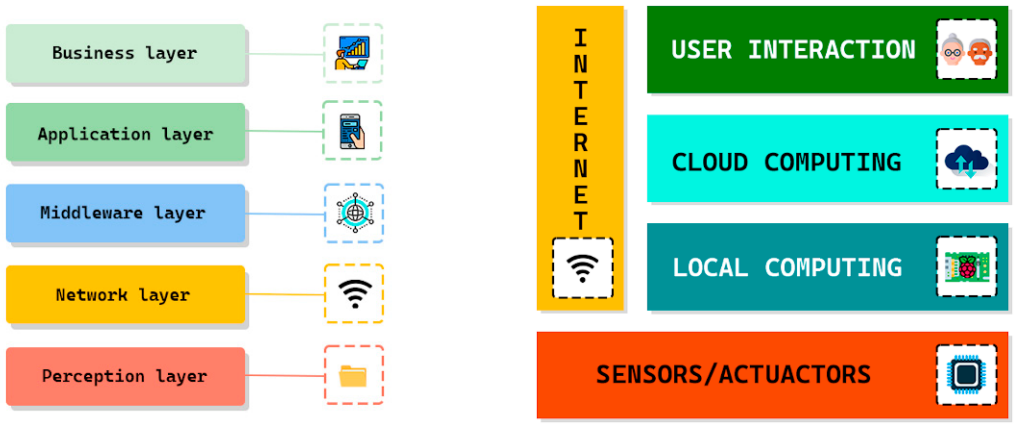

Figure 6. Support architectures of MFPTG4IoTDs for IoTS development. (**a**) Vashi et al.'s architecture [197]. (**b**) Guerrero-Ulloa et al.'s architecture, adapted with permission from Ref. [22]. 2022, Gleiston Guerrero-Ulloa, Carlos Rodríguez-Domínguez, Miguel J. Hornos.

Nugra et al. [57] present an IoTS to manage urban traffic, implemented following a three-layer client/server architecture. The *client* layer consists of applications for end users, whose input data are basically provided by Pentaho BI. The *business* layer is made up of APIs that provide weather data, in addition to the *Pentaho BI server layer* and the web application. They use two database managers: MySQL, which stores data captured by sensors and weather forecasts, and PostgreSQL, which is where data is copied, from time to time, to form OLAP cubes to allow data analysis.

ELDAMeth [171] uses ELDASim, which is an ELDA simulation environment, which is based on a four-layer architecture: (1) A configuration layer allows setting up its components through the MAS (Multi-Agent System) simulation module; (2) an agent layer provides adaptations of ELDA agents on agent servers; (3) the platform (i.e., the agent servers, the network that interconnects them, the signalling messages between agent servers and various types of systems, and the coordination infrastructures to fully support the distinctive multi-coordination feature of the ELDA model), which defines a distributed infrastructure consisting of a network of agent servers supporting ELDA agents; and (4) an engine that provides the key mechanisms for the simulation of discrete events of general purpose systems.

The methodology presented by Usländer and Batz [102] is based on IIRA v1.9 [199]. It defines a three-level scheme: (1) The edge level collects data from edge nodes, using the proximity network. The architectural features of this level, including the breadth of distribution, location, the scope of governance, and nature of the proximity network, vary depending on the specific use cases. (2) The platform level receives, processes, and forwards control commands from the enterprise level (explained below) to the edge level. It consolidates processes and analyses edge-level and non-edge data flows, as well as provides device and asset management capabilities. It also offers non-domain-specific services, such as querying and analysing data. (3) The enterprise tier implements domain-specific applications, decision support systems, and provides interfaces to end users, including operation specialists. This level receives data streams from the edge and platform levels, and issues control commands directed at both [200].

In the work by Lekidis et al. [168], the system design is specified in a domain-specific language (DSL), which they use to maintain the match between the automatically generated BIP (Behaviour, Interaction, Priority) model and the application code. BIP is a language with formally defined semantics for constructing executable models of mixed software/hardware systems (SW/HW). The BIP model is based on the standards of the WPAN architecture. This mixed architecture consists of four layers of abstraction, where the lower layer (1) is defined by the abstraction of the hardware architecture, the upper

layer (4) is the abstraction of the software application, the third layer is defined by the operating system, and the second layer is defined by the network stack and device drivers.

5.2. Service-Oriented Architectures

SOA is used to build software systems from composite, heterogeneous, and autonomous software units, called services. In addition, service composition is a common approach to the development of complex software systems. Software systems and applications in turn are also becoming services. We call this service-based systems or applications [38].

In some of the works considered in this review, Sosa-Reyna et al. [123–125] propose an architecture based on SOA, consisting of four layers: (1) object layer (hardware objects available on the network), (2) network layer (wired, wireless or mobile connection infrastructure), (3) service (creation and management of required services), and (4) application layer (responsible for delivering applications to IoTS users). This service-oriented architecture supports development methodologies with two different approaches: MDD [124,125] and MDE [123].

Another work with this type of mixed architecture is defined by Brambilla et al. [170], with basically 3 layers: the client or front-end layer, the communications layer or API Gateway, and the server or back-end layer. This last layer is in turn defined by microservices, which provide information for user management, group concepts related to both the organisational structure of the actors and the definition of things and allow clients to access both data values and graphic resources.

The method for the development of IoTSs proposed by Sulistyo [38] is SOA-based. In the first instance, this author considers the existence of concrete services to abstract the abstract services to model the system in question. After obtaining the model, it generates the code for the new service-based application. This type of architecture (SOA) significantly reduces the complexity in the design of a heterogeneous system, such as an IoTS [168]. Another SOA-based work is the Arrowhead framework [183], where operations on different resources can be grouped into different services. In Arrowhead, a resource could be a temperature sensor or the energy consumption reading of an energy meter.

5.3. Other Types of Architectures

MDE4IoT [172] is based on the MARTE architectural model [201], which includes 3 packages: (1) MarteFoundations, which defines all the basic concepts required for the analysis and model-based design of real-time and embedded systems (RT/ESs); (2) MarteDesignModel, which covers from requirements capture to requirements specification, design and implementation (V-cycle development process [202]); and (3) MarteAnalysisModel, which defines specific model abstractions and relevant annotations to be used by external tools. Therefore, package (1) defines general concepts for quantitative analysis techniques, which are extensible to support new RT/ES UML model analysis techniques, while packages (2) and (3), respectively, focus on programmability and performance analysis.

The ROOD architecture [176] is based on MDA. It includes four levels of abstraction, ordered from the highest to the lowest level of abstraction as follows: (M3) Meta-Metamodel layer, which serves to establish the basis for different metamodels; (M2) Metamodel layer, where DSLs are specified to define models at the M1 level; (M1) Model layer, where system models are defined; and (M0) Instance layer, which contains instances of data for a given platform.

The COMFIT architecture [192] includes two modules: (1) Application Development Module (ADM), and (2) Application Management and Execution Manager (AMEM). ADM includes PIM and PSM models, as well as M2M transformations and code generation templates to use M2T transformations. On the other hand, AMEM includes: (1) the Interface Manager component, which is directly connected to ADM and is responsible for providing the functionality of uploading the generated code to a server hosted on the Cloud; and (2) the Execution Manager, which is connected to both the Testbed Manager and the Compile Manager and deploys the services that are released through the Interface Manager.

The framework proposed by Cai et al. [191] is based on software architecture for mobile service development, consisting of three modules: (1) an information module for device encapsulation, which supports multiple business modelling views; (2) an ejection environment, used to configure the application environment and the execution rules of a business application; and (3) a resource repository for information configuration, which is designed to connect the information modeller and the execution environment.

In the SDG-Pro framework architecture [203], components are classified according to their Internet connectivity: Edge or Cloud components. Edge components are sensors, actuators, and communication devices. Communication devices are connected to Cloud components through software-defined gateways that are part of Cloud components. In addition, as part of the Cloud components, we find the Information Technology components for data storage, processing, and intelligent analysis.

Another component-based architecture is the one presented by Schauer and Falas [181]. The first component is the IoT system builder. The second component is that of computational resources or a server, which is the one that interacts with feeding or receiving data from the other components. The server executes business logic when requested by Docker Engine clients and has the monitoring tools to interact with the supervisor component. The third component consists of the Docker Engine, which contains the technology for message exchange (RabbitMQ server). The fourth component is the IoT systems supervisor, which monitors resources and detects system failures. This last component interacts with the first one to update the monitored system according to what is measured/detected by the sensors.

The SEM methodology [182] is supported by an architecture based on models focused on software construction. Its authors present a functional metamodel and a data metamodel. Another methodology that is supported by this type of architecture is IdeA [184], where the IoT application engineer breaks down the system into functional components that interact with each other to meet system requirements. IDeA models expose devices as services using port notation. These services will be used by application engineers to create information views [184]. In this same line, we can mention Fortino et al.'s proposal [185], which is based on metamodels for smart objects as a very high-level metamodel that specifically exposes static and dynamic characteristics of smart objects. An ELDA-based metamodel specialises in the high-level smart object metamodel, providing the functional components of the system, their relationships, and interactions. And the ACOSO-based metamodel is a middleware specifically designed for the complete management of cooperating and agent-oriented smart objects.

Chauhan et al. [175] propose a publish-subscribe architecture in which sensors act as publishers. Computer services are subscribers that make actuators execute the corresponding actions (changes in the environment, notifications to end users, and so on) whenever new data is received.

The architecture presented by Harbouche et al. [169] is a wireless body sensor network architecture based on a mobile data collector. The architecture presented by Alvear-Puertas et al. [204] is similar. In fact, both are oriented to explain the operation and deployment of the system. They also provide feedback to the users about the outcomes of the requirements analysis stage. On the other hand, Wang et al. [180] present an architecture that can guide the development of an IoTS and understand how it works. In some works [37,86,174], the architecture of the IoTSs presented by their authors has not been addressed, only focusing on explaining its operation.

6. Conclusions

We have presented a comprehensive study on the methodologies as well as frameworks, tools, and architectures that could be applied to develop IoTSs. The main conclusions of the analysis carried out following the review of the existing proposals so far are set out below.

This article has discussed important aspects and steps that existing methodologies address or that a methodology for IoTS development should address, and these outline guidelines for implementation. An important aspect that differentiates IoTSs from other ISs is that they are

made up of objects (*things*) that interact autonomously with each other, considering people as other objects or *things* of the system. To do this, IoTSs need to rely on sensor/actuator networks and efficient wireless communications [34,35]. Consequently, for the development of an IoTS, it is important to analyse existing and available technology, and even develop the necessary devices, if possible, to help meet not only FRs but also NFRs [32,35].

Model-based approaches, i.e., MDE, MDD, and MDA, are among the most widely used methodologies for IoTS development. However, very few of these methodologies deal with the elicitation and analysis of requirements, and except for one of them, no other deals with the maintenance phase, and none the withdrawal phase. Other aspects that most of the methodologies reviewed do not address or do not mention are those related to NFRs. The success of applying these approaches may be because they help to solve the technology heterogeneity problem involved in IoTS development.

We consider that the software development process should cover each of the stages/phases of the system life cycle, from the elicitation and analysis of requirements to its dismantling. However, with this type of system, as in others in some cases, the client is not clear about the requirements at the beginning, so it is considered that developers must present early prototypes. For this, model-based approaches must be present for the generation of software quickly, and therefore also RP. In addition, RP and agile methodologies are the most widely used in IoTS development. Therefore, SE researchers must design a methodology for the development of IoTSs that collects the best practices of model-based approaches (i.e., MDD, MDE, and MDA), PR, and the 4 values and the 12 principles of the agile manifesto on which agile methodologies (such as Scrum and XP, among others) are based.

None of the IoTS development methodologies proposed in the literature and presented in this document have been used by developers or researchers other than their own authors. Most documents that present the development of some IoTSs apply Scrum as the only methodology, and some of them combine it with others, such as XP, RP, and Kanban.

The development frameworks used for IoTS development are mostly concerned with modelling and code generation for IoT devices, not addressing the generation of applications that offer an interface with the end user. We think that these development frameworks should also provide support for the development of such applications. An important advance in this area would be to create a tool that facilitates the work of IoTS developers in such a way that it becomes universally used by all of them. To achieve that goal, said tool should consider the functionalities of some of the tools presented in this review and add others that none of them currently have.

The most common architectural style in all the works that have been reviewed is the layer-based one. Usually, data processing occurs from the lower layer, known as the physical, perception, or sensors/actuators (among other names) layer, to the highest one, known as an application, user interaction, user, or cloud processing (among other names) layer, and the interconnection must be present between all its layers. The architecture serves as a guide to support the work of developers to fulfil the FRs and NFRs of the system, and therefore to obtain quality software.

Author Contributions: Conceptualization, G.G.-U., C.R.-D. and M.J.H.; methodology, G.G.-U., C.R.-D. and M.J.H.; software, G.G.-U.; validation, G.G.-U.; formal analysis, G.G.-U.; investigation, G.G.-U.; resources, G.G.-U., C.R.-D. and M.J.H.; data curation, G.G.-U.; writing—original draft preparation, G.G.-U.; writing—review and editing, C.R.-D. and M.J.H.; visualization, G.G.-U.; supervision, C.R.-D. and M.J.H.; project administration, G.G.-U., C.R.-D. and M.J.H.; funding acquisition, C.R.-D. and M.J.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Spanish Ministry of Science and Innovation (State Research Agency), grant number PID2019-109644RB-I00, and Junta de Andalucía (Andalusian Regional Government), grant number B-TIC-320-UGR20.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available due to privacy restrictions.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Royce, W.W. Managing the Development of Large Software Systems: Concepts and Techniques. In Proceedings of the 9th International Conference on Software Engineering, ICSE '87, Monterey, CA, USA, 30 March–2 April 1987; IEEE Computer Society Press: Monterey, CA, USA, 1987; pp. 1–9, ISBN 0897912160.
2. Boehm, B.W. A Spiral Model of Software Development and Enhancement. *Computer* **1988**, *21*, 61–72. [CrossRef]
3. Lantz, K.E. *The Prototyping Methodology*; Prentice-Hall: Saddle River, NJ, USA, 1986; ISBN 978-0-8359-5897-4.
4. Fern, D.A.; Donaldson, S.E. Tri-Cycle: A Prototype Methodology for Advanced Software Development. In Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences; IEEE Computer Society: Kailua-Kona, HI, USA, 1989; Volume 2, pp. 377–386. [CrossRef]
5. Abrahamsson, P.; Salo, O.; Ronkainen, J.; Warsta, J. Agile Software Development Methods: Review and Analysis. *arXiv* **2002**. [CrossRef]
6. Anwer, F.; Aftab, S.; Waheed, U.; Muhammad, S.S. Agile Software Development Models TDD, FDD, DSDM, and Crystal Methods: A Survey. *Int. J. Multidiscip. Sci.* **2017**, *8*, 1–10. Available online: http://www.ijmse.org/Volume8/Issue2/paper1.pdf (accessed on 21 October 2022).
7. Srivastava, A.; Bhardwaj, S.; Saraswat, S. SCRUM Model for Agile Methodology. In Proceedings of the IEEE International Conference on Computing, Communication and Automation, ICCCA 2017, Greater Noida, India, 5–6 May 2017; IEEE: Greater Noida, India, 2017; Volume 2017, pp. 864–869. [CrossRef]
8. Schwaber, K.; Sutherland, J. The Scrum Guide. *Acrum Allience* **2011**, *21*, 1.
9. Salo, O.; Abrahamsson, P. Agile Methods in European Embedded Software Development Organisations: A Survey on the Actual Use and Usefulness of Extreme Programming and Scrum. *IET Software* **2008**, *2*, 58–64. [CrossRef]
10. Holzinger, A.; Errath, M.; Searle, G.; Thurnher, B.; Slany, W. From Extreme Programming and Usability Engineering to Extreme Usability in Software Engineering Education (XP+UE→XU). In Proceedings of the International Computer Software and Applications Conference, Edinburgh, UK, 26–28 July 2005; IEEE: Edinburgh, UK, 2005; Volume 2, pp. 169–172. [CrossRef]
11. Stott, W. Extreme Programming: Turning the World Upside Down. *IEE Comput. Control. Eng.* **2003**, *14*, 18–23. [CrossRef]
12. Schwabe, D.; Rossi, G. Building Hypermedia Applications as Navigational Views of Information Models. In Proceedings of the Annual Hawaii International Conference on System Sciences 1995, Maui, HI, USA, 3–6 January 1995; Volume 3, pp. 231–240. [CrossRef]
13. Schwabe, D.; Rossi, G. The Object-Oriented Hypermedia Design Model. *Commun. ACM* **1995**, *38*, 45–46. [CrossRef]
14. Garzotto, F.; Paolini, P.; Schwabe, D. HDM—A Model-Based Approach to Hypertext Application Design. *ACM Trans. Inf. Syst.* **1993**, *11*, 1–26. [CrossRef]
15. Garzotto, F.; Mainetti, L.; Paolini, P. Navigation Patterns in Hypermedia Data Bases. In Proceedings of the Annual Hawaii International Conference on System Sciences, Wailea, HI, USA, 5–8 January 1993; IEEE: Wailea, HI, USA, 1993; Volume 3, pp. 370–379. [CrossRef]
16. Lange, D.B. Object-Oriented Hypermodeling of Hypertext Supported Information Systems. In Proceedings of the Annual Hawaii International Conference on System Sciences, Wailea, HI, USA, 5–8 January 1993; IEEE Computer Society: Wailea, HI, USA, 1993; Volume 3, pp. 380–389. [CrossRef]
17. Lange, D.B. Object-Oriented Design Method for Hypermedia Information Systems. In Proceedings of the Hawaii International Conference on System Sciences, Wailea, HI, USA, 4–7 January 1994; IEEE: Wailea, HI, USA, 1994; Volume 3, pp. 366–375. [CrossRef]
18. Lange, D.B. An Object-oriented Design Approach for Developing Hypermedia Information Systems. *J. Organ. Comput. Electron. Commer.* **2009**, *6*, 269–293. [CrossRef]
19. Isakowitz, T.; Stohr, E.A.; Balasubramanian, P. RMM: A Methodology for Structured Hypermedia Design. *Commun. ACM* **1995**, *38*, 34–44. [CrossRef]
20. Bieber, M.P.; Isakowitz, T. Introduction to the Special Issue: Hypermedia in Information Systems and Organizations. *J. Organ. Comput. Electron. Commer.* **2009**, *6*, 3–7. [CrossRef]
21. Singh, D.; Tripathi, G.; Jara, A.J. A Survey of Internet-of-Things: Future Vision, Architecture, Challenges and Services. In Proceedings of the 2014 IEEE World Forum on Internet of Things, WF-IoT 2014, Seoul, Republic of Korea, 6–8 March 2014; IEEE Computer Society: Seoul, Republic of Korea, 2014; pp. 287–292. [CrossRef]
22. Guerrero-Ulloa, G.; Rodríguez-Domínguez, C.; Hornos, M.J. IoT-Based System to Help Care for Dependent Elderly. *Commun. Comput. Inf. Sci.* **2019**, *895*, 41–55. [CrossRef]
23. Dado, M.; Janota, A.; Spalek, J. Challenges and Unwanted Features of the Smarter Cities Development. *Lect. Notes Inst. Comput. Sci. Soc.-Inform. Telecommun. Eng. LNICST* **2015**, *151*, 3–8. [CrossRef]
24. Madakam, S.; Ramaswamy, R.; Tripathi, S. Internet of Things (IoT): A Literature Review. *J. Comput. Commun.* **2015**, *3*, 164–173. [CrossRef]
25. Stankovic, J.A. Research Directions for the Internet of Things. *IEEE Internet Things J.* **2014**, *1*, 3–9. [CrossRef]

26. Santos, P.M.; Rodrigues, J.G.P.; Cruz, S.B.; Lourenço, T.; D'Orey, P.M.; Luis, Y.; Rocha, C.; Sousa, S.; Crisóstomo, S.; Queirós, C.; et al. PortoLivingLab: An IoT-Based Sensing Platform for Smart Cities. *IEEE Internet Things J.* **2018**, *5*, 523–532. [CrossRef]
27. Pan, J.; Jain, R.; Paul, S.; Vu, T.; Saifullah, A.; Sha, M. An Internet of Things Framework for Smart Energy in Buildings: Designs, Prototype, and Experiments. *IEEE Internet Things J.* **2015**, *2*, 527–537. [CrossRef]
28. Ta-Shma, P.; Akbar, A.; Gerson-Golan, G.; Hadash, G.; Carrez, F.; Moessner, K. An Ingestion and Analytics Architecture for IoT Applied to Smart City Use Cases. *IEEE Internet Things J.* **2018**, *5*, 765–774. [CrossRef]
29. Ng, I.C.L.; Wakenshaw, S.Y.L. The Internet-of-Things: Review and Research Directions. *Int. J. Res. Mark.* **2017**, *34*, 3–21. [CrossRef]
30. Alaba, F.A.; Othman, M.; Hashem, I.A.T.; Alotaibi, F. Internet of Things Security: A Survey. *J. Netw. Comput. Appl.* **2017**, *88*, 10–28. [CrossRef]
31. Guerrero-Ulloa, G.; Hornos, M.J.; Rodríguez-Domínguez, C.; Fernández-Coello, M.M. IoT-Based Smart Medicine Dispenser to Control and Supervise Medication Intake. In Proceedings of the Intelligent Environments 2020: Workshop Proceedings of the 16th International Conference on Intelligent Environments 2020, Madrid, Spain, 20–23 July 2020; pp. 39–48. [CrossRef]
32. Matias, I.; Garcia, N.; Pirbhulal, S.; Felizardo, V.; Pombo, N.; Zacarias, H.; Sousa, M.; Zdravevski, E. Prediction of Atrial Fibrillation Using Artificial Intelligence on Electrocardiograms: A Systematic Review. *Comput. Sci. Rev.* **2021**, *39*, 100334. [CrossRef]
33. Han, T.; Zhang, L.; Pirbhulal, S.; Wu, W.; de Albuquerque, V.H.C. A Novel Cluster Head Selection Technique for Edge-Computing Based IoMT Systems. *Comput. Netw.* **2019**, *158*, 114–122. [CrossRef]
34. Dayo, Z.A.; Aamir, M.; Dayo, S.A.; Khoso, I.A.; Soothar, P.; Sahito, F.; Zheng, T.; Hu, Z.; Guan, Y. A Novel Compact Broadband and Radiation Efficient Antenna Design for Medical IoT Healthcare System. *Math. Biosci. Eng.* **2022**, *19*, 3909–3927. [CrossRef] [PubMed]
35. Memon, S.K.; Nisar, K.; Hijazi, M.H.A.; Chowdhry, B.S.; Sodhro, A.H.; Pirbhulal, S.; Rodrigues, J.J.P.C. A Survey on 802.11 MAC Industrial Standards, Architecture, Security & Supporting Emergency Traffic: Future Directions. *J. Ind. Inf. Integr.* **2021**, *24*, 100225. [CrossRef]
36. Guarda, T.; Leon, M.; Augusto, M.F.; Haz, L.; de La Cruz, M.; Orozco, W.; Alvarez, J. Internet of Things Challenges. In Proceedings of the Iberian Conference on Information Systems and Technologies, CISTI, Lisbon, Portugal, 21–24 June 2017; IEEE: Lisbon, Portugal, 2017. [CrossRef]
37. Guerrero-Ulloa, G.; Hornos, M.J.; Rodríguez-Domínguez, C. TDDM4IoTS: A Test-Driven Development Methodology for Internet of Things (IoT)-Based Systems. *Commun. Comput. Inf. Sci.* **2020**, *1193*, 41–55. [CrossRef]
38. Sulistyo, S. Software Development Methods in the Internet of Things. In *Information and Communication Technology. ICT-EurAsia 2013. Lecture Notes in Computer Science*; Mustofa, K., Neuhold, E.J., Tjoa, A.M., Weippl, E.Y.I., Eds.; Springer: Berlin/Heidelberg, Germany, 2013; Volume 7804, pp. 50–59. [CrossRef]
39. Fortino, G.; Savaglio, C.; Spezzano, G.; Zhou, M. Internet of Things as System of Systems: A Review of Methodologies, Frameworks, Platforms, and Tools. *IEEE Trans. Syst. Man Cybern. Syst.* **2021**, *51*, 223–236. [CrossRef]
40. Bouanaka, C.; Benlahrache, N.; Benhamaid, S.; Bouhamed, E. A Review of IoT Systems Engineering: Application to the Smart Traffic Lights System. In Proceedings of the 4th International Conference on Advanced Aspects of Software Engineering, ICAASE 2020, Constantine, Algeria, 28–30 November 2020; Institute of Electrical and Electronics Engineers Inc.: New York, NY, USA, 2020. [CrossRef]
41. Pressman, R.S.; Maxim, B. *Software Engineering: A Practitioner's Approach*, 8th ed.; McGraw-Hill Education: New York, NY, USA, 2015; ISBN 9781259253157.
42. Beck, K.; Beedle, M.; van Bennekum, A.; Cockburn, A.; Cunningham, W.; Fowler, M.; Grenning, J.; Highsmith, J.; Hunt, A.; Jeffries, R.; et al. Manifesto for Agile Software Development. Available online: http://agilemanifesto.org/ (accessed on 1 October 2019).
43. Fowler, M.; Highsmith, J. The Agile Manifesto. *Softw. Dev.* **2001**, *9*, 28–35.
44. Hazzan, O.; Dubinsky, Y. The Agile Manifesto. In *SpringerBriefs in Computer Science*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 9–14. [CrossRef]
45. Hijazi, H.; Khdour, T.; Alarabeyyat, A. A Review of Risk Management in Different Software Development Methodologies. *Int. J. Comput. Appl. Technol.* **2012**, *45*, 8–12. [CrossRef]
46. Jones, T.S.; Richey, R.C. Rapid Prototyping Methodology in Action: A Developmental Study. *Educ. Technol. Res. Dev.* **2000**, *48*, 63–80. [CrossRef]
47. Pierre de Oliveira, R.; Grande, C.; Tiago Massoni, B.; Narallynne Maciel de Araújo, B.; Freitas Sarmento, C.; Silva dos Santos, F.; Massoni, T.; Maciel de Araújo, N. Ants Doing Legwork: Investigating Motivators for Software Development Career Abandonment. In Proceedings of the ACM International Conference Proceeding Series; Association for Computing Machinery: Joinville, Brazil, 2021; pp. 353–362.
48. Matsubara, P.G.F.; Steinmacher, I.; Gadelha, B.; Conte, T.U. Buying Time in Software Development: How Estimates Become Commitments? In Proceedings of the IEEE/ACM 13th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2021, Madrid, Spain, 20–21 May 2021; IEEE: New York, NY, USA, 2021; pp. 61–70. [CrossRef]
49. Ravaglia, C.C.; Mexas, M.P.; Dias, A.C.; da Silveira Batista, H.M.; da Silva Nunes, K. Management of Software Development Projects in Brazil Using Agile Methods. *Indep. J. Manag. Prod.* **2021**, *12*, 1357–1374. [CrossRef]
50. Narang, P.; Mittal, P. Performance Assessment of Traditional Software Development Methodologies and DevOps Automation Culture. *Eng. Technol. Appl. Sci. Res.* **2022**, *12*, 9726–9731. [CrossRef]

51. Beerbaum, D.O. Applying Agile Methodology to Regulatory Compliance Projects in the Financial Industry: A Case Study Research. *J. Appl. Res. Dig. Econ.* **2019**, *2*, 1–11. [CrossRef]
52. Thesing, T.; Feldmann, C.; Burchardt, M. Agile versus Waterfall Project Management: Decision Model for Selecting the Appropriate Approach to a Project. *Procedia Comput. Sci.* **2021**, *181*, 746–756. [CrossRef]
53. Soares, D.; da Silva, F.J.; Ramos, S.C.F.; Kirytopoulos, K.; Sá, J.C.; Ferreira, L.P. Identifying Barriers in the Implementation of Agile Methodologies in Automotive Industry. *Sustainability* **2022**, *14*, 5453. [CrossRef]
54. Younus, A.M.; Younis, H. Conceptual Framework of Agile Project Management, Affecting Project Performance, Key: Requirements and Challenges. *Int. J. Innov. Res. Eng. Manag.* **2021**, *8*, 10–14. [CrossRef]
55. Gea, T.; Paradells, J.; Lamarca, M.; Roldan, D. Smart Cities as an Application of Internet of Things: Experiences and Lessons Learnt in Barcelona. In Proceedings of the 7th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, IMIS 2013, Taichung, Taiwan, 3–5 July 2013; IEEE: Taichung, Taiwan, 2013; pp. 552–557. [CrossRef]
56. Yelamarthi, K.; Aman, M.S.; Abdelgawad, A. An Application-Driven Modular IoT Architecture. *Wirel. Commun. Mob. Comput.* **2017**, *2017*, 1–16. [CrossRef]
57. Nugra, H.; Abad, A.; Fuertes, W.; Galarraga, F.; Aules, H.; Villacis, C.; Toulkeridis, T. A Low-Cost IoT Application for the Urban Traffic of Vehicles, Based on Wireless Sensors Using GSM Technology. In Proceedings of the IEEE International Symposium on Distributed Simulation and Real-Time Applications, DS-RT, London, UK, 21–23 September 2016; IEEE: Uxbridge, UK, 2016; pp. 161–169. [CrossRef]
58. Fuertes, W.; Carrera, D.; Villacis, C.; Toulkeridis, T.; Galarraga, F.; Torres, E.; Aules, H. Distributed System as Internet of Things for a New Low-Cost, Air Pollution Wireless Monitoring on Real Time. In Proceedings of the 2015 IEEE/ACM 19th International Symposium on Distributed Simulation and Real Time Applications, DS-RT 2015, Chengdu, China, 14–16 October 2015; pp. 58–67. [CrossRef]
59. Peterson, B.; Vogel, B. Prototyping the Internet of Things with Web Technologies: Is It Easy? In Proceedings of the 2018 IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2018, Athens, Greece, 19–23 March 2018; IEEE: Athens, Greece, 2018; pp. 518–522. [CrossRef]
60. Guerra Terán, P.; Plua, R.K. Home Automation Application for the Monitoring and Control of an Electric Water Heater Using AWS Technology. In Proceedings of the IEEE 38th Central America and Panama Convention, CONCAPAN 2018, San Salvador, El Salvador, 7–9 November 2018; IEEE: San Salvador, El Salvador, 2018; pp. 1–6. [CrossRef]
61. *ISO/IEC/IEEE 15289:2019*; Systems and Software Engineering-Content of Life-Cycle Information Items (Documentation)-IEEE Standard. International Organization for Standardization: Geneva, Switzerland, 2019; Volume 2019, pp. 1–94. [CrossRef]
62. Laporte, C.Y.; Vargas, E.P. The Development of International Standards to Facilitate Process Improvements for Very Small Entities. 2014, pp. 1335–1361. Available online: https://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/978-1-4666-4301-7.ch065 (accessed on 29 December 2022).
63. Bourque, P.; Fairley, R.E. *Guide to the Software Engineering Body of Knowledge (SWEBOK(R))*, 3rd ed.; IEEE Computer Society Press: Washington, DC, USA, 2014; ISBN 978-0-7695-5166-1.
64. Skordalakis, E. Software Engineering Teaching At NTUA. *WIT Trans. Inf. Commun. Technol.* **1970**, *7*, 472. [CrossRef]
65. Fortino, G.; Savaglio, C.; Palau, C.E.; de Puga, J.S.; Ghanza, M.; Paprzycki, M.; Montesinos, M.; Liotta, A.; Llop, M. Towards Multi-Layer Interoperability of Heterogeneous IoT Platforms: The INTER-IoT Approach. In *Integration, Interconnection, and Interoperability of IoT Systems*; Springer: Cham, Switzerland, 2018; pp. 199–232. [CrossRef]
66. Lawal, A.; Chukwu Ogbu, R. A Comparative Analysis of Agile and Waterfall Software Development Methodologies. *Bakolori J. Gen. Stud.* **2021**, *11*, 1–2.
67. *ISO/IEC/IEEE 15288*; Systems and Software Engineering-System Life Cycle Processes. International Organization for Standardization: Geneva, Switzerland, 2015; Volume 17, pp. 1–108. [CrossRef]
68. *ISO/IEC/IEEE 24765:2017*; ISO/IEC/IEEE International Standard-Systems and Software Engineering–Vocabulary. International Organization for Standardization: Geneva, Switzerland, 2017; pp. 1–541. [CrossRef]
69. Guide to the Systems Engineering Body of Knowledge (SEBoK). Available online: https://www.sebokwiki.org/wiki/Guide_to_the_Systems_Engineering_Body_of_Knowledge_(SEBoK) (accessed on 7 November 2022).
70. Project Management Institute. *Software Extension to the PMBoK Guide*, 5th ed.; Project Management Institute: Newtown Square, PA, USA, 2013; ISBN 9781628250138.
71. Project Management Institute. *A Guide to the Project Management Body of Knowledge (PMBOK®Guide)*, 7th ed.; Project Management Institute: Newtown Square, PA, USA, 2021; ISBN 9781628256659.
72. *12207-2017-ISO/IEC/IEEE*; International Standard—Systems and Software Engineering—Software Life Cycle Processes. IEEE Standards Association: Piscataway, NJ, USA, 2017; pp. 1–148. [CrossRef]
73. Barker, T.T. Documentation for Software and IS Development. In *Encyclopedia of Information Systems*; Academic Press: Cambridge, MA, USA, 2003; pp. 683–693. [CrossRef]
74. Wikipedia. Software Development Process. Available online: https://en.wikipedia.org/wiki/Software_development_process (accessed on 4 October 2022).
75. Badawi, H.F.; Laamarti, F.; el Saddik, A. ISO/IEEE 11073 Personal Health Device (X73-PHD) Standards Compliant Systems: A Systematic Literature Review. *IEEE Access* **2019**, *7*, 3062–3073. [CrossRef]
76. Alberternst, S.; Anisimov, A.; Antakli, A.; Duppe, B.; Hoffmann, H.; Meiser, M.; Muaz, M.; Spieldenner, D.; Zinnikus, I. Orchestrating Heterogeneous Devices and AI Services as Virtual Sensors for Secure Cloud-Based IoT Applications. *Sensors* **2021**, *21*, 7509. [CrossRef]

77. Lakhan, A.; Mohammed, M.A.; Abdulkareem, K.H.; Jaber, M.M.; Nedoma, J.; Martinek, R.; Zmij, P. Delay Optimal Schemes for Internet of Things Applications in Heterogeneous Edge Cloud Computing Networks. *Sensors* **2022**, *22*, 5937. [CrossRef] [PubMed]
78. Patel, P.; Cassou, D. Enabling High-Level Application Development for the Internet of Things. *J. Syst. Softw.* **2015**, *103*, 62–84. [CrossRef]
79. Tabor, L.P. The EDVAC, an Electronic Digital Computer. *Astron. J.* **1948**, *53*, 205. [CrossRef]
80. Guthrie, R. Program Design, Coding, and Testing. *Encycl. Inf. Syst.* **2003**, 529–543. [CrossRef]
81. Yourdon, E. *Modern Structured Analysis*, 1st ed.; Yourdon Press: Ann Arbor, MI, USA, 1989; ISBN 9780135986240.
82. Arpita, G.; Netra, P. *Magnifying Object-Oriented Analysis and Design*; PHI Learning: New Delhi, India, 2014; ISBN 9788120340688.
83. Sharma, S.; Hasteer, N.; Mishra, S.P.; van Belle, J.P. Identifying the Contextual Relationship among the Agile Adoption Factors through Interpretive Structural Modeling. In Proceedings of the International Conference on Information Technology, InCITe 2016, Noida, India, 6–7 October 2016; Institute of Electrical and Electronics Engineers Inc.: Noida, India, 2017; pp. 87–92. [CrossRef]
84. Medina Otalvaro, C.M.; Blandon Andrade, J.C.; Zapata Jaramillo, C.M.; RiosPatino, J.I. IoT Best Practices and Their Components: A Systematic Literature Review. *IEEE Latin Am. Trans.* **2022**, *20*, 2217–2228. [CrossRef]
85. Erazo, O.; Guerrero-Ulloa, G.; Guzmán, D.; Cáceres, C. From a Common Chair to a Device That Issues Reminders to Seniors. *Commun. Comput. Inf. Sci.* **2020**, *1194*, 439–448. [CrossRef]
86. Gogineni, S.K.; Riedelsheimer, T.; Stark, R. Systematic Product Development Methodology for Customizable IoT Devices. In Proceedings of the Procedia CIRP, Póvoa de Varzim, Portgal, 8–10 May 2019; Elsevier B.V.: Amsterdam, The Netherlands, 2019; Volume 84, pp. 393–399. [CrossRef]
87. *24748-1-2018-ISO/IEC/IEEE*; International Standard—Systems and Software Engineering—Life Cycle Management—Part 1: Guidelines for Life Cycle Management. IEEE Standards Association: Piscataway, NJ, USA, 2018; pp. 1–75. [CrossRef]
88. *24748-3-2020-ISO/IEC/IEEE*; International Standard—Systems and Software Engineering—Life Cycle Management—Part 3: Guidelines for the Application of ISO/IEC/IEEE 12207 (Software Life Cycle Processes). IEEE Standards Association: Piscataway, NJ, USA, 2020; pp. 1–69. [CrossRef]
89. *24748-4-2016-ISO/IEC/IEEE*; International Standard for Systems and Software Engineering—Life Cycle Management—Part 4: Systems Engineering Planning. IEEE Standards Association: Piscataway, NJ, USA, 2016; pp. 1–75. [CrossRef]
90. Preston, M. 7 Phases of the System Development Life Cycle Guide. Available online: https://www.clouddefense.ai/blog/system-development-life-cycle (accessed on 4 October 2022).
91. Stetsuyk, E.; Maevsky, D.; Maevskaya, E. Methodology of Green Software Development for the IoT Devices. *Int. J. Inf. Technol. Secur.* **2018**, *10*, 3–12.
92. Zelfia, H.; Simanungkalit, T.; Raharjo, T. Comparison of Scrum Maturity between Internal and External Software Development: A Case Study at One of the State-Owned Banks in Indonesia. In Proceedings of the 1st International Conference on Information System and Information Technology, ICISIT 2022, Yogyakarta, Indonesia, 26–27 July 2022; Institute of Electrical and Electronics Engineers Inc.: New York, NY, USA, 2022; pp. 312–317. [CrossRef]
93. Teslyuk, V.; Batyuk, A.; Voityshyn, V. Method of Software Development Project Duration Estimation for Scrum Teams with Differentiated Specializations. *Systems* **2022**, *10*, 123. [CrossRef]
94. Pico-Valencia, P.; Holgado-Terriza, J.A.; Paderewski, P. A Systematic Method for Building Internet of Agents Applications Based on the Linked Open Data Approach. *Future Gener. Comput. Syst.* **2019**, *94*, 250–271. [CrossRef]
95. Rising, L.; Janoff, N.S. Scrum Software Development Process for Small Teams. *IEEE Softw.* **2000**, *17*, 26–32. [CrossRef]
96. Muntés-Mulero, V.; Ripolles, O.; Gupta, S.; Dominiak, J.; Willeke, E.; Matthews, P.; Somosköi, B. Agile Risk Management for Multi-Cloud Software Development. *IET Softw.* **2019**, *13*, 172–181. [CrossRef]
97. Renanti, M.D.; Darmawan, A.C. Application of The Multiple Intelligent Level Determination for Interest and Talent Development. *E3S Web Conf.* **2022**, *348*, 00016. [CrossRef]
98. Pecchia, C.; Trincardi, M.; di Bello, P. Expressing, Managing, and Validating User Stories: Experiences from the Market. *Commun. Comput. Inf. Sci.* **2016**, *422*, 103–111. [CrossRef]
99. *26515-2018-ISO/IEC/IEEE*; International Standard—Systems and Software Engineering—Developing Information for Users in an Agile Environment. IEEE Standards Association: Piscataway, NJ, USA, 2018. [CrossRef]
100. Scrum.org. *Scrum Master Trends 2019*; Scrum.org.: Burlington, VT, USA; Berlin, Germany, 2019.
101. Dalpiaz, F.; van der Schalk, I.; Brinkkemper, S.; Aydemir, F.B.; Lucassen, G. Detecting Terminological Ambiguity in User Stories: Tool and Experimentation. *Inf. Softw. Technol.* **2019**, *110*, 3–16. [CrossRef]
102. Usländer, T.; Batz, T. Agile Service Engineering in the Industrial Internet of Things. *Future Internet* **2018**, *10*, 100. [CrossRef]
103. Zheng, M.; Xu, D.; Jiang, L.; Gu, C.; Tan, R.; Cheng, P. Challenges of Privacy-Preserving Machine Learning in IoT. In Proceedings of the International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things, AIChallengeIoT 2019, New York, NY, USA, 10 November 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 1–7. [CrossRef]
104. Weber, M.; Boban, M. Security Challenges of the Internet of Things. In Proceedings of the 39th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2016, Opatija, Croatia, 30 May–3 June 2016; pp. 638–643. [CrossRef]
105. Tseng, K.-H.; Chung, M.-Y.; Chen, L.-H.; Wei, M.-Y. Applying an Integrated System of Cloud Management and Wireless Sensing Network to Green Smart Environments-Green Energy Monitoring on Campus. *Sensors* **2022**, *22*, 6521. [CrossRef]

106. Maddikunta, P.K.R.; Gadekallu, T.R.; Kaluri, R.; Srivastava, G.; Parizi, R.M.; Khan, M.S. Green Communication in IoT Networks Using a Hybrid Optimization Algorithm. *Comput. Commun.* **2020**, *159*, 97–107. [CrossRef]
107. Ren, H.; Li, H.; Dai, Y.; Yang, K.; Lin, X. Querying in Internet of Things with Privacy Preserving: Challenges, Solutions and Opportunities. *IEEE Netw.* **2018**, *32*, 144–151. [CrossRef]
108. Ammar, M.; Russello, G.; Crispo, B. Internet of Things: A Survey on the Security of IoT Frameworks. *J. Inf. Secur. Appl.* **2018**, *38*, 8–27. [CrossRef]
109. Pirbhulal, S.; Zhang, H.; Mukhopadhyay, S.C.; Li, C.; Wang, Y.; Li, G.; Wu, W.; Zhang, Y.T. An Efficient Biometric-Based Algorithm Using Heart Rate Variability for Securing Body Sensor Networks. *Sensors* **2015**, *15*, 15067–15089. [CrossRef]
110. Pirbhulal, S.; Zhang, H.; Alahi, M.E.E.; Ghayvat, H.; Mukhopadhyay, S.C.; Zhang, Y.T.; Wu, W. A Novel Secure IoT-Based Smart Home Automation System Using a Wireless Sensor Network. *Sensors* **2016**, *17*, 69. [CrossRef] [PubMed]
111. Babaie, M.; Kuo, F.W.; Chen, H.N.R.; Cho, L.C.; Jou, C.P.; Hsueh, F.L.; Shahmohammadi, M.; Staszewski, R.B. A Fully Integrated Bluetooth Low-Energy Transmitter in 28 Nm CMOS With 36% System Efficiency at 3 DBm. *IEEE J. Solid-State Circuits* **2016**, *51*, 1547–1565. [CrossRef]
112. Pullini, A.; Rossi, D.; Loi, I.; Tagliavini, G.; Benini, L. Mr.Wolf: An Energy-Precision Scalable Parallel Ultra Low Power SoC for IoT Edge Processing. *IEEE J. Solid-State Circuits* **2019**, *54*, 1970–1981. [CrossRef]
113. Koteshwara, S.; Parhi, K.K. Incremental-Precision Based Feature Computation and Multi-Level Classification for Low-Energy Internet-of-Things. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2018**, *8*, 822–835. [CrossRef]
114. Lang, Y.; Wang, Q.; Yang, Y.; Hou, C.; Liu, H.; He, Y. Joint Motion Classification and Person Identification via Multitask Learning for Smart Homes. *IEEE Internet Things J.* **2019**, *6*, 9596–9605. [CrossRef]
115. Yu, Z.; Du, H.; Xiao, D.; Wang, Z.; Han, Q.; Guo, B. Recognition of Human Computer Operations Based on Keystroke Sensing by Smartphone Microphone. *IEEE Internet Things J.* **2018**, *5*, 1156–1168. [CrossRef]
116. Sachdeva, V.; Chung, L. Handling Non-Functional Requirements for Big Data and IOT Projects in SCRUM. In Proceedings of the 7th International Conference Confluence 2017 on Cloud Computing, Data Science and Engineering, Noida, India, 12–13 January 2017; IEEE: Noida, India, 2017; pp. 216–221. [CrossRef]
117. Keshta, N.; Morgan, Y. Comparison between Traditional Plan-Based and Agile Software Processes According to Team Size & Project Domain (A Systematic Literature Review). In Proceedings of the 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference, IEMCON 2017, Vancouver, BC, Canada, 3–5 October 2017; Institute of Electrical and Electronics Engineers Inc.: Vancouver, BC, Canada, 2017; pp. 567–575. [CrossRef]
118. Alqudah, M.; Razali, R. A Comparison of Scrum and Kanban for Identifying Their Selection Factors. In Proceedings of the 6th International Conference on Electrical Engineering and Informatics: Sustainable Society through Digital Innovation, ICEEI 2017, Langkawi Island, Malaysia, 25–27 November 2017; Institute of Electrical and Electronics Engineers Inc.: Vancouver, BC, Canada, 2018; Volume 2017, pp. 1–6. [CrossRef]
119. Kettunen, P.; Laanti, M. Future Software Organizations–Agile Goals and Roles. *Eur. J. Futures Res.* **2017**, *5*, 1–15. [CrossRef]
120. Morais dos Santos, M.V.; Barbosa da Silva, P.D.; Lamas Otero, A.G.; Wisnieski, R.T.; Sousa Goncalves, G.; Esteves Maria, R.; Vieira Dias, L.A.; Marques da Cunha, A. Applying Scrum in an Interdisciplinary Project for Fraud Detection in Credit Card Transactions. *Adv. Intell. Syst. Comput.* **2016**, *448*, 461–471. [CrossRef]
121. Dai, H.N.; Zheng, Z.; Zhang, Y. Blockchain for Internet of Things: A Survey. *IEEE Internet Things J.* **2019**, *6*, 8076–8094. [CrossRef]
122. Hou, J.; Li, Y.; Yu, J.; Shi, W. A Survey on Digital Forensics in Internet of Things. *IEEE Internet Things J.* **2020**, *7*, 1–15. [CrossRef]
123. Sosa-Reyna, C.M.; Tello-Leal, E.; Lara-Alabazares, D.; Mata-Torres, J.A.; Lopez-Garza, E. A Methodology Based on Model-Driven Engineering for IoT Application Development. In *ICDS 2018*; Berntzen, L., Hartog, M., Eds.; IARIA: Rome, Italy, 2018; pp. 36–41. ISBN 978-1-61208-615-6.
124. Sosa-Reyna, C.M.; Tello-Leal, E.; Lara-Alabazares, D. An Approach Based on Model-Driven Development for IoT Applications. In Proceedings of the IEEE International Congress on Internet of Things, ICIOT 2018, San Francisco, CA, USA, 2–7 July 2018; IEEE: New York, NY, USA, 2018; pp. 134–139. [CrossRef]
125. Sosa-Reyna, C.M.; Tello-Leal, E.; Lara-Alabazares, D. Methodology for the Model-Driven Development of Service Oriented IoT Applications. *J. Syst. Archit.* **2018**, *90*, 15–22. [CrossRef]
126. OpenMBEE. Open Model Based Engineering Environment. Available online: https://www.openmbee.org/index.html (accessed on 21 October 2022).
127. Cabot, J. Clarifying Concepts: MBE vs MDE vs MDD vs MDA. Available online: https://modeling-languages.com/clarifying-concepts-mbe-vs-mde-vs-mdd-vs-mda/ (accessed on 20 August 2022).
128. Object Management Group. MDA Specifications. Available online: http://www.omg.org/mda/specs.htm (accessed on 20 August 2022).
129. Ameller, D. Considering Non-Functional Requirements in Model-Driven Engineering. Master's Thesis, Universitat Politècnica de Catalunya, Barcelona, Spain, 2009.
130. Object Management Group. Model Driven Architecture (MDA). Available online: https://www.omg.org/mda/ (accessed on 17 August 2022).
131. Belaunde, M.; Burt, C.; Casanave, C.; Cummins, F.; DSouza, D.; Duddy, K.; el Kaim, W.; Kenne-dy, A.; Frank, W.; Frankel, D.; et al. Model Driven Architecture (MDA) 2003; 62p. Available online: http://www.omg.org/cgi-bin/doc?omg/03-06-01 (accessed on 17 August 2022).

132. Schmidt, D.C. Model-Driven Engineering. *Computer* **2006**, *39*, 25–31. [CrossRef]
133. Ashton, K. That "Internet of Things" Thing. *RFID J.* **2009**, *22*, 97–114.
134. Praveen, S.V.; Ittamalla, R.; Deepak, G. Analyzing Indian General Public's Perspective on Anxiety, Stress and Trauma during COVID-19—A Machine Learning Study of 840,000 Tweets. *Diabetes Metab. Syndr. Clin. Res. Rev.* **2021**, *15*, 667–671. [CrossRef] [PubMed]
135. Goel, R.; Sharma, R. Studying Leaders & Their Concerns Using Online Social Media during the Times of Crisis—A COVID Case Study. *Soc. Netw. Anal. Min.* **2021**, *11*, 46. [CrossRef]
136. Fong, S.L.; Wui Yung, D.C.; Ahmed, F.Y.H.; Jamal, A. Smart City Bus Application with Quick Response (QR) Code Payment. In Proceedings of the 2019 8th International Conference on Software and Computer Applications 2019, Penang, Malaysia, 19–21 February 2019; Volume F1479, pp. 248–252. [CrossRef]
137. Paasivaara, M.; Vanhanen, J.; Lassenius, C. Collaborating with Industrial Customers in a Capstone Project Course: The Customers' Perspective. In Proceedings of the 41st International Conference on Software Engineering: Software Engineering Education and Training, ICSE-SEET 2019, Montreal, QC, Canada, 25–31 May 2019; IEEE: Montreal, QC, Canada, 2019; pp. 12–22. [CrossRef]
138. Enciso, L.; Sarango, J.; Valladarez, A.; Condolo, J. A Mobile Application for a Smart Car. In Proceedings of the Iberian Conference on Information Systems and Technologies, CISTI 2019, Coimbra, Portugal, 19–22 June 2019; IEEE: Coimbra, Portugal, 2019; Volume 2019, pp. 1–7. [CrossRef]
139. Cahill, J.; Portales, R.; McLoughin, S.; Nagan, N.; Henrichs, B.; Wetherall, S. IoT/Sensor-Based Infrastructures Promoting a Sense of Home, Independent Living, Comfort and Wellness. *Sensors* **2019**, *19*, 485. [CrossRef]
140. Rodriguez-Ruiz, J.G.; Galvan-Tejada, C.E.; Vazquez-Reyes, S.; Galvan-Tejada, J.I.; Gutiérrez-Gnecchi, J.A. Cardiopulmonary Simulator Using an Internet of Things Approach. In Proceedings of the 6th International Conference in Software Engineering Research and Innovation, CONISOFT 2018, San Luis Potosi, Mexico, 24–26 October 2018; IEEE: San Luis Potosí, Mexico, 2019; pp. 123–131. [CrossRef]
141. da Silva, D.A.; de Barros Santana, R.M.; Navas, J.; Goncalves, G.S.; Vieira Dias, L.A.; da Cunha, A.M.; Tasinaffo, P.M. Health Care Transformation: An Academic Application System Case Study. *IFAC-PapersOnLine* **2018**, *51*, 413–418. [CrossRef]
142. Ibba, S.; Pinna, A.; Seu, M.; Pani, F.E. CitySense: Blockchain-Oriented Smart Cities. In Proceedings of the XP2017 Scientific Workshops 2017, Cologne, Germany, 22–26 May 2017; Volume F1299, pp. 1–5. [CrossRef]
143. Fahrianto, F.; Anggraini, N.; Suseno, H.B.; Shabrina, A.; Reza, A. Smart Data Centre Monitoring System Based on Internet of Things (IoT) (Study Case: Pustipanda UIN Jakarta). In Proceedings of the 5th International Conference on Cyber and IT Service Management, CITSM 2017, Denpasar, Indonesia, 8–10 August 2017; IEEE: Denpasar, Indonesia, 2017; pp. 1–9.
144. Müller, R.; Vette, M.; Hörauf, L.; Speicher, C.; Burkhard, D. Lean Information and Communication Tool to Connect Shop and Top Floor in Small and Medium-Sized Enterprises. *Procedia Manuf.* **2017**, *11*, 1043–1052. [CrossRef]
145. Rizqyawan, M.I.; Amri, M.F.; Pratama, R.P.; Turnip, A. Design and Development of Android-Based Cloud ECG Monitoring System. In Proceedings of the 3rd International Conference on Information Technology, Computer, and Electrical Engineering, ICITACEE 2016, Semarang, Indonesia, 19–20 October 2016; IEEE: Semarang, Indonesia, 2017; pp. 1–5.
146. Lima, G.L.B.; Ferreira, G.A.L.; Saotome, O.; da Cunha, A.M.; Dias, L.A.V. Hardware Development: Agile and Co-Design. In Proceedings of the 12th International Conference on Information Technology: New Generations, ITNG 2015, Las Vegas, NV, USA, 13–15 April 2015; IEEE: Las Vegas, NV, USA, 2015; pp. 784–787. [CrossRef]
147. Esteves María, R.; Rodrigues Junior, L.A.; Guarino De Vasconcelos, L.E.; Mancilha Pinto, A.F.; Tsoucamoto, P.T.; Angelim Silva, H.N.; Lastori, A.; Marques Cunha, D.A.; Vieira Dias, L.A. Applying Scrum in an Interdisciplinary Project Using Big Data, Internet of Things, and Credit Cards. In Proceedings of the 12th International Conference on Information Technology: New Generations, ITNG 2015, Las Vegas, NV, USA, 13–15 April 2015; IEEE: Las Vegas, NV, USA, 2015; pp. 67–72. [CrossRef]
148. Dafoulas, G.; Samuels-Clarke, J.; Maia, C.C.; Ali, A.A.; Tsiakara, A. Offering Smarter Learning Support through the Use of Biometrics. In Proceedings of the 26th International Conference on Telecommunications, ICT 2019, Hanoi, Vietnam, 8–10 April 2019; IEEE: Hanoi, Vietnam, 2019; pp. 270–274. [CrossRef]
149. Guan, G.; Dong, W.; Gao, Y.; Bu, J. Towards Rapid and Cost-Effective Prototyping of IoT Platforms. In Proceedings of the International Conference on Network Protocols, ICNP, Singapore, 8–11 November 2016; IEEE: Singapore, 2016; Volume 2016, pp. 1–5. [CrossRef]
150. Musyoka, F.M.; Thiga, M.M.; Muketha, G.M. A 24-Hour Ambulatory Blood Pressure Monitoring System for Preeclampsia Management in Antenatal Care. *Inform. Med. Unlocked* **2019**, *16*, 100199. [CrossRef]
151. Sharma, S.; Das, S.; Virmani, J.; Sharma, M.; Singh, S.; Das, A. IoT Based Dipstick Type Engine Oil Level and Impurities Monitoring System: A Portable Online Spectrophotometer. In Proceedings of the 2019 4th International Conference on Internet of Things: Smart Innovation and Usages, IoT-SIU 2019, Ghaziabad, India, 18–19 April 2019; IEEE: Ghaziabad, India, 2019; pp. 1–4. [CrossRef]
152. Gray, S.; Clark, F.; Burgess, K.; Metcalfe, T.; Kadijevic, A.; Cater, K.; Bennett, P. Gorilla Game Lab: Exploring Modularity, Tangibility and Playful Engagement in Cognitive Enrichment Design. In Proceedings of the Fifth International Conference on Animal-Computer Interaction 2018, Atlanta, GA, USA, 4–6 December 2018; ACM Press: Atlanta, GA, USA, 2018; pp. 1–13. [CrossRef]
153. Martillano, D.A.; Chowdhury, A.F.D.; Dellosa, J.C.M.; Murcia, A.A.; Mangoma, R.J.P. Pindots: An Assistive Six-Dot Braille Cell Keying Device on Basic Notation Writing for Visually Impaired Students with IoT Technology. In Proceedings of the 2018 2nd International Conference on Education and E-Learning 2018, Bali, Indonesia, 5–7 November 2018; Association for Computing Machinery: New York, NY, USA, 2018; pp. 41–47.

154. Pawar, N.; Bourgeau, T.; Chaouchi, H. PrIoT: Prototyping the Internet of Things. In Proceedings of the 2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud), Barcelona, Spain, 6–8 August 2018; IEEE: Barcelona, Spain, 2018; pp. 216–223. [CrossRef]
155. Karvinen, K.; Karvinen, T. IoT Rapid Prototyping Laboratory Setup. *Int. J. Eng. Educ.* **2018**, *34*, 263–272.
156. Moon, S.; Min, M.; Nam, J.; Park, J.; Lee, D.; Kim, D. Drowsy Driving Warning System Based on GS1 Standards with Machine Learning. In Proceedings of the 2017 IEEE 6th International Congress on Big Data, BigData Congress 2017, Honolulu, HI, USA, 25–30 June 2017; IEEE: Honolulu, HI, USA, 2017; pp. 289–296. [CrossRef]
157. Scheible, J.; Engeln, A.; Burmester, M.; Zimmermann, G.; Keber, T.; Schulz, U.; Palm, S.; Funk, M.; Schaumann, U. SMARTK-ITCHEN Media Enhanced Cooking Environment. In Proceedings of the 6th International Conference on the Internet of Things 2016, Granada, Spain, 22–25 October 2019; Association for Computing Machinery: New York, NY, USA, 2016; pp. 169–170. [CrossRef]
158. Kruger, C.P.; Abu-Mahfouz, A.M.; Hancke, G.P. Rapid Prototyping of a Wireless Sensor Network Gateway for the Internet of Things Using Off-the-Shelf Components. In Proceedings of the IEEE International Conference on Industrial Technology, Seville, Spain, 17–19 March 2015; IEEE: Seville, Spain, 2015; Volume 2015, pp. 1926–1931. [CrossRef]
159. Al-Taee, M.A.; Sungoor, A.H.; Abood, S.N.; Philip, N.Y. Web-of-Things Inspired e-Health Platform for Integrated Diabetes Care Management. In Proceedings of the 2013 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies, AEECT 2013, Amman, Jordan, 3–5 December 2013; IEEE: Amman, Jordan, 2013; pp. 1–6. [CrossRef]
160. Al-Taee, M.A.; Al-Nuaimy, W.; Al-Ataby, A.; Muhsin, Z.J.; Abood, S.N. Mobile Health Platform for Diabetes Management Based on the Internet-of-Things. In Proceedings of the Jordan Conference on Applied Electrical Engineering and Computing Technologies, AEECT 2015, Amman, Jordan, 3–5 November 2015; IEEE: Amman, Jordan, 2015; pp. 1–5. [CrossRef]
161. Kim, H.J. Rapid Smart Environment Prototyping for Early Conceptual Design. In Proceedings of the Designing Interactive Systems Conference, DIS 2018, Hong Kong, China, 9–13 June 2018; Association for Computing Machinery: New York, NY, USA, 2018; pp. 363–366. [CrossRef]
162. Mora, S.; Gianni, F.; Divitini, M. RapIoT Toolkit: Rapid Prototyping of Collaborative Internet of Things Applications. In Proceedings of the International Conference on Collaboration Technologies and Systems, CTS 2016, Orlando, FL, USA, 31 October–4 November 2016; IEEE: Orlando, FL, USA, 2017; pp. 438–445. [CrossRef]
163. Gianni, F.; Mora, S.; Divitini, M. RapIoT Toolkit: Rapid Prototyping of Collaborative Internet of Things Applications. *Future Gener. Comput. Syst.* **2019**, *95*, 867–879. [CrossRef]
164. Pereira, A.; Patrício, B.; Fonte, F.; Marques, S.; Reis, C.I.; Maximiano, M. Collecting Information About Air Quality Using Smartphones. *Procedia Comput. Sci.* **2018**, *138*, 33–40. [CrossRef]
165. Escobar, L.; Carvajal, N.; Naranjo, J.; Ibarra, A.; Villacis, C.; Zambrano, M.; Galarraga, F. Design and Implementation of Complex Systems Using Mechatronics and Cyber-Physical Systems Approaches. In Proceedings of the IEEE International Conference on Mechatronics and Automation, ICMA 2017, Takamatsu, Japan, 6–9 August 2017; IEEE: Takamatsu, Japan, 2017; pp. 147–154. [CrossRef]
166. Yang, L.Q.; Bi, Y.Y. Internet of Things Technology Implementation by Applying SDLC Model: The Intelligent Storage Management System. *Appl. Mech. Mater.* **2014**, *556–562*, 5385–5390. [CrossRef]
167. ITU. Statistics—Individuals Using the Internet. Available online: www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx (accessed on 6 October 2019).
168. Lekidis, A.; Stachtiari, E.; Katsaros, P.; Bozga, M.; Georgiadis, C.K. Model-Based Design of IoT Systems with the BIP Component Framework. *Softw. Pract. Exp.* **2018**, *48*, 1167–1194. [CrossRef]
169. Harbouche, A.; Djedi, N.; Erradi, M.; Ben-Othman, J.; Kobbane, A. Model Driven Flexible Design of a Wireless Body Sensor Network for Health Monitoring. *Comput. Netw.* **2017**, *129*, 548–571. [CrossRef]
170. Brambilla, M.; Umuhoza, E.; Acerbis, R. Model-Driven Development of User Interfaces for IoT Systems Via Domain-Specific Components and Patterns. *J. Internet Serv. Appl.* **2017**, *8*, 14. [CrossRef]
171. Fortino, G.; Russo, W. ELDAMeth: An Agent-Oriented Methodology for Simulation-Based Prototyping of Distributed Agent Systems. *Inf. Softw. Technol.* **2012**, *54*, 608–624. [CrossRef]
172. Ciccozzi, F.; Spalazzese, R. MDE4IoT: Supporting the Internet of Things with Model-Driven Engineering. *Stud. Comput. Intell.* **2017**, *678*, 67–76. [CrossRef]
173. Gomes, L.; Moutinho, F.; Pereira, F. IOPT-Tools—A Web Based Tool Framework for Embedded Systems Controller Development Using Petri Nets. In Proceedings of the 23rd International Conference on Field Programmable Logic and Applications, FPL 2013, Porto, Portugal, 2–4 September 2013; IEEE: Porto, Portugal, 2013; p. 1. [CrossRef]
174. Ataíde, A.; Barros, J.P.; Brito, I.S.; Gomes, L. Towards Automatic Code Generation for Distributed Cyber-Physical Systems: A First Prototype for Arduino Boards. In Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation, ETFA, Limassol, Cyprus, 13–15 September 2017; IEEE: Limassol, Cyprus, 2017; pp. 1–4. [CrossRef]
175. Chauhan, S.; Patel, P.; Delicato, F.C.; Chaudhary, S. A Development Framework for Programming Cyber-Physical Systems. In Proceedings of the 2nd International Workshop on Software Engineering for Smart Cyber-Physical Systems, SEsCPS 2016, Austin, TX, USA, 16 May 2016; Association for Computing Machinery, Inc.: New York, NY, USA, 2016; pp. 47–53. [CrossRef]
176. Corredor, I.; Bernardos, A.M.; Iglesias, J.; Casar, J.R. Model-Driven Methodology for Rapid Deployment of Smart Spaces Based on Resource-Oriented Architectures. *Sensors* **2012**, *12*, 9286–9335. [CrossRef]

177. Fortino, G.; Garro, A.; Mascillaro, S.; Russo, W. Using Event-Driven Lightweight DSC-Based Agents for MAS Modelling. *Int. J. Agent-Oriented Softw. Eng.* **2010**, *4*, 113–140. [CrossRef]
178. Reichlmayr, T. Working towards the Student Scrum—Developing Agile Android Applications. In Proceedings of the 2011 ASEE Annual Conference & Exposition 2011, Vancouver, BC, Canada, 26–29 June 2011; pp. 22.1712.1–22.1712.12. [CrossRef]
179. Maylawati, D.S.; Ramdhani, M.A. Logical Framework of Information Technology: Systematization of Software Development Research. *Telfor J.* **2022**, *14*, 26–32. [CrossRef]
180. Wang, Z.; Cui, L.; Guo, W.; Zhao, L.; Yuan, X.; Gu, X.; Tang, W.; Bu, L.; Huang, W. A Design Method for an Intelligent Manufacturing and Service System for Rehabilitation Assistive Devices and Special Groups. *Adv. Eng. Inform.* **2022**, *51*, 101504. [CrossRef]
181. Schauer, P.; Falas, Ł. Adaptation-Enabled Architecture for Internet of Things Systems. *Lect. Notes Netw. Syst.* **2021**, *182*, 195–204. [CrossRef]
182. Cicirelli, F.; Fortino, G.; Guerrieri, A.; Spezzano, G.; Vinci, A. Metamodeling of Smart Environments: From Design to Implementation. *Adv. Eng. Inform.* **2017**, *33*, 274–284. [CrossRef]
183. Varga, P.; Blomstedt, F.; Ferreira, L.L.; Eliasson, J.; Johansson, M.; Delsing, J.; Martínez de Soria, I. Making System of Systems Interoperable – The Core Components of the Arrowhead Framework. *J. Netw. Comput. Appl.* **2017**, *81*, 85–95. [CrossRef]
184. Costa, B.; Pires, P.F.; Delicato, F.C. Modeling IoT Applications with SysML4IoT. In Proceedings of the 42nd Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2016, Limassol, Cyprus, 31 August–2 September 2016; Institute of Electrical and Electronics Engineers Inc.: New York, NY, USA, 2016; pp. 157–164. [CrossRef]
185. Fortino, G.; Guerrieri, A.; Russo, W.; Savaglio, C. Towards a Development Methodology for Smart Object-Oriented IoT Systems: A Metamodel Approach. In Proceedings of the 2015 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2015, Hong Kong, 9–12 October 2015; pp. 1297–1302. [CrossRef]
186. Fortino, G.; Guerrieri, A.; Russo, W.; Savaglio, C. Integration of Agent-Based and Cloud Computing for the Smart Objects-Oriented IoT. In Proceedings of the 2014 IEEE 18th International Conference on Computer Supported Cooperative Work in Design (CSCWD), Hsinchu, Taiwan, 21–23 May 2014; pp. 493–498. [CrossRef]
187. Bellifemine, F.; Poggi, A.; Rimassa, G. Developing Multi-Agent Systems with a FIPA-Compliant Agent Framework. *Softw. Pract. Exp.* **2001**, *31*, 103–128. [CrossRef]
188. Fortino, G. Agents Meet the IoT: Toward Ecosystems of Networked Smart Objects. *IEEE Syst. Man Cybern. Mag.* **2016**, *2*, 43–47. [CrossRef]
189. Li, B.; Dong, W. Automatic Generation of Iot Device Platforms with Autolink. *IEEE Internet Things J.* **2021**, *8*, 5893–5903. [CrossRef]
190. Dong, W.; Li, B.; Guan, G.; Cheng, Z.; Zhang, J.; Gao, Y. TinyLink: A Holistic System for Rapid Development of IoT Applications. *ACM Trans. Sens. Netw.* **2020**, *17*, 2020. [CrossRef]
191. Cai, H.; Gu, Y.; Vasilakos, A.V.; Xu, B.; Zhou, J. Model-Driven Development Patterns for Mobile Services in Cloud of Things. *IEEE Trans. Cloud Comput.* **2018**, *6*, 771–784. [CrossRef]
192. de Farias, C.M.; Brito, I.C.; Pirmez, L.; Delicato, F.C.; Pires, P.F.; Rodrigues, T.C.; dos Santos, I.L.; Carmo, L.F.R.C.; Batista, T. COMFIT: A Development Environment for the Internet of Things. *Future Gener. Comput. Syst.* **2017**, *75*, 128–144. [CrossRef]
193. Ramesh, R.; Lin, R.; Iannopollo, A.; Sangiovanni-Vincentelli, A.; Hartmann, B.; Dutta, P. Turning Coders into Makers: The Promise of Embedded Design Generation. In Proceedings of the 1st Annual ACM Symposium on Computational Fabrication 2017, Cambridge, MA, USA, 12–13 June 2017. [CrossRef]
194. Botta, A.; de Donato, W.; Persico, V.; Pescapé, A. Integration of Cloud Computing and Internet of Things: A Survey. *Future Gener. Comput. Syst.* **2016**, *56*, 684–700. [CrossRef]
195. Kefalakis, N.; Soldatos, J.; Anagnostopoulos, A.; Dimitropoulos, P. A Visual Paradigm for IoT Solutions Development. In *Lecture Notes in Computer Science*; Podnar Žarko, I., Pripužić, K., Serrano, M., Eds.; Springer: Cham, Switzerland, 2015; Volume 9001, pp. 26–45. [CrossRef]
196. Fazio, M.; Celesti, A.; Marquez, F.G.; Glikson, A.; Villari, M. Exploiting the FIWARE Cloud Platform to Develop a Remote Patient Monitoring System. In Proceedings of the 2015 IEEE Symposium on Computers and Communication (ISCC) 2015, Larnaca, Cyprus, 6–9 July 2015; Volume 2016, pp. 264–270. [CrossRef]
197. Vashi, S.; Ram, J.; Modi, J.; Verma, S.; Prakash, C. Internet of Things (IoT): A Vision, Architectural Elements, and Security Issues. In Proceedings of the International Conference on IoT in Social, Mobile, Analytics and Cloud, I-SMAC 2017, Nadu, India, 10–11 February 2017; IEEE: Palladam, India, 2017; pp. 492–496. [CrossRef]
198. Qiang, M.; Yu-feng, D.; Ting, X.; Shun-li, W. Research of Visualization Monitoring Technology Based on Internet of Things in Discrete Manufacturing Process. In Proceedings of the 2nd International Symposium on Dependable Computing and Internet of Things (DCIT), Wuhan, China, 16–18 November 2015; IEEE: New York, NY, USA, 2015; pp. 128–133. [CrossRef]
199. Industry IoT Consortium. The Industrial Internet Reference Architecture. Available online: https://www.iiconsortium.org/IIRA/ (accessed on 23 October 2022).
200. Lin, S.W.; Durand, B.; Bleakley, G.; Chigani, A.; Martin, R.; Murphy, B.; Crawford, M. *The Industrial Internet of Things Volume G1: Reference Architecture, Version 1.9*; IIC Technical White Paper; Lin, S.-W., Simmon, E., Eds.; Industrial Internet Consortium: Boston, MA, USA, 2019.

201. Faugère, M.; Bourbeau, T.; de Simone, R.; Gérard, S. MARTE: Also an UML Profile for Modeling AADL Applications. In Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems, ICECCS 2007, Auckland, New Zealand, 11–14 July 2007; IEEE: Auckland, New Zealand, 2007; pp. 359–364. [CrossRef]
202. Carnevali, L.; Ridi, L.; Vicario, E. Putting Preemptive Time Petri Nets to Work in a V-Model SW Life Cycle. *IEEE Trans. Softw. Eng.* **2011**, *37*, 826–844. [CrossRef]
203. Nastic, S.; Truong, H.-L.H.-L.L.; Dustdar, S. SDG-Pro: A Programming Framework for Software-Defined IoT Cloud Gateways. *J. Internet Serv. Appl.* **2015**, *6*, 21. [CrossRef]
204. Alvear-Puertas, V.E.; Burbano-Prado, Y.A.; Rosero-Montalvo, P.D.; Tözün, P.; Marcillo, F.; Hernandez, W. Smart and Portable Air-Quality Monitoring IoT Low-Cost Devices in Ibarra City, Ecuador. *Sensors* **2022**, *22*, 7015. [CrossRef]