TDDM4IoTS: Una metodología de desarrollo basada en pruebas para sistemas basados en Internet de las Cosas (IoT)

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

[1] Facultad de Ciencias de la Ingeniería, Universidad Técnica Estatal de Quevedo, Quevedo 120501, Ecuador
gguerrero@uteq.edu.ec
[2] Departamento de Ingeniería del Software, Universidad de Granada, 18071 Granada, España
gleiston@correo.ugr.es, {mhornos,carlosrodriguez}@ugr.es

Resumen. En este trabajo se presenta una metodología de desarrollo para Sistemas Basados en Internet de las Cosas (IoT) que recoge ideas de varios de los paradigmas de desarrollo de software más destacados en la actualidad, como la Ingeniería Dirigida por Modelos (MDE) y el Desarrollo Dirigido por Pruebas (TDD), además de incorporar los principios que rigen las metodologías ágiles de desarrollo de software, como SCRUM y XP. La metodología que aquí se presenta, denominada Test-Driven Development Method- ology for IoTS (TDDM4IoTS), ha sido propuesta tras una exhaustiva revisión de diferentes metodologías de desarrollo de software, lo que nos ha llevado a concluir que ninguna de ellas está especialmente orientada al desarrollo de IoTS. La metodología consta principalmente de once fases, cuyo orden de aplicación puede ser establecido por el equipo que vaya a desarrollar el proyecto en cuestión. En este trabajo, sugerimos un orden a seguir, así como herramientas software existentes que podrían ser utilizadas como soporte para la obtención de los correspondientes entregables en cada fase.

Palabras clave: Metodología de desarrollo de software - Desarrollo basado en pruebas - Ingeniería basada en modelos - Metodologías ágiles - Internet de las cosas (IoT) - Sistemas basados en IoT.

1 Introducción

En Ingeniería del Software, las propuestas de nuevos lenguajes y paradigmas de programación han sido siempre el tema principal, seguido de cerca por las metodologías. Así, primero surgió la programación estructurada, y después se propusieron metodologías adecuadas para el Análisis y Diseño Estructurados (SAD). Del mismo modo, la programación orientada a objetos se propuso por primera vez en 1972 [1], mientras que en 1978 [2] y 1982 [3] se publicaron propuestas para el Análisis y Diseño Orientados a Objetos (OOAD) y una metodología para el desarrollo de

Software Orientado a Objetos. Con la aparición de la era de Internet y la World Wide Web (WWW), los desarrolladores se enfrentaron a la necesidad de adaptar las metodologías existentes para el desarrollo de sistemas basados en la Web. El primer desarrollo Web documentado

M. Botto-Tobar et al. (Eds.): ICAT 2019, CCIS 1193, pp. 41-55, 2020.
https://doi.org/10.1007/978-3-030-42517-3_4

fue presentada por Schwabe y Rossi en 2002 [4]. Así, tradicionalmente se ha considerado necesario revisar las metodologías de desarrollo tras la aparición de nuevos paradigmas tecnológicos en la Ingeniería del Software.

Hoy en día, IoT es uno de los paradigmas tecnológicos más destacados. Este término, acuñado por Ashton [5], surge del objetivo de "digitalizar los objetos físicos" para que puedan interactuar sin problemas entre sí y con las personas que los rodean para mejorar su estilo de vida y su productividad [6]. IoT es el resultado de la confluencia/colaboración de varias áreas de investigación, como la comunicación y la cooperación, la localización y la identificación, las redes de sensores y actuadores, el procesamiento integrado de información dis- tribuida, la inteligencia artificial y las interfaces de usuario adaptativas, por nombrar sólo algunos de los campos convergentes más importantes.

Los primeros IoTS se desarrollaron utilizando metodologías ad hoc propias de cada equipo de desarrollo, surgidas de la correspondiente adaptación de metodologías empleadas en el desarrollo de sistemas de información (SI) más tradicionales.

Sin embargo, el desarrollo de IoTS difiere del de los sistemas informáticos tradicionales en varios aspectos clave. Por ejemplo, el desarrollo de IoTS implica necesariamente el despliegue y la configuración de componentes de hardware (sensores, actuadores, controladores...) para interactuar con el entorno físico y digital, lo que no es el caso habitual en los SI tradicionales. Cada uno de los dispositivos hardware desplegados en el entorno (como sensores, actuadores y ordenadores de placa única) requiere una programación y configuración específicas, así como la implementación de mecanismos de difusión de información (Publish/Subscribe o Request/Response se encuentran entre los más comunes) para distribuir datos de forma eficiente y crear complejos flujos de datos entre ellos.

Por otro lado, tanto en los SI tradicionales como en los IoTS, deben implementarse aplicaciones cliente de usuario final, principalmente basadas en web [7] o en móviles [8], para interactuar con las personas, dependiendo de las necesidades de los propios usuarios [9]. Sin embargo, en la literatura, la gran mayoría de las metodologías de desarrollo de IoTS se centran exclusivamente en la implementación de software de configuración para dispositivos IoT o de un conjunto de aplicaciones de usuario final, pero no cubren ambos aspectos al mismo tiempo. Además, ninguna de las metodologías estudiadas incorpora etapas de análisis de viabilidad o mantenimiento, sino que se centran en el diseño de software y la generación de código. En este trabajo proponemos una metodología para el desarrollo de IoTS que cubre todos estos aspectos al mismo tiempo. En consecuencia, los principales objetivos de este trabajo son: (1) Presentar una revisión exhaustiva de las metodologías de desarrollo de IoTS existentes, basadas en TDD, MDE y/o metodologías ágiles;
(2) Comprobar que no existe una metodología específicamente diseñada para el desarrollo de IoTS; y (3) Proponer una nueva metodología de desarrollo de IoTS que, además del software encargado de la lógica de negocio y la interacción usuario-sistema, aborde la configuración y despliegue del hardware (sensores, actuadores, procesadores,...) y la programación de ordenadores de placa única (Arduino, Raspberry,...), de forma que puedan realizar un adecuado pre-procesado de los datos captados por los sensores.

El resto de este documento se estructura como sigue: La sección 2 presenta el estado del arte de las metodologías de desarrollo de IoTS basadas en TDD, MDE y/o metodologías de desarrollo ágil. La Sección 3 propone una nueva metodología para el desarrollo de IoTS que intenta superar la ausencia de una metodología específica para el desarrollo de IoTS. Por último, la Secc. 4 esboza nuestras conclusiones y el trabajo

futuro.

2 Estado de la técnica

Se buscaron artículos publicados sobre metodologías de desarrollo de IoTS en la plataforma Web of Science. Se seleccionaron libros, capítulos de libros y artículos publicados en revistas de prestigio, por considerarse más relevantes, y en inglés, por ser éste el idioma adoptado internacionalmente para las publicaciones científicas. Los términos de búsqueda que utilizamos se muestran en la columna central de la Tabla 1.

Tabla 1. Palabras clave y cadenas de consulta utilizadas y número de resultados de búsqueda obtenidos

No.	Estructura de la consulta	Resultados
#1	TS = (IoT O "Internet de los objetos")	15.597
#2	TS = (Marco OR Método*)	8.957.432
#3	TS = (Desarrollo O Despliegue O Implantación* O Diseño O Construcción*)	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 Y #2 Y #3	3.303
#7	#4 O #5	86.224
#8	**#6 Y #7**	**38**

Como resultado, obtuvimos 38 documentos (véase la última fila de la Tabla 1). Tras un examen minucioso de estos documentos, se descartaron los que no presentaban una metodología de desarrollo, y finalmente se seleccionaron 12 documentos (que figuran en la Tabla 2) para su posterior análisis.

Tabla 2. Metodologías para el desarrollo de IoTS

Ref.	Enfoques	General IoTS	Dominio
[10]	MDE	✓	Alumbrado público inteligente
[11]	MDD*, SOA♣	✗	IIoT♠, Automóviles
[12]	MDD*, MDA♥	✗	Aplicaciones móviles
[13]	Diseño basado en componentes, BIP, Diseño incremental	✗	Sistemas inalámbricos de red de área personal
[14]	MDD*	✓	Domótica, IIoT♠
[15]	MDE	✗	Vigilancia de la salud
[16]	SOA♣, Principios de desarrollo ágil	✓	Sistemas de gestión medioambiental y de riesgos para las IIoT♠.

(*continuación*)

Cuadro 2 (continuación) (*continuación*)

Ref.	Enfoques	General IoTS	Dominio
[17]	Marco SCRUM, Metamodelos, SOA♣	✓	Casas inteligentes
[18]	MDE, SOA♣	✓	General
[19]	MDA♥	✓	Red de sensores inalámbricos
[20]	Cascada, principios ágiles	✓	No se aplica
[21]	División por funciones o responsabilidades	✓	Edificios inteligentes

*Model-Driven Development/Design; ♣Service-Oriented Architecture; ♥Model-Driven Architecture; ♠Industrial IoT; ¸Behavior Interaction Priority; ✓ Metodología para IoTS en general; ✗ Metodología para IoTS específicos.

2.1 Fundamentos de las metodologías examinadas

Ninguno de los documentos analizados relacionados con TDD presentaba una metodología de desarrollo para IoTS, a diferencia de los relacionados con MDE y metodologías ágiles de desarrollo. La Tabla 2 muestra las referencias donde se encontraron las diferentes metodologías, así como los enfoques en los que se basan, además del tipo de IoTS y el dominio para el que se desarrollaron o al que se aplicaron.

En TDD4IoTS, hemos integrado algunas de las etapas metodológicas más comunes que se proponen en la literatura estudiada para resolver los retos de IoTS. Además de ellas, hemos incorporado las ventajas de TDD para aumentar la calidad del software (cumplimiento de requisitos, detección de errores, mejora de la fiabilidad del software, etc.).

2.2 Análisis de las metodologías existentes

El estudio de los requisitos del sistema es el primer paso en el desarrollo de un sistema. Por lo tanto, debe ser la primera fase de la metodología aplicada a su desarrollo. La Tabla 3 muestra una comparación de las metodologías existentes, centradas en el análisis de requisitos, detallando todas las herramientas y modelos utilizados para el desarrollo de IoTS.

Analizando el estado del arte de las metodologías de desarrollo para IoTS, nos dimos cuenta de que algunos trabajos [10, 11] no mencionan los requisitos del sistema. En consecuencia, estas metodologías no consideran la fase de análisis de requisitos. El resto de las metodologías revisadas coinciden en la importancia del análisis de requisitos para el desarrollo de un IoTS. La metodología presentada en [12] describe el análisis de requisitos en mayor profundidad, y presenta algunas herramientas que los desarrolladores pueden utilizar para recopilar y analizar los requisitos. Mientras que las metodologías de [13-15] asumen que los requisitos están disponibles antes de que comience el desarrollo, por el contrario, las de [16, 17] consideran que los requisitos rara vez están disponibles al comienzo del desarrollo de un IoTS. En nuestra contribución, nos inclinamos más por lo segundo. Por ello, proponemos TDD4IoTS como una metodología que enfatiza suficientemente la fase de obtención y análisis de requisitos.

La naturaleza del IoTS hace que sea importante considerar cuidadosamente todos los estados y transiciones del sistema, ya que éste tendrá que reaccionar ante los

acontecimientos que se produzcan en el entorno

Tabla 3. Comparación de metodologías

Referencia	Análisis de los requisitos	Utilizar UML	Diagramas UML						Modelo y notación de procesos empresariales (BPMN)
			Casos prácticos	Actividades	Clases	Estados	Secuencias	Despliegue	
[10]	✗	✓	✗	✗	✗	✗	✗	✗	✗
[11]	✗	✓	✗	✗	✗	✗	✗	✗	✗
[12]	~	✓	✓	✓	✓	✗	✗	✗	✓
[13]	~	✗	✗	✗	✗	✗	✗	✗	✗
[14]	~	✓	✗	✗	✗	✗	✗	✗	✗
[15]	~	✓	✗	✓	✓	✓	✗	✗	✗
[16]	✓	✓	✓	✓	✗	✗	✗	✗	✗
[17]	~	✓	✗	✓	✓	✓	✓	✗	✗
[18]	✓	✓	✓	✓	✗	✗	✗	✗	✗
[19]	✓	✓	✗	✓	✓	✗	✗	✓	✗
[20]	~	✗	✗	✗	✗	✗	✗	✗	✗
[21]	~	✓	✗	✗	✗	✗	✗	✗	✗

✓ Utilizado o considerado; ~ Mencionado; ✗ No especificado

que controla. Este aspecto sólo se ha tenido en cuenta en [15] y [17]. Además, los diagramas de despliegue, que pueden mostrar las interrelaciones entre los componentes del sistema (basados en software y hardware), sólo se utilizan en [19].

Un enfoque que puede reducir y aliviar el trabajo de los desarrolladores es especificar el contexto del entorno que IoTS debe controlar: redes disponibles, calidad del servicio (QoS), niveles de privacidad, el entorno físico en el que se desplegará el sistema, así como las preferencias que pueda tener el usuario, como cambios estéticos y parámetros de accesibilidad. Esta especificación también puede ayudar a tomar decisiones sobre las tecnologías y herramientas específicas que se utilizarán. Estos aspectos se consideran importantes en el presente trabajo, aunque no se incluyen en ninguna de las metodologías revisadas.

Otro aspecto ampliamente olvidado del desarrollo de IoTS en la bib- liografía revisada es el almacenamiento de la información. Aunque la mayoría de los autores destacan la importancia del almacenamiento de la información, asumen que está listo para su uso al principio del desarrollo, a través de componentes de software existentes que no son significativos en el proceso de desarrollo. Sin embargo, consideramos que el tipo de base de datos (relacional, NoSQL,...) que se utilice debe ser cuidadosamente elegido durante el proceso de desarrollo, además de diseñar adecuadamente su estructura y seleccionar los recursos dedicados a su gestión (motor de base de datos, servidor local o en la nube, etc.).

En consecuencia, hemos llegado a la conclusión de que las metodologías existentes presentan importantes deficiencias en cuanto a su aplicabilidad al IoTS. Por lo tanto, en la siguiente sección presentamos una nueva propuesta metodológica para abordar mejor los requisitos específicos de IoTS.

3 Visión general de TDDM4IoTS

Presentamos una nueva metodología diseñada específicamente para el desarrollo de IoTS que integra ideas de las metodologías de desarrollo de software más destacadas e intenta mitigar las debilidades encontradas en las metodologías revisadas. De hecho, se basa en las fases de la metodología TDD [22], al tiempo que aplica los fundamentos de MDE y los principios de las metodologías ágiles. Por eso la hemos llamado Test-Driven Develop- ment Methodology for IoT-based Systems (TDDM4IoTS). Además, hace hincapié en el uso de herramientas que, según los expertos, garantizan que el software cumple los requisitos que el cliente ha proporcionado.

Proponemos TDDM4IoTS como una metodología independiente de herramientas o frameworks de automatización específicos, de modo que los desarrolladores sean libres de elegir la(s) herramienta(s) adecuada(s) a utilizar, en función de sus necesidades y preferencias. No obstante, estamos trabajando en el desarrollo de una herramienta automatizada que dé soporte a TDDM4IoTS en todas sus fases. Para la especificación de requisitos, proponemos utilizar casos de uso en lugar de describir los requisitos en lenguaje natural, que es lo más habitual y lo que los hace propensos a errores y llenos de ambigüedades [16]. Uno de los objetivos de nuestra herramienta automatizada será reducir las ambigüedades a este nivel. Los casos de uso, junto con el modelo conceptual de clases, nos permitirán generar automáticamente tanto las pruebas como las partes del software que deben superar dichas pruebas. Los desarrolladores se centrarán en especificar y analizar los requisitos del sistema, así como en completar y adaptar el software generado automáticamente.

Las fases del ciclo de vida de TDDM4IoTS, mostradas en la Fig. 1, tienen en cuenta el desarrollo de todos los tipos de IoTS, por lo que el orden y la frecuencia de aplicación, así como la asignación de recursos para cada fase dependerán tanto de la naturaleza del proyecto como de los conocimientos, habilidades, experiencia y número de miembros del proyecto. No obstante, el orden de aplicación sugerido por los números mostrados en la Fig. 1 sería válido para el desarrollo de un gran número de IoTS.

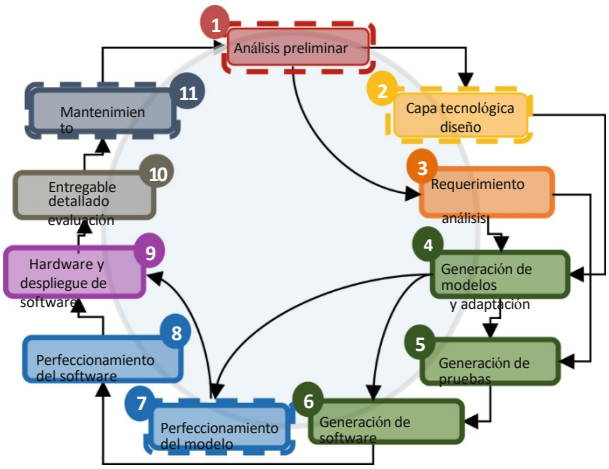

Fig. 1. Fases del ciclo de vida de TDDM4IoTS

Estas fases se repetirán de forma iterativa para cada entregable. Sin embargo, en el desarrollo de algunos entregables, puede que no sea necesario aplicar algunas de estas fases (dibujadas con una línea discontinua en la Fig. 1). Por ejemplo, puede que no sea necesario realizar el análisis preliminar en una segunda iteración, o llevar a cabo el perfeccionamiento del modelo en el desarrollo de un determinado entregable. El equipo de desarrollo debe estimar el esfuerzo y la duración del proceso para obtener cada entregable o componente en que suele dividirse cualquier proyecto. La negociación entre el cliente y el equipo de desarrollo sobre la prioridad (orden de desarrollo) de cada entregable será de vital importancia para el éxito del proyecto, a diferencia del marco de desarrollo SCRUM, en el que el cliente (propietario del producto) asigna prioridades de desarrollo a los entregables (Sprints) [23-25].

TDDM4IoTS exige que la responsabilidad de *facilitador del proyecto se* asigne al miembro con más experiencia en gestión de proyectos que posea las características de un líder. Los desarrolladores que siguen TDDM4IoTS no están sujetos a imposiciones de tareas, sino a negociaciones. El facilitador no es responsable de todo el proyecto, sólo es el gestor de la negociación entre los equipos de desarrollo. La responsabilidad del proyecto recae en cada uno de sus miembros. En consecuencia, TDDM4IoTS adopta un enfoque de gestión hori- zontal con responsabilidad compartida, dejando que los miembros del equipo se autoorganicen. El número de desarrolladores dependerá del tamaño del proyecto, y teniendo en cuenta que sugerimos equipos "ágiles", no deberían ser más de diez [23, 26]. Los miembros del proyecto tienen distintas responsabilidades, dependiendo de sus funciones, que se muestran en la Tabla 4. Cada equipo de desarrollo está formado por un máximo de tres desarrolladores, y debe estar equilibrado en cuanto a conocimientos y experiencia [26]. Uno de sus miembros será designado (informalmente) como *consejero*, en caso necesario.

En consecuencia, mientras que los participantes pueden desempeñar tres roles principales en SCRUM, que se detallan en [29, 30], junto con sus responsabilidades, nosotros consideramos cuatro roles en TDDM4IoTS, como se muestra en la Tabla 4.

Podemos concluir que (1) SCRUM se basa en los principios para el desarrollo ágil de software. (2) IoT engloba diferentes y variados aspectos, como la complejidad del desarrollo de software, el despliegue de hardware, las comunicaciones indispensables, la nube...

Tabla 4. Funciones y responsabilidades en TDDM4IoTS

Rolea	Descripción	Responsabilidades
Facilitador del proyecto	Experto con amplia experiencia en gestión de proyectos y en el desarrollo ágil de IoTS. Solucionador de conflictos, formador, facilitador y con las características innatas de un líder [27, 28].	(1) Apoyar al equipo de desarrollo en la consecución de sus objetivos. (2) Contribuir con su experiencia al desarrollo de los entregables. (3) Negociar con el cliente aspectos del desarrollo (orden de los entregables, tiempo, recursos,...)[b]
Consejero	Miembro del equipo de desarrollo que se convierte en "líder" (sin una designación formal) debido a su rendimiento.	Instruir a sus compañeros sobre los temas de su competencia.

(*continuación*)

Tabla 4. (*continuación*)

Rolea	Descripción	Responsabilidades
Cliente/Usuario final	Persona con buena comunicación y conocimiento de todas las funcionalidades del IoTS que encarga su desarrollo al equipo de desarrollo	(1) Contribuir a los requisitos del IoTS. (2) Aprobar la funcionalidad de los entregables acabados y el IoTS final.
Equipo de desarrollo	Grupo multidisciplinar de expertos con conocimientos en los diferentes ámbitos del proyecto, y que es responsable del desarrollo del IoTS. Facilitadores de conocimientos y experiencia	(1) Negociar con el cliente los aspectos del desarrollo del IoTS (orden de los entregables, tiempo, recursos,...)[b]. (2) Crear entregables que cumplan plenamente los requisitos del cliente

aEn función del proyecto, puede ser necesario que expertos en otros ámbitos presten sus servicios durante un tiempo determinado.
bEsta responsabilidad es compartida por dos funciones.

almacenamiento y procesamiento, y la intercomunicación entre estos elementos, entre otros. (3) SCRUM y XP ignoran los requisitos no funcionales, que son un aspecto importante en IoTS [31]. Estas tres premisas hacen bastante difícil realizar una adaptación sencilla de SCRUM para el desarrollo de IoTS. Por ello, pretendemos incorporar las mejores características del manifiesto para el desarrollo ágil de software en TDDM4IoTS, adaptándolas a las particularidades de IoTS.

En los siguientes subapartados se describe cómo aplicar los fundamentos en los que se basa TDDM4IoTS y sus fases, así como las actividades a realizar y las posibles herramientas a utilizar en cada una de ellas.

3.1 Fundamentos de TDDM4IoTS

A continuación se describen los principales fundamentos en los que se basa TDDM4IoTS.

Valores y principios del desarrollo ágil de software. El desarrollo ágil de software se rige por cuatro valores, a saber: (i) las personas y las interacciones antes que los procesos y las herramientas, (ii) el software de trabajo antes que la documentación exhaustiva, (iii) la colaboración con el cliente antes que la negociación contractual, y (iv) la respuesta al cambio antes que el seguimiento del plan [29, 30]. Las metodologías ágiles han revolucionado el proceso de desarrollo de software, demostrando que los equipos de desarrollo que han completado sus proyectos con éxito y a tiempo no respetaban metodologías rígidas y pesadas [30]. Para cumplir con estos valores, se detallan doce principios que una metodología debe cumplir para ser calificada como metodología ágil [29, 30].

TDDM4IoTS cumple los doce principios, teniendo en cuenta que, como ya se ha dicho, los IoTS comprenden el despliegue de una serie de dispositivos de hardware además del software correspondiente. Por lo tanto, un entregable será un elemento acabado del sistema, obtenido como resultado de una iteración, que es importante para el cliente.

TDD como metodología ágil. TDD [32-34] es una de las metodologías utilizadas para el desarrollo de SI tradicionales que no ha sido aplicada al desarrollo de IoTS. Dado que este tipo de metodología garantiza que el software del sistema desarrollado satisface los requisitos que el usuario ha proporcionado, la hemos incorporado a nuestra propuesta. En la actualidad, muchos entornos de desarrollo integrados (IDE) soportan TDD, como IntelliJ IDEA, .Net, o Eclipse, por citar algunos [35].

TDD, que se considera una metodología ágil, se aplica para garantizar la calidad de los entregables. Tiene tres fases en su ciclo de desarrollo [33]. Al escribir primero las pruebas y luego el código que tiene que superarlas, se garantiza que el software haga exactamente lo que quiere el cliente (es decir, las pruebas especifican formalmente los casos de uso) [22].

Las metodologías ágiles de desarrollo de software que se han tenido en cuenta en esta investigación son XP (eXtreme Programming) [36] y SCRUM [23]. Nos interesa cómo enfocan los proyectos de desarrollo de software, y especialmente en: frecuencias de entrega, aceptación de cambios, dar mayor importancia al entregable que a una documentación exhaustiva, y la calidad del software (probado y aprobado por el cliente) como una de las características más importantes, entre otros principios de las metodologías ágiles que se deben cumplir. Por un lado, XP es una de las metodologías que ha promovido TDD [35]. Por otro lado, SCRUM entrega software probado, es decir, software que ha superado las pruebas, sin especificar si las pruebas se escriben antes o después del código del software. Sin embargo, separa muy bien las pruebas del desarrollo, ya que otro equipo (de probadores) prueba el software.

MDE. La heterogeneidad de las tecnologías y estándares IoTS hacen del MDE una base importante para TDDM4IoTS. Uno de los objetivos es reutilizar estos modelos (con o sin adaptaciones) en el desarrollo de otros sistemas, para lo cual estos modelos podrían convertirse en código ejecutable [37].

3.2 Fases de TDDM4IoTS

Las fases de TDDM4IoTS están diseñadas para que las herramientas utilizadas por los desarrolladores en cada fase sean de su propia elección. Se recomiendan las herramientas que los desarrolladores pueden utilizar para cumplir los objetivos de cada fase. El primer reto a superar es la comunicación entre los miembros del proyecto y entre los miembros de cada equipo de desarrollo. La mejor manera de lograr una comunicación eficaz es cara a cara, como se indica en el manifiesto ágil [29, 30], mediante reuniones periódicas fijadas por los equipos al inicio del proyecto o al comienzo del desarrollo del entregable correspondiente. Establecer la frecuencia de dichas reuniones es responsabilidad de todos los miembros del proyecto. No obstante, se recomiendan de tres a cinco reuniones semanales. Deben buscarse mecanismos para que los equipos de desarrollo se sientan motivados y a gusto con su trabajo, con la organización, con el resto de miembros del proyecto y con su propio equipo, con el fin de lograr un mejor rendimiento [38].

(1) Análisis preliminar. El objetivo de esta fase es obtener un estudio de viabilidad del sistema (completo) en cuanto a sus aspectos tecnológicos, económicos y operativos, así como un análisis del contexto en el que se desplegará el sistema y la interacción con el usuario final, basándose en los requisitos y objetivos expresados por el cliente. Las actividades a realizar pueden incluir:

- *Análisis de requisitos.* Este análisis determina dos tipos de requisitos: (i) funcionales, que son las especificaciones del cliente (lista de entregables), que determinan su prioridad de implementación, y (ii) no funcionales, también llamados atributos de calidad, como escalabilidad, intrusividad, estética del entorno (despliegue, apariencia, etc.).
- *Análisis tecnológico.* Determina la tecnología que debe utilizarse y que responde a las necesidades del sistema: (i) Recursos de hardware ya disponibles; (ii) Hardware existente, teniendo en cuenta sus características y costes; (iii) Herramientas (software) para la configuración del hardware; (iv) Herramientas tanto para el desarrollo de software como para la gestión del almacenamiento; (v) Hardware de terceros para tareas específicas, y (vi) Desarrollo de hardware propio, si es necesario y factible.
- *Análisis del entorno.* El cliente puede especificar las características del entorno en el que se implantará el sistema. Por ejemplo, puntos de suministro eléctrico disponibles y/o viables, redes de comunicación de datos -acceso a Internet-, métodos de interacción preferidos por el cliente, etc.
- *Análisis de viabilidad.* Entre los tipos de viabilidad que deben analizarse se encuentran: (i) Viabilidad técnica, para la que habría que responder a las siguientes preguntas: (a) ¿Existen las tecnologías necesarias para desarrollar el proyecto? (b) ¿Se dispone de personal formado para desarrollar el sistema? (c) ¿Puede desarrollarse el sistema? (ii) Viabilidad económica, donde la pregunta a responder sería: ¿Existe un presupuesto adecuado para desarrollar el proyecto? (iii) Viabilidad operativa, que es importante para que el sistema siga funcionando una vez implantado, por lo que hay que responder a estas preguntas: (a) ¿Será posible instalar el sistema una vez finalizado? (b) ¿Podrá funcionar el sistema con los recursos disponibles? (c) ¿Existen las garantías necesarias para que el sistema siga funcionando una vez instalado? (d) ¿Tendrá el IoTS un mantenimiento debidamente programado?

Dado que IoTS puede implicar el uso de múltiples tecnologías, la disponibilidad de cada una de ellas podría alterar el orden o las prioridades de los entregables. Por ello, es necesario que esta primera fase se realice de forma global al inicio del proyecto, y se revise al comienzo del desarrollo de cada entregable, para considerar los cambios relacionados con la tecnología (nuevos dispositivos, nuevas herramientas, etc.) y los requisitos que puedan surgir durante el desarrollo del proyecto.

No existen herramientas específicas para cumplir el objetivo de esta fase en particular. Sin embargo, para la planificación del proyecto, se sugiere utilizar una herramienta de software libre, como OpenProj, GanttProject, dotProject, y para aquellos que prefieren el software propietario, MS-Project, entre muchos otros.

El resultado de esta fase es fundamental para las primeras negociaciones con el cliente, relativas a entregas, plazos y presupuesto. Todo esto es preliminar y no puede considerarse definitivo, ya que puede negociarse entre ambas partes al inicio del desarrollo de cada entregable.

(2) Diseño de la capa tecnológica. El objetivo de esta fase es obtener el primer diseño del sistema global que servirá de guía para los equipos de desarrollo. Esto es muy importante, dado que IoT implica tecnologías emergentes y heterogéneas, y hasta ahora es difícil encontrar un profesional que domine todas las tecnologías implicadas en el desarrollo de IoTS [21].

Para el diseño del sistema, se pueden utilizar herramientas de diseño de circuitos que puedan representar lo más claramente posible los elementos que se han determinado para el proyecto. Por ejemplo, si se utilizan placas Arduino, pueden utilizarse herramientas en línea como Circuito.io o Fritzing. El equipo de desarrollo puede complementar los diseños obtenidos. Si es necesario, el diseño resultante puede actualizarse al final de cada entregable. Este será uno de los documentos de mayor interés para todos los equipos de desarrollo del proyecto. Por lo tanto, debe estar siempre accesible para todos ellos.

En esta fase debe diseñarse la arquitectura con la que se implantará el sistema. Por consiguiente, el resultado de esta fase servirá de guía para todo el proceso de desarrollo.

(3) Análisis detallado de requisitos. El objetivo de esta fase, que se ejecutará para cada entrega del sistema, es obtener los requisitos detallados de la entrega que se va a desarrollar. Además, se pedirá al cliente que los describa junto con los desarrolladores, para reducir las ambigüedades. Para la especificación de requisitos, se recomienda utilizar herramientas que sean comprensibles para todas las partes interesadas, considerando al cliente como una de ellas. Una de las notaciones a utilizar podría ser UML, con sus diferentes herramientas, como casos de uso y casos de uso semiestructurados, utilizando plantillas preestablecidas, buscando eliminar ambigüedades, además de otras herramientas, como diagramas de estado y despliegue, que ayudan a comprender los requisitos proporcionados por el cliente.

(4) Generación y adaptación de modelos.
El objetivo de aplicar el MDE en TDDM4IoTS es reutilizar modelos y mejorar así la productividad del equipo de desarrollo. Por lo tanto, en esta fase se generarán nuevos modelos o se adaptarán los ya existentes. El uso de modelos abstrae o al menos minimiza los aspectos de heterogeneidad de las tecnologías, lo que permite una buena comunicación entre los equipos de desarrollo y los clientes. Además, dependiendo de las herramientas de software utilizadas para el modelado, el software puede generarse automáticamente a través de modelos, desde modelos abstractos hasta modelos específicos [10, 18]. Uno de los modelos es el diagrama de clases, que se utiliza para generar la base de datos.

Los lenguajes de modelado sugeridos son: UML, BPMN [12], la adaptación de alguno de ellos [14], su combinación o la creación de uno nuevo, buscando siempre cubrir las características particulares del IoTS a desarrollar y que sea de fácil comprensión para los involucrados en el proyecto. Entre las herramientas automatizadas para el modelado de sistemas se encuentran StarUML, ArgoUML, MagicDraw y Visual Studio .Net, por ejemplo. Quienes opten por BPMN, pueden elegir herramientas como Lucidchart o VisualParadigm, entre muchas otras.

(5) Generación de pruebas. TDDM4IoTS sigue el paradigma TDD, por lo que debe generar las pruebas que el software debe superar para asegurar la calidad del sistema. Las pruebas se pueden agrupar en dos grupos: (1) Las pruebas escritas por los desarrolladores, dentro de las cuales están:
(a) pruebas unitarias, que son las más exhaustivas, para examinar el funcionamiento completo de una función, es decir, se comprueba si la función produce los resultados que debe producir e incluso si admite las excepciones que puedan surgir; y (b) pruebas de integración, que también implican pruebas del sistema. Y (2) pruebas documentadas por el cliente, que son básicamente pruebas de aceptación, incluidas las

pruebas funcionales [22].

Las herramientas automatizadas para esta fase dependerán del IDE que se haya seleccionado para el desarrollo, aunque hasta ahora ninguna herramienta genera automáticamente pruebas basadas en los requisitos.

(6) Generación de software. Esta fase se basa en modelos y pruebas. El desarrollador debe escribir/generar el código para que se superen las pruebas [33, 39]. Los nuevos modelos generados y/o los modelos existentes adaptados forman parte de la documentación del sistema. Por lo tanto, a diferencia de las metodologías SCRUM y XP, que sugieren que la documentación relacionada con el análisis y el diseño de la solución se escriba al final del desarrollo de cada entregable [30], TDDM4IoTS propone hacerlo antes de la generación de código, ya que existen herramientas que, basadas en los modelos, ayudan en esta tarea. Asimismo, al encuestar a desarrolladores de software que trabajan o han trabajado con metodología SCRUM y/o XP, la mayoría coinciden en que la solución que se implementará posteriormente debería, al menos, esbozarse al inicio del ciclo de vida de la metodología. Esto corrobora lo expuesto en TDDM4IoTS respecto al análisis y diseño de la solución, que debería realizarse durante el desarrollo de los entregables.

Una vez generado el programa informático, se comprueba que supera los correspondientes pruebas, y entonces termina esta fase. El resultado de esta fase es el software probado y funcionando, aunque casi siempre habrá que perfeccionarlo más adelante.

Para generar el código, se pueden utilizar algunas de las herramientas mencionadas en la fase de generación y refinamiento del modelo. De hecho, existen varias herramientas, tanto de software libre como propietario, que permiten incluso simular el comportamiento del sistema.

(7) Perfeccionamiento de modelos. Los modelos UML facilitan el refinamiento del modelo. Hay que tener en cuenta que la solución encontrada es una solución funcional, pero no necesariamente una solución óptima. Por ello, es importante realizar los ajustes y refinamientos necesarios, como mejorar la robustez, escalabilidad o reusabilidad de los entregables que componen el IoTS a desarrollar [40]. El resultado de esta fase será el modelo definitivo, a partir del cual se generará el código del sistema.

Las herramientas recomendadas para realizar las tareas de esta fase son las mismas que en la fase de generación y adaptación del modelo.

(8) Perfeccionamiento del software. El trabajo de los desarrolladores en esta fase consistirá en garantizar la calidad del software, eliminando redundancias y facilitando su mantenimiento. Las herramientas que darán soporte a esta actividad serán las proporcionadas por el IDE elegido. Se debe garantizar que el software final cumple las especificaciones de un código limpio [34].

(9) Despliegue de hardware y software. Una vez que el software ha sido probado (simulado), se implementa y despliega en los dispositivos y recursos que se utilizarán en el sistema, confirmando el cumplimiento de los requisitos finales negociados entre el cliente y el equipo de desarrollo. En este punto, y para el primer entregable, ya se habrán configurado tanto el sistema de almacenamiento de información como las aplicaciones que servirán para la interacción usuario-sistema [41, 42]. Para los siguientes entregables, se realizarán los cambios necesarios en esta infraestructura ensamblada para añadir los nuevos entregables. Dado que los entregables posteriores dependen de la tecnología ya instalada, es necesario garantizar el funcionamiento del (nuevo) sistema integrado antes de continuar con el proceso de desarrollo.

Las herramientas que se utilicen en esta fase dependerán de la tecnología utilizada (placa

única
ordenadores, sensores integrados...).

(10) Evaluación del entregable. Una vez finalizado el desarrollo del entregable (ha tenido que superar las pruebas necesarias para garantizar su funcionamiento), en esta fase se deben volver a realizar las pruebas de integración, las pruebas del sistema y, por supuesto, las pruebas funcionales [43, 44].

(11) Mantenimiento. Los IoTS combinan la complejidad del mantenimiento del software con la menor complejidad del mantenimiento del hardware. Si un componente físico de IoTS (por ejemplo, un sensor, un actuador, un ordenador de a bordo, entre otros) falla, se sustituye por otro dispositivo similar. Sin embargo, si cambian los requisitos del software, hay que modificar el código, dado que no tiene piezas de repuesto [45]. Además, los IoTS necesitan un mantenimiento operativo constante (baterías o líneas eléctricas, y conectividad, entre otros).

4 Conclusiones y trabajo futuro

A partir de una revisión exhaustiva de las metodologías utilizadas para el desarrollo de IoTS, se ha constatado que no existe una metodología estándar para este dominio de aplicación. Este hecho ha llevado a los desarrolladores de IoTS a utilizar metodologías diseñadas para desarrollar otros tipos de SI más tradicionales y a realizar ajustes ad-hoc para satisfacer las necesidades particulares de cada proyecto. Además, el uso de estas metodologías no específicas para el desarrollo de IoTS ha pasado por alto la necesidad de considerar específicamente aspectos importantes de estos sistemas, como sus requisitos especiales y las características particulares del hardware que se desplegará en ellos.

En consecuencia, se ha propuesto una nueva metodología denominada TDDM4IoTS para abordar de forma específi- ca el desarrollo de IoTS, que incluye los fundamentos más importantes de TDD, MDE y desarrollo ágil, y que trata de solventar cada uno de los aspectos que se han detectado como debilidades en las metodologías revisadas que se han utilizado para el mismo fin.

Uno de nuestros próximos proyectos será validar TDDM4IoTS para determinar su eficacia y aceptación. Además, se desarrollará una herramienta para apoyar la generación automatizada de los entregables correspondientes a cada fase de TDDM4IoTS. En particular, nos centraremos en la generación de pruebas.

Referencias

1. Dahl, O.-J., Hoare, C.A.R.: Capítulo III: Estructuras jerárquicas de programas. En: Structured Programming, pp. 175-220. Academic Press Ltd. (1972)
2. Ingalls, D.H.H.: Diseño e implementación del sistema de programación Smalltalk-76. En: Pro- ceedings of the 5th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages - POPL 1978, pp. 9-16. ACM (1978)
3. Pashtan, A.: Sistemas operativos orientados a objetos: una metodología de diseño emergente. En: Proceedings of the ACM 1982 Conference on ACM 1982, pp. 126-131. ACM (1982)
4. Schwabe, D., Rossi, G.: El modelo de diseño hipermedia orientado a objetos. Commun. ACM **38**(8), 45-46 (2002)
5. Ashton, K.: Eso del "Internet de los objetos". 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., y otros: 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 frame- work for code offloading in the Internet of Things. Future Gener. Comput. Syst. **92**(marzo), 424-437 (2019)
9. Cervantes-Solis, J.W., Baber, C., Khattab, A., Mitch, R.: Rule and theme discovery in human interacciones con una Internet de los objetos. En: Proceedings of the 2015 British HCI Conference on - British HCI 2015, pp. 222-227. ACM, REINO UNIDO (2015)
10. Ciccozzi, F., Spalazzese, R.: MDE4IoT: supporting the Internet of Things with model-driven ingeniería. En: 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., y otros: Heterogeneous applications, tools, and methodologies in the car industria manufacturera a través de un enfoque IoT. 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.: Diseño basado en modelos. de sistemas IoT con el marco de componentes BIP. J. Softw: Pract. Exp. **48**(6), 1167-1194 (2018)
14. Brambilla, M., Umuhoza, E., Acerbis, R.: Model-driven development of user interfaces for Sistemas IoT mediante componentes y patrones específicos de dominio. J. Internet Serv. Appl. **8**(1), 1-21 (2017)
15. Harbouche, A., Djedi, N., Erradi, M., Ben-Othman, J., Kobbane, A.: Model driven flexible diseño de una red inalámbrica de sensores corporales para la vigilancia de la salud. Comput. Netw. **129-2**, 548- 571 (2017)
16. Usländer, T., Batz, T.: Agile service engineering in the industrial Internet of Things. Futuro Internet **10**(10), 100 (2018)
17. Pico-Valencia, P., Holgado-Terriza, J.A., Paderewski, P.: Un método sistemático para la construcción de aplicaciones de internet de agentes basado en el enfoque de linked open data. Future Gener. Comput. Syst. **94**, 250-271 (2019)
18. Sosa-Reyna, C.M., Tello-Leal, E., Lara-Alabazares, D.: Methodology for the model-driven desarrollo de aplicaciones IoT orientadas a servicios. J. Syst. Architect. **90**, 15-22 (2018)
19. de Farias, C.M., y otros: COMFIT: un entorno de desarrollo para el Internet de las Cosas. Future Gener. Comput. Syst. **75**, 128-144 (2017)
20. Fortino, G., y otros: Towards multi-layer interoperability of heterogeneous IoT platforms: the Enfoque INTER-IoT. En: 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.: Habilitación del desarrollo de aplicaciones de alto nivel para la Internet de los objetos.
J. Syst. Softw. **103**, 62-84 (2015)
22. Martin, R.C.: Clean Coder Blog, TDD (2017). https://blog.cleancoder.com/. Consultado el 11 de septiembre de 2019
23. Rising, L., Janoff, N.S.: Proceso de desarrollo de software Scrum para equipos pequeños. 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: Revisión y análisis. 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→XU). En: 29th Annual International Computer Software and Applications Conference (COMPSAC 2005), vol. 2, pp. 169-172. IEEE, REINO UNIDO (2005)
27. Mowday, R.T.: Leader characteristics, self-confidence, and methods of upward influence in situaciones de decisión organizativa. 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.: Manifiesto para el desarrollo ágil de software (2001). http://agilemanifesto.org/. Consultado el 11 de mayo de 2019
30. Hazzan, O., Dubinsky, Y.: El manifiesto ágil. En: 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. En: 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.: Desarrollo basado en pruebas: conceptos, taxonomía y dirección futura. Computer **38**(9), 43-50 (2005)
34. Martin, R.C.: Clean Code, A Handbook of Agile Software Craftsmanship, 1ª 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. En: Kosiuczenko, P., Madeyski, L. (eds.) Towards a Synergistic Combination of Research and Practice in Soft- ware 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 expandida: programación extrema distribuida. En: 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.: Un enfoque basado en el desarrollo dirigido por modelos para aplicaciones IoT. En: 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.: Factores que afectan al rendimiento de los desarrolladores de software: un enfoque integrado. MIS Q. **16**(3), 395-413 (1992)
39. Nyznar, M., Pałka, D.: Generación de plantillas de código fuente a partir de pruebas unitarias. En: Grzech, A., Świa˛tek, J., Wilimowska, Z., Borzemski, L. (eds.) Information Systems Architecture and Technology: Proceedings of 37th International Conference on Information Systems Archi-
tectura y Tecnología - ISAT 2016 - Parte 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. En: 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), pp. 17-18. IEEE, Zúrich (2018)
42. Guerrero-Ulloa, G., Rodríguez-Domínguez, C., Hornos, M.J.: IoT-based system to help care for dependent elderly. En: 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.: Ingeniería de software: A Practitioner's Approach, 8ª edn. McGraw-Hill Education, Boston (2015)