TDDM4IoTS: A Test-Driven Development Methodology for Internet of Things (IoT)-Based Systems

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

[1] Facultad de Ciencias de la Ingeniería, Universidad Técnica Estatal de Quevedo, Quevedo 120501, Ecuador
gguerrero@uteq.edu.ec

[2] Software Engineering Department, University of Granada, 18071 Granada, Spain
gleiston@correo.ugr.es, {mhornos,carlosrodriguez}@ugr.es

Abstract. This paper presents a development methodology for Internet of Things (IoT)-based Systems (IoTS) that gathers ideas from several of the most outstanding software development paradigms nowadays, such as Model-Driven Engineering (MDE) and Test-Driven Development (TDD), in addition to incorporating the principles that govern agile software development methodologies, such as SCRUM and XP. The methodology presented here, called Test-Driven Development Methodology for IoTS (TDDM4IoTS), has been proposed after an exhaustive review of different software development methodologies, leading us to conclude that none of them are specially oriented towards the development of IoTS. The methodology mainly consists of eleven phases, whose order of application can be established by the team that will develop the project in question. In this paper, we suggest an order to follow, as well as existing software tools that could be used as support for obtaining the corresponding deliverables at each phase.

Keywords: Software development methodology · Test-Driven Development · Model-Driven Engineering · Agile methodologies · Internet of Things (IoT) · IoT-based Systems

1 Introduction

In Software Engineering, proposals for new programming languages and paradigms have always been the main issue, closely followed by methodologies. Thus, structured programming emerged first, and then appropriate methodologies for Structured Analysis and Design (SAD) were proposed. Likewise, object-oriented programming was firstly proposed in 1972 [1], while proposals for Object-Oriented Analysis and Design (OOAD) and a methodology for the development of Object-Oriented Software were published in 1978 [2] and 1982 [3]. With the emergence of the Internet era and the World Wide Web (WWW), developers were faced with the need to adapt existing methodologies for the development of Web-based systems. The first documented Web development methodology was presented by Schwabe and Rossi in 2002 [4]. Thus, it has traditionally been considered necessary to review development methodologies after the emergence of new technological paradigms in Software Engineering.

Nowadays, IoT is one of the most prominent technological paradigms. This term, coined by Ashton [5], stems from the objective of "digitalizing physical objects" so that they can seamlessly interact with each other and with the people surrounding them to improve their lifestyles and productivity [6]. IoT is the result of the confluence/collaboration of several research areas, such as communication and cooperation, location and identification, sensor and actuator networks, integrated processing of distributed information, artificial intelligence and adaptive user interfaces, to name just a few of the most important converging fields.

The first IoTS were developed using ad-hoc methodologies specific to each development team, arising from the corresponding adaptation of methodologies employed in the development of more traditional information systems (IS).

However, the development of IoTS differs from the development of traditional computer systems in several key aspects. For instance, the development of IoTS necessarily involves deploying and setting up hardware components (sensors, actuators, controllers…) to interact with the physical and digital environment, which is not the usual case in traditional IS. Each of the hardware devices deployed in the environment (such as sensors, actuators and single board computers) requires a specific programming and configuration, as well as the implementation of information dissemination mechanisms (Publish/Subscribe or Request/Response are among the most common) to efficiently distribute data and create complex data flows between them.

On the other hand, in both traditional IS and IoTS, end-user client applications, mainly web-based [7] or mobile-based [8], must be implemented to interact with people, depending on the needs of the users themselves [9]. However, in the literature, the vast majority of IoTS development methodologies are focused exclusively on the implementation of either configuration software for IoT devices or a set of end-user applications, but they do not cover both aspects at the same time. In addition, none of the studied methodologies incorporate feasibility analysis or maintenance stages, instead they focus on software design and code generation. In this paper, we propose a methodology for the development of IoTS that covers all these aspects at the same time. Consequently, the main objectives of this paper are: (1) To present an exhaustive review of existing IoTS development methodologies, based on TDD, MDE and/or agile methodologies; (2) To check that there is no methodology specifically designed for the development of IoTS; and (3) To propose a new development methodology for IoTS that, in addition to the software in charge of business logic and user-system interaction, addresses the configuration and deployment of the hardware (sensors, actuators, processors,…) and the programming of single board computers (Arduino, Raspberry,…), so that they can perform an adequate pre-processing of the data captured by the sensors.

The remainder of this paper is structured as follows: Sect. 2 presents the state of the art on IoTS development methodologies based on TDD, MDE, and/or agile development methodologies. Section 3 proposes a new methodology for the development of IoTS that attempts to overcome the absence of a specific methodology for the development of IoTS. Finally, Sect. 4 outlines our conclusions and future work.

2 State of the Art

We searched for published papers on IoTS development methodologies at the Web of Science platform. Books, book chapters, and articles published in prestigious journals were selected, because they are considered more relevant, and in English, as this is the internationally adopted language for scientific publications. The search terms we used are shown in the central column of Table 1.

Table 1. Keywords and query strings used and number of search results obtained

No.	Query structure	Results
#1	TS = (IoT OR "Internet of Things")	15.597
#2	TS = (Framework OR Method*)	8.957.432
#3	TS = (Development OR Deploy OR Implement* OR Design OR construct*)	7.384.102
#4	TS = (Agile OR SCRUM OR XP OR "Extreme Programming" OR "Agile Inception" OR "Design Sprint" OR Kanban)	14.452
#5	TS = (TDD OR "Test-Driven Development" OR MDE OR "Model-Driven Engineering" OR MDA OR "Model-Driven Architecture" OR MDD OR "Model-Driven Development" OR "Model-Driven Design")	71.897
#6	#1 AND #2 AND #3	3.303
#7	#4 OR #5	86.224
#8	**#6 AND #7**	**38**

As a result, we obtained 38 documents (see last row of Table 1). After a thorough review of these documents, those that did not present a development methodology were discarded, and 12 papers (shown in Table 2) were finally selected for further analysis.

Table 2. Methodologies for the development of IoTS

Ref.	Approaches	General IoTS	Domain
[10]	MDE	✓	Intelligent street lights
[11]	MDD*, SOA♣	✗	IIoT♠, Automobiles
[12]	MDD*, MDA♥	✗	Mobile applications
[13]	Design based on components, BIP♦, Incremental design	✗	Wireless Personal Area Network Systems
[14]	MDD*	✓	Domotics, IIoT♠
[15]	MDE	✗	Health monitoring
[16]	SOA♣, Principles of agile development	✓	Environmental and risk management systems for IIoT♠

(*continued*)

Table 2. (*continued*)

Ref.	Approaches	General IoTS	Domain
[17]	SCRUM frame, Metamodels, SOA♣	✓	Smart Homes
[18]	MDE, SOA♣	✓	General
[19]	MDA♥	✓	Wireless Sensor Network
[20]	Waterfall, Agile principles	✓	Not applied
[21]	Division by roles or responsibilities	✓	Intelligent Buildings

*Model-Driven Development/Design; ♣Service-Oriented Architecture; ♥Model-Driven Architecture; ♠Industrial IoT; ♦Behavior Interaction Priority; ✓ Methodology for IoTS in general; ✗ Methodology for specific IoTS

2.1 Foundations of the Reviewed Methodologies

None of the analyzed TDD-related documents presented a development methodology for IoTS, unlike those related to MDE and agile development methodologies. Table 2 shows the references where the different methodologies were found, as well as the approaches they are based on, in addition to the type of IoTS and the domain for which they were developed or to which they were applied.

In TDD4IoTS, we have integrated some of the most common methodological stages that are proposed in the studied literature to solve the challenges of IoTS. In addition to them, we have incorporated the advantages of TDD to increase software quality (fulfillment of requirements, bug detection, improved software reliability, etc.).

2.2 Analysis of Existing Methodologies

The study of system requirements is the first step in the development of a system. Therefore, it should be the first phase in the methodology applied to its development. Table 3 shows a comparison of existing methodologies, focused on requirements analysis, detailing all the tools and models used for the development of IoTS.

Analyzing the state of the art on development methodologies for IoTS, we realized that some works [10, 11] do not mention the system requirements. Consequently, these methodologies do not consider the requirements analysis phase. The remaining of the reviewed methodologies agree on the importance of requirements analysis for the development of an IoTS. The methodology presented in [12] describes requirements analysis in greater depth, and presents some tools that developers can use to collect and analyze requirements. Whereas methodologies in [13–15] assume that the requirements are available before the development starts, conversely, those in [16, 17] consider that requirements are rarely available at the beginning of the development of an IoTS. In our contribution, we are more inclined towards the latter. Therefore, we propose TDD4IoTS as a methodology that sufficiently emphasizes the phase of obtaining and analyzing requirements.

The nature of IoTS makes it important to carefully consider all system states and transitions, since the system will have to react to events occurring in the environment

Table 3. Comparison of methodologies

Reference	Requirement analysis	Use UML	UML diagrams						Business Process Model and Notation (BPMN)
			Use cases	Activities	Classes	States	Sequences	Deployment	
[10]	✗	✓	✗	✗	✗	✗	✗	✗	✗
[11]	✗	✓	✗	✗	✗	✗	✗	✗	✗
[12]	~	✓	✓	✓	✓	✗	✗	✗	✓
[13]	~	✗	✗	✗	✗	✗	✗	✗	✗
[14]	~	✓	✗	✗	✗	✗	✗	✗	✗
[15]	~	✓	✗	✓	✓	✓	✗	✗	✗
[16]	✓	✓	✓	✓	✗	✗	✗	✗	✗
[17]	~	✓	✗	✓	✓	✓	✓	✗	✗
[18]	✓	✓	✓	✓	✗	✗	✗	✗	✗
[19]	✓	✓	✗	✓	✓	✗	✗	✓	✗
[20]	~	✗	✗	✗	✗	✗	✗	✗	✗
[21]	~	✓	✗	✗	✗	✗	✗	✗	✗

✓ Used or considered; ~ Mentioned; ✗ Not specified

it controls. That aspect has only been taken into consideration by [15] and [17]. In addition, deployment diagrams, which can show interlinks between the (software- and hardware-based) system components, are only used in [19].

One approach that can reduce and alleviate the work of developers is specifying the context of the environment that IoTS must control: available networks, quality of service (QoS), privacy levels, the physical environment in which the system will be deployed, as well as the preferences that the user may have, such as aesthetic changes and accessibility parameters. This specification may also help in making decisions regarding the specific technologies and tools to use. These aspects are considered important in the present work, although they are not included in any of the reviewed methodologies.

Another widely forgotten aspect of the development of IoTS in the reviewed bibliography is information storage. Although most authors highlight the importance of information storage, they assume that it is ready to use at the beginning of the development, via existing software components that are not significant in the development process. However, we consider that the type of database (relational, NoSQL,…) that is used must be carefully chosen during the development process, in addition to adequately designing its structure and selecting the resources dedicated to its management (database engine, local server or in the cloud, etc.).

Consequently, we have reached the conclusion that existing methodologies have important shortcomings in terms of their applicability to IoTS. Therefore, we present a new methodological proposal to better tackle the specific requirements of IoTS in the next section.

3 An Overview of TDDM4IoTS

We present a new methodology specifically designed for the development of IoTS that integrates ideas from the most prominent software development methodologies and tries to mitigate the weaknesses found in the reviewed methodologies. In fact, it is based on the TDD methodology phases [22], while applying the fundamentals of MDE and the principles of agile methodologies. That is why we have named it Test-Driven Development Methodology for IoT-based Systems (TDDM4IoTS). Additionally, it emphasizes the use of tools that, according to experts, ensure that the software meets the requirements that the customer has provided.

We propose TDDM4IoTS as a methodology independent of specific automation tools or frameworks, so that developers are free to choose the appropriate tool(s) to be used, depending on their needs and preferences. Nonetheless, we are working on the development of an automated tool to support TDDM4IoTS in all its phases. For the requirements specification, we propose using use cases instead of describing the requirements in natural language, which is more usual and what makes them error-prone and full of ambiguities [16]. One of the goals of our automated tool will be to reduce ambiguities at this level. The use cases, together with the conceptual class model, will allow us to automatically generate both the tests and parts of the software that must pass those tests. The developers will focus on specifying and analyzing system requirements, as well as completing and adapting the automatically generated software.

The phases of the life cycle of TDDM4IoTS, shown in Fig. 1, take into account the development of all types of IoTS, which is why the order and frequency of application, as well as the allocation of resources for each phase will depend on both the nature of the project and the knowledge, skills, experience and number of project members. Nonetheless, the order of application suggested by the numbers shown in Fig. 1 would be valid for the development of a large number of IoTS.

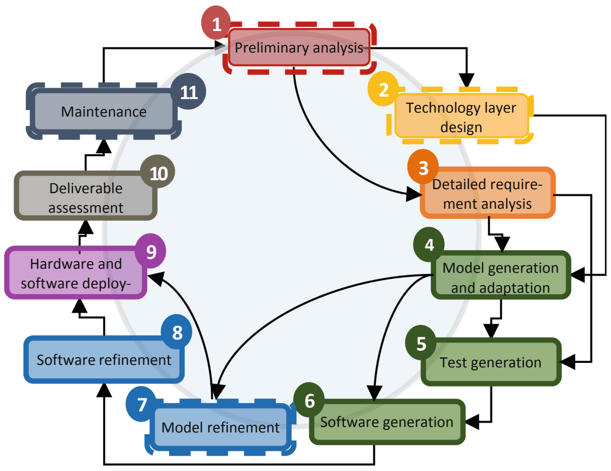
1 Preliminary analysis
2 Technology layer design
3 Detailed require-ment analysis
4 Model generation and adaptation
5 Test generation
6 Software generation
7 Model refinement
8 Software refinement
9 Hardware and software deploy-
10 Deliverable assessment
11 Maintenance

Fig. 1. Phases of the life cycle of TDDM4IoTS

These phases will be repeated iteratively for each deliverable. However, in the development of some deliverables, it may not be necessary to apply some of these phases (drawn with a dashed line in Fig. 1). For example, it may not be necessary to perform the preliminary analysis in a second iteration, or to carry out the refinement of the model in the development of a given deliverable. The development team must estimate the effort and duration of the process to obtain each deliverable or component into which any project is usually divided. Negotiation between the client and the development team on the priority (development order) of each deliverable will be of vital importance to the success of the project, unlike the SCRUM development framework, where the client (product owner) assigns development priorities to the deliverables (Sprints) [23–25].

TDDM4IoTS requires the *project facilitator*'s responsibility to be assigned to the member with the most experience in project management who possesses the characteristics of a leader. Developers who follow TDDM4IoTS are not subject to task impositions but to negotiations. The facilitator is not responsible for the whole project, s/he is just the manager of the negotiation between the development teams. The responsibility of the project rests with each of its members. Consequently, TDDM4IoTS adopts a horizontal management approach with shared responsibility, leaving the team members to self-organize. The number of developers will depend on the project size, and considering that we suggest "agile" teams, they should not exceed ten [23, 26]. The project members have different responsibilities, depending on their roles, shown in Table 4. Each development team is made up of a maximum of three developers, and should be balanced in terms of both knowledge and experience [26]. One of its members will be (informally) designated as a *counselor*, if necessary.

Consequently, while the participants can play three main roles in SCRUM, which are detailed in [29, 30], together with their responsibilities, we consider four roles in TDDM4IoTS, as shown in Table 4.

We can conclude that: (1) SCRUM is based on the principles for agile software development. (2) IoT encompasses different and varied aspects, such as the complexity of software development, hardware deployment, indispensable communications, cloud

Table 4. Roles and responsibilities in TDDM4IoTS

Role[a]	Description	Responsibilities
Project facilitator	Expert with extensive experience in project management and in the agile development of IoTS. Conflict solver, trainer, facilitator and with the innate characteristics of a leader [27, 28]	(1) Support the development team in achieving its objectives. (2) Contribute with their experience to the development of deliverables. (3) Negotiate with the client aspects of the development (order of deliverables, time, resources,...)[b]
Counselor	Member of the development team that becomes a "leader" (without a formal designation) because of his performance	Instruct his/her teammates on the subjects of his/her domain

(*continued*)

Table 4. (*continued*)

Role[a]	Description	Responsibilities
Customer/End user	Person with good communication and knowledge of all the functionality of the IoTS who commissions the development team to develop it	(1) Contribute to the requirements of the IoTS. (2) Approve the functionality of the finished deliverables and the final IoTS
Development team	Multidisciplinary group of experts with knowledge of the different project domains, and which is responsible for the development of the IoTS. Facilitators of knowledge and experience	(1) Negotiate with the client on aspects of the IoTS development (order of deliverables, time, resources,…)[b]. (2) Create deliverables that fully meet customer requirements

[a]Depending on the project, experts in other domains may be needed to provide their services for a specific time.
[b]This responsibility is shared by two roles.

storage and processing, and the intercommunication between these elements, among others. (3) SCRUM and XP ignore non-functional requirements, which are an important aspect in IoTS [31]. These three premises make it quite difficult to make a simple adaptation of SCRUM for the development of IoTS. This is why we intend to incorporate the best features of the manifesto for the agile development of software into TDDM4IoTS, adapting them to the particularities of IoTS.

The following subsections describe how to apply the foundations on which TDDM4IoTS is based and its phases, as well as the activities to be carried out and possible tools to be used in each of them.

3.1 Foundations of TDDM4IoTS

The main foundations on which TDDM4IoTS is based are described below.

Values and Principles for Agile Software Development. Agile software development is governed by four values, namely: (i) Individuals and interactions before processes and tools, (ii) Software working before extensive documentation, (iii) Collaboration with the customer before contractual negotiation, and (iv) Response to change before following a plan [29, 30]. Agile methodologies have revolutionized the software development process, demonstrating that development teams that have completed their projects successfully and on time did not respect rigid and heavy methodologies [30]. To comply with these values, twelve principles are detailed that a methodology must comply with to be qualified as an agile methodology [29, 30].

TDDM4IoTS complies with the twelve principles, bearing in mind that, as already said, IoTS comprise the deployment of a series of hardware devices in addition to the corresponding software. Therefore, a deliverable will be a finished element of the system, obtained as a result of an iteration, which is important for the customer.

TDD as an Agile Methodology. TDD [32–34] is one of the methodologies used for the development of traditional IS that has not been applied to the development of IoTS. As this type of methodology guarantees that the software of the developed system satisfies the requirements that the user has provided, we have incorporated it into our proposal. Nowadays, many integrated development environments (IDE) support TDD, such as IntelliJ IDEA, .Net, or Eclipse, to name just a few [35].

TDD, which is considered an agile methodology, is applied to ensure the quality of the deliverables. It has three phases in its development cycle [33]. By first writing the tests and then writing the code that has to pass the tests, it is ensured that the software does exactly what the customer wants (i.e., tests formally specify use cases) [22].

The agile software development methodologies that have been considered in this research are XP (eXtreme Programming) [36] and SCRUM [23]. We are interesting in how they approach software development projects, and especially in: delivery frequencies, acceptance of changes, giving greater importance to the deliverable than to an exhaustive documentation, and the quality of the software (tested and approved by the client) as one of the most important features, among other principles of agile methodologies that must be complied with. On the one hand, XP is one of the methodologies that has promoted TDD [35]. On the other hand, SCRUM delivers tested software, that is, software that has passed the tests, without specifying if the tests are written before or after the software code. However, it separates the tests from the development very well, as another (tester) team tests the software.

MDE. The heterogeneity of IoTS technologies and standards make MDE an important foundation for TDDM4IoTS. One objective is to reuse these models (with or without adaptations) in the development of other systems, for which these models could be converted into executable code [37].

3.2 Phases of TDDM4IoTS

The phases of TDDM4IoTS are designed so that the tools used by the developers in each phase are of their own choice. The tools that developers can use to meet the objectives of each phase are recommended. The first challenge to overcome is the communication among the project members and among the members of each development team. The best way to achieve an effective communication is face-to-face, as indicated in the agile manifesto [29, 30], involving periodic meetings set by the teams either at the beginning of the project or at the beginning of the development of the corresponding deliverable. Establishing the frequency of such meetings is the responsibility of all the project members. However, three to five meetings per week are recommended. Mechanisms should be sought so that development teams feel motivated and are at ease with their work, with the organization, with the rest of the project members and with their own team, in order to achieve a better performance [38].

(1) Preliminary Analysis. The objective of this phase is to obtain a feasibility study of the (complete) system regarding its technological, economic and operational aspects, as well as an analysis of the context in which the system will be deployed and the interaction with the end user, based on the requirements and objectives expressed by the client. The activities to be carried out may include:

- *Requirements analysis.* This analysis determines two types of requirements: (i) functional, which are the customer's specifications (list of deliverables), determining its implementation priority, and (ii) non-functional, also called quality attributes, such as scalability, intrusiveness, environment aesthetics (deployment, appearance, etc.).
- *Technology analysis.* This determines the technology to be used that meets the system requirements: (i) Hardware resources already available; (ii) Existing hardware, taking into account its characteristics and costs; (iii) Tools (software) for hardware configuration; (iv) Tools for both software development and storage management; (v) Third-party hardware for specific tasks, and (vi) First-party hardware development, if necessary and feasible.
- *Analysis of the environment.* The client can specify the characteristics of the environment in which the system will be deployed. For example, available and/or feasible power supply points, data communication networks – Internet access, customer's preferred methods of interaction, etc.
- *Feasibility analysis.* Among the types of feasibility to be analyzed are: (i) Technical feasibility, for which the questions to be answered would be: (a) Are there the technologies necessary to develop the project? (b) Are trained personnel available to develop the system? (c) Can the system be developed? (ii) Economic feasibility, where the question to be answered would be: Is there an adequate budget to develop the project? (iii) Operational feasibility, which is important for the system to remain in operation after it is implemented, so these questions must be answered: (a) Will it be possible to install the system once it is completed? (b) Will the system be able to operate with available resources? (c) Are there the necessary guarantees for the system to continue operating once installed? (d) Will the IoTS have a properly scheduled maintenance?

Since IoTS may involve the use of multiple technologies, the availability of each of them could alter the order or priorities of the deliverables. Therefore, it is necessary that this first phase is carried out globally at the beginning of the project, and reviewed at the beginning of the development of each deliverable, to consider the changes related to the technology (new devices, new tools, etc.) and requirements that might arise during the development of the project.

There are no specific tools to meet the objective of this particular phase. However, for the project planning, it is suggested to use a free software tool, such as OpenProj, GanttProject, dotProject, and for those who prefer proprietary software, MS-Project, among many others.

The result of this phase is fundamental for the first negotiations with the client, concerning deliveries, times, and budget. All this is preliminary and cannot be considered definitive, as it can be negotiated between both parties at the beginning of the development of each deliverable.

(2) Technology Layer Design. The objective of this phase is to obtain the first design of the global system that will serve as a guide for the development teams. This is very important, given that IoT involves emerging and heterogeneous technologies, and so far, it is difficult to find a professional who can master all the technologies involved in the development of IoTS [21].

For the system design, circuit design tools may be used that can represent as clearly as possible the elements that have been determined for the project. For example, if Arduino boards are used, online tools such as Circuito.io or Fritzing can be used. The development team may complement the designs that have been obtained. If necessary, the resulting design can be updated at the end of each deliverable. This will be one of the documents of greatest interest for all the development teams of the project. Therefore, it must always be accessible to all of them.

In this phase, the architecture with which the system will be implemented should be designed. Consequently, the outcome of this phase will serve as a guide for the entire development process.

(3) Detailed Requirement Analysis. The objective of this phase, which will be executed for each system deliverable, is to obtain the detailed requirements of the deliverable to be developed. In addition, the client will be asked to describe the tests together with the developers, to reduce ambiguities. For the requirements specification, it is recommended to use tools that are understandable for all the stakeholders, considering the customer as one of them. One of the notations to be used could be UML, with its different tools, such as use cases and semi-structured use cases, using pre-established templates, seeking to eliminate ambiguities, in addition to other tools, such as state and deployment diagrams, which help to understand the requirements provided by the client.

(4) Model Generation and Adaptation.
The purpose of applying MDE in TDDM4IoTS is to reuse models and thus improve the productivity of the development team. Therefore, in this phase, new models will be generated or existing models will be adapted. Using models abstracts out or at least minimizes the heterogeneity aspects of technologies, enabling a good communication between the development teams and the clients. Also, depending on the software tools used for modeling, the software can be automatically generated across models, from abstract models to specific models [10, 18]. One of the models is the class diagram, which is used to generate the database.

The suggested modeling languages are: UML, BPMN [12], the adaptation of one of them [14], their combination or the creation of a new one, always seeking to cover the particular characteristics of the IoTS to be developed and to be easily understood by those involved in the project. Among the automated tools for system modeling are StarUML, ArgoUML, MagicDraw and Visual Studio .Net, for example. For those who opt for BPMN, they can select tools such as Lucidchart or VisualParadigm, among many others.

(5) Test Generation. TDDM4IoTS follows the TDD paradigm, so it must generate the tests that the software must pass to ensure the quality of the system. The tests can be grouped into two groups: (1) The tests written by the developers, within which there are: (a) unit tests, which are the most exhaustive, to examine the complete functioning of a function, i.e., it is tested whether the function yields the results that it should yield and even whether it supports the exceptions that may arise; and (b) integration tests, which also involve system tests. And (2) tests documented by the client, which are basically acceptance tests, including functional tests [22].

The automated tools for this phase will depend on the IDE that has been selected for development, although so far no tool automatically generates tests based on requirements.

(6) Software Generation. This phase is based on models and tests. The developer must write/generate the code for the tests to be passed [33, 39]. The new models generated and/or the existing models adapted are part of the system documentation. Therefore, unlike SCRUM and XP methodologies, which suggest that the documentation related to the analysis and design of the solution should be written at the end of each deliverable development [30], TDDM4IoTS propose to do it before the code generation, as there are tools that, based on the models, help in this task. Also, when surveying software developers who work or have worked with SCRUM and/or XP methodology, most agree that the solution that will be implemented later should at least be sketched at the beginning of the life cycle of the methodology. This corroborates what is stated in TDDM4IoTS regarding the analysis and design of the solution, which should be carried out during the development of the deliverables.

Once the software has been generated, it is checked that it passes the corresponding tests, and then this phase finishes. The result of this phase is the software tested and working, though it will almost always have to be refined later.

To generate the code, some of the tools mentioned in the model generation and refinement phase can be used. In fact, there are several tools, both free software and proprietary software, which even allow simulation of the system behavior.

(7) Model Refinement. UML models facilitate model refinement. Note that the solution found is a functional solution, but not necessarily an optimal solution. Hence, it is important to make the necessary adjustments and refinements, such as improving the robustness, scalability or reusability of the deliverables that made up of the IoTS to be developed [40]. The result of this phase will be the definitive model, based on which the system code will be generated.

The tools recommended to perform the tasks of this phase are the same tools as in the model generation and adaptation phase.

(8) Software Refinement. The work of the developers in this phase will be to guarantee the software quality, eliminating redundancies and making the software easy to maintain. The tools that support this activity will be those provided by the chosen IDE. It must be ensured that the final software meets the specifications of a clean code [34].

(9) Hardware and Software Deployment. Once the software has been tested (simulated), it is implemented and deployed in the devices and resources to be used in the system, confirming compliance with the final requirements negotiated between the client and the development team. At this point, and for the first deliverable, both the information storage system and the applications that will serve for the user-system interaction will have already been configured [41, 42]. For subsequent deliverables, the necessary changes will be made to this assembled infrastructure to add the new deliverables. Since the subsequent deliverables depend on the technology already installed, the operation of the (new) integrated system must be guaranteed before continuing the development process.

The tools to be used in this phase will depend on the technology used (single board computers, embedded sensors…).

(10) Deliverable Assessment. Once the development of the deliverable is complete (it had to pass the necessary tests to guarantee its operation), the integration tests, the system tests and, of course, the functional tests must be performed once again at this stage [43, 44].

(11) Maintenance. IoTS combine the complexity of software maintenance with the minor complexity of hardware maintenance. If an IoTS physical component (e.g., sensor, actuator, board computer, among others) fails, it is replaced by some other similar device. However, if the software requirements change, the code has to be modified, given that it has no spare parts [45]. Additionally, IoTS need constant operational maintenance (batteries or power lines, and connectivity, among others).

4 Conclusions and Future Work

Based on a thorough review of the methodologies used for the development of IoTS, it has been found that there is no standard methodology for this application domain. This fact has led IoTS developers to use methodologies designed to develop other types of more traditional IS and to make ad-hoc adjustments to meet the particular needs of each project. Moreover, the use of these non-specific methodologies for the development of IoTS has overlooked the need to specifically consider important aspects of these systems, such as their special requirements and the particular characteristics of the hardware to be deployed in them.

Consequently, a new methodology called TDDM4IoTS has been proposed to specifically approach the development of IoTS, which includes the most important foundations of TDD, MDE, and agile development, and which tries to solve each of the aspects that have been detected as weaknesses in the reviewed methodologies that have been used for the same purpose.

One of our next projects will be to validate TDDM4IoTS, to determine its effectiveness and acceptance. In addition, a tool will be developed to support the automated generation of the deliverables corresponding to each phase of TDDM4IoTS. In particular, we will focus on test generation.

References

1. Dahl, O.-J., Hoare, C.A.R.: Chapter III: Hierarchical program structures. In: Structured Programming, pp. 175–220. Academic Press Ltd. (1972)
2. Ingalls, D.H.H.: The Smalltalk-76 programming system design and implementation. In: Proceedings of the 5th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages - POPL 1978, pp. 9–16. ACM (1978)
3. Pashtan, A.: Object oriented operating systems: an emerging design methodology. In: Proceedings of the ACM 1982 Conference on ACM 1982, pp. 126–131. ACM (1982)
4. Schwabe, D., Rossi, G.: The object-oriented hypermedia design model. Commun. ACM **38**(8), 45–46 (2002)
5. Ashton, K.: That ‘Internet of Things’ thing. RFID J. **22**(7), 97–114 (2009)
6. Ray, P.P.: A survey on Internet of Things architectures. J. King Saud Univ. – Comput. Inf. Sci. **30**(3), 291–319 (2018)
7. Leotta, M., et al.: An acceptance testing approach for Internet of Things systems. IET Softw. **12**(5), 430–436 (2018)

8. Benedetto, J.I., González, L.A., Sanabria, P., Neyem, A., Navón, J.: Towards a practical framework for code offloading in the Internet of Things. Future Gener. Comput. Syst. **92**(March), 424–437 (2019)
9. Cervantes-Solis, J.W., Baber, C., Khattab, A., Mitch, R.: Rule and theme discovery in human interactions with an Internet of Things. In: Proceedings of the 2015 British HCI Conference on - British HCI 2015, pp. 222–227. ACM, UK (2015)
10. Ciccozzi, F., Spalazzese, R.: MDE4IoT: supporting the Internet of Things with model-driven engineering. In: Badica, C., et al. (eds.) IDC 2016. SCI, vol. 678, pp. 67–76. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-48829-5_7
11. Khaleel, H., et al.: Heterogeneous applications, tools, and methodologies in the car manufacturing industry through an IoT approach. IEEE Syst. J. **11**(3), 1412–1423 (2017)
12. 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. **6**(3), 771–784 (2018)
13. Lekidis, A., Stachtiari, E., Katsaros, P., Bozga, M., Georgiadis, C.K.: Model-based design of IoT systems with the BIP component framework. J. Softw.: Pract. Exp. **48**(6), 1167–1194 (2018)
14. 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. **8**(1), 1–21 (2017)
15. 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. **129-2**, 548–571 (2017)
16. Usländer, T., Batz, T.: Agile service engineering in the industrial Internet of Things. Future Internet **10**(10), 100 (2018)
17. 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. **94**, 250–271 (2019)
18. Sosa-Reyna, C.M., Tello-Leal, E., Lara-Alabazares, D.: Methodology for the model-driven development of service oriented IoT applications. J. Syst. Architect. **90**, 15–22 (2018)
19. de Farias, C.M., et al.: COMFIT: a development environment for the Internet of Things. Future Gener. Comput. Syst. **75**, 128–144 (2017)
20. Fortino, G., et al.: Towards multi-layer interoperability of heterogeneous IoT platforms: the INTER-IoT approach. In: Gravina, R., Palau, C.E., Manso, M., Liotta, A., Fortino, G. (eds.) Integration, Interconnection, and Interoperability of IoT Systems. IT, pp. 199–232. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-61300-0_10
21. Patel, P., Cassou, D.: Enabling high-level application development for the Internet of Things. J. Syst. Softw. **103**, 62–84 (2015)
22. Martin, R.C.: Clean Coder Blog, TDD (2017). https://blog.cleancoder.com/. Accessed 11 Sept 2019
23. Rising, L., Janoff, N.S.: Scrum software development process for small teams. IEEE Softw. **17**(4), 26–32 (2000)
24. Heeager, L.T., Nielsen, P.A.: A conceptual model of agile software development in a safety-critical context: a systematic literature review. Inf. Softw. Technol. **103**, 22–39 (2018)
25. Abrahamsson, P., Salo, O., Ronkainen, J., Warsta, J.: Agile Software Development Methods: Review and Analysis. VTT Publications 478 (2002)
26. 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$\rightarrow$XU). In: 29th Annual International Computer Software and Applications Conference (COMPSAC 2005), vol. 2, pp. 169–172. IEEE, UK (2005)
27. Mowday, R.T.: Leader characteristics, self-confidence, and methods of upward influence in organizational decision situations. Acad. Manag. J. **22**(4), 709–725 (1979)

28. Koo, K., Park, C.: Foundation of leadership in Asia: leader characteristics and leadership styles review and research agenda. Asia Pac. J. Manag. **35**(3), 697–718 (2018)
29. Beck, K.: Manifesto for Agile Software Development (2001). http://agilemanifesto.org/. Accessed 11 May 2019
30. Hazzan, O., Dubinsky, Y.: The agile manifesto. In: Zdonik, S., et al. (eds.) Agile Anywhere. SCS, pp. 9–14. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10157-6_3
31. Sachdeva, V., Chung, L.: Handling non-functional requirements for big data and IoT Projects in Scrum. In: 2017 7th International Conference on Cloud Computing, Data Science & Engineering - Confluence, pp. 216–221. IEEE (2017)
32. Tort, A., Olivé, A., Sancho, M.R.: An approach to test-driven development of conceptual schemas. Data Knowl. Eng. **70**(12), 1088–1111 (2011)
33. Janzen, D., Saiedian, H.: Test-driven development: concepts, taxonomy, and future direction. Computer **38**(9), 43–50 (2005)
34. Martin, R.C.: Clean Code, A Handbook of Agile Software Craftsmanship, 1st edn. Pearson Education Inc., Boston (2011)
35. Madeyski, L., Kawalerowicz, M.: Continuous test-driven development: a preliminary empirical evaluation using agile experimentation in industrial settings. In: Kosiuczenko, P., Madeyski, L. (eds.) Towards a Synergistic Combination of Research and Practice in Software Engineering. SCI, vol. 733, pp. 105–118. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-65208-5_8
36. Braithwaite, K., Joyce, T.: XP expanded: distributed extreme programming. In: Baumeister, H., Marchesi, M., Holcombe, M. (eds.) XP 2005. LNCS, vol. 3556, pp. 180–188. Springer, Heidelberg (2005). https://doi.org/10.1007/11499053_21
37. Sosa-Reyna, C.M., Tello-Leal, E., Lara-Alabazares, D.: An approach based on model-driven development for IoT applications. In: Proceedings of the 2018 IEEE International Congress on IoT, ICIOT, 2018 IEEE World Congress on Services, pp. 134–139. IEEE, San Francisco, EEUU (2018)
38. Rasch, R.H., Tosi, H.L.: Factors affecting software developers' performance: an integrated approach. MIS Q. **16**(3), 395–413 (1992)
39. Nyznar, M., Pałka, D.: Generating source code templates on the basis of unit tests. In: Grzech, A., Świątek, J., Wilimowska, Z., Borzemski, L. (eds.) Information Systems Architecture and Technology: Proceedings of 37th International Conference on Information Systems Architecture and Technology – ISAT 2016 – Part II. AISC, vol. 522, pp. 213–223. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-46586-9_17
40. Chen, Z., Liu, Z., Ravn, A.P., Stolz, V., Zhan, N.: Refinement and verification in component-based model-driven design. Sci. Comput. Program. **74**(4), 168–196 (2009)
41. Monteiro, K., Rocha, E., Silva, E., Santos, G.L., Santos, W., Endo, P.T.: Developing an e-health system based on IoT, fog and cloud computing. In: 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), pp. 17–18. IEEE, Zurich (2018)
42. Guerrero-Ulloa, G., Rodríguez-Domínguez, C., Hornos, M.J.: IoT-based system to help care for dependent elderly. In: Botto-Tobar, M., Pizarro, G., Zúñiga-Prieto, M., D'Armas, M., Zúñiga Sánchez, M. (eds.) CITT 2018. CCIS, vol. 895, pp. 41–55. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-05532-5_4
43. Latorre, R.: Effects of developer experience on learning and applying unit test-driven development. IEEE Trans. Softw. Eng. **40**(4), 381–395 (2014)
44. Shihab, E., Jiang, Z.M., Adams, B., Hassan, A.E., Bowerman, R.: Prioritizing the creation of unit tests in legacy software systems. Softw.: Pract. Exp. **41**(10), 1027–1048 (2011)
45. Pressman, R.S., Maxim, B.: Software Engineering: A Practitioner's Approach, 8th edn. McGraw-Hill Education, Boston (2015)