Capítulo I
Análisis de las Metodologías Ágiles para el Desarrollo de Software

Carmen Janeth Parada - janethpc@ufps.edu.co
Docente investigadora de GIDIS, Universidad Francisco de Paula Santander, Cúcuta
Pilar Rojas Puentes - pilarrojas@ufps.edu.co
Docente investigador de GIDIS, Universidad Francisco de Paula Santander, Cúcuta
Judith del Pilar Rodríguez - judithdelpilarrt@ufps.edu.co
Docente investigadora de GIDIS, Universidad Francisco de Paula Santander, Cúcuta

I. INTRODUCCIÓN

Las metodologías ágiles han tomado gran importancia e interés en los equipos de trabajo de la industria del software, por las ventajas y bondades que se obtienen con su uso en el desarrollo rápido, la flexibilidad a los cambios a que están expuestos este tipo de proyectos, la entrega de productos a satisfacción del cliente y la adaptabilidad de estas metodologías.

El artículo describe las características que tienen las metodologías ágiles, para facilitar la estructuración de una nueva metodología que permita a los equipos de trabajo personalizarlas, de acuerdo con los proyectos que desarrollan. Se responde a la pregunta: ¿Cómo especificar una nueva metodología ágil a partir de las características de las metodologías ágiles existentes?

La estructura del presente artículo se organizó de la siguiente manera: en el apartado 2 se presentan los conceptos más relevantes en cuanto a metodologías ágiles en los beneficios, y el contexto de aplicación e impacto; en el apartado 3, la metodología de desarrollo de la investigación; en el apartado 4, el análisis de las características de las metodologías ágiles; en el apartado final, la conclusión y las referencias usadas en la investigación.

II. REVISIÓN DE LITERATURA

A. Metodologías Ágiles para el Desarrollo de Software

El Instituto Nacional de Tecnologías de la Comunicación de España (Inteco) [1], en su capítulo de Ingeniería de Software, define "una metodología como un conjunto integrado de técnicas y métodos que permite abordar de forma homogénea y abierta cada una de las actividades del ciclo de vida de un proyecto de desarrollo". Igualmente, en [2] se define que, en el ámbito de los proyectos, "una metodología se considera un proceso ordenado compuesto de pasos e involucra herramientas, técnicas, responsabilidades y entregables, el cual pasa por un estado de avance, concluye y se entrega con éxito". Lo anterior permite indicar que una metodología establece un camino para desarrollar un proyecto de manera sistemática, y proporciona un estándar de trabajo al equipo de desarrollo para alcanzar los objetivos propuestos y satisfacer los requisitos de los interesados.

Las metodologías de desarrollo presentan dos enfoques: predictivos y adaptativos [3]; los enfoques predictivos son utilizados cuando se tienen identificados los requisitos con anterioridad, antes de iniciar el proceso de desarrollo y cuando el producto software final no está sujeto a cambios. En cambio, en los enfoques adaptativos, el alcance del producto se define y se aprueba antes del comienzo de una iteración.

Bajo esta definición nacieron las metodologías ágiles, las cuales son adaptativas más que predictivas, y se centran más en las personas que en los procesos [4]. Con base en estas características, a inicios del año 2001, un grupo de desarrolladores y de expertos de la industria de software consolidaron el término ágil aplicado al desarrollo de software y definieron el "Manifiesto Ágil" [5], donde plasmaron los principios y valores que los grupos desarrolladores deben aplicar para crear softwares rápidamente y respondiendo a los cambios que puedan surgir a lo largo de un proyecto. Estas metodologías se caracterizan principalmente por representar la antítesis de lo que es el tradicional proceso para desarrollar softwares; pasan por alto la utilización de elaborados casos de uso, la exhaustiva definición de requerimientos y la producción de una extensa documentación [6].

Para los proyectos pequeños y medianos, donde los requerimientos cambian constantemente y donde no existe la necesidad de tener equipos grandes, los usos de metodologías ágiles permiten obtener productos softwares en tiempos más cortos y con costos reducidos sin perder la calidad del producto desarrollado [7].

B. Beneficios del Uso de las Metodologías ágiles

Usar metodologías ágiles para el desarrollo de softwares no solo permite disminuir los tiempos de desarrollo, sino que se pueden dar otros beneficios como simplificación de la sobrecarga de procesos, calidad mejorada, mejora en la previsibilidad con una adecuada gestión del riesgo, mejor perfil de productividad, entre otros.

Simplificación de la sobrecarga de procesos. Es normal que los equipos de trabajo que desarrollan softwares usando metodologías tradicionales tengan sobrecarga de trabajo para poder cumplir con el desarrollo del software, de acuerdo con los estándares de la industria. Las metodologías ágiles permiten cumplir los estándares definidos por la industria con menos sobrecarga utilizando iteraciones más cortas y empaquetadas, lo que genera beneficios en un proceso que puede adaptarse a los cambios que surgen, y donde se requiere menos sobrecarga en el proceso desde su inicio hasta su finalización, lo que implica menos trabajo a medida que se acerca la fecha final.

Calidad mejorada. Las prácticas de desarrollo ágil proporcionan la mínima funcionalidad con la máxima calidad. La mínima funcionalidad no implica necesariamente una pobre funcionalidad, sino la suficiente como para conseguir que el trabajo se realice. Desarrollar iteraciones en poco tiempo y demostrar a los clientes los productos pronto y con frecuencia, permite tanto a los clientes como a los equipos de desarrollo ponerse de acuerdo en que el producto cumple con cada una de sus necesidades.

Mejorar la previsibilidad a través de una adecuada gestión del riesgo. Hay muchas razones justificadas por las cuales los equipos desarrolladores no cumplen con las entregas de un producto en las fechas determinadas; sea cual sea la causa que genera este incumplimiento, los negocios requieren que estas fechas de entrega se cumplan. Las metodologías ágiles ayudan a que los proyectos de tecnología cumplan con las fechas de entrega estipuladas en los contratos; esto se puede hacer de dos formas: 1) Priorizando los riesgos que se puedan causar en el desarrollo de un software y, 2) Evaluando el riesgo en paralelo, esto es, varios equipos trabajan de forma paralela resolviendo el mismo problema con diferentes soluciones, lo cual puede permitir tomar decisiones claves para seguir el desarrollo.

Mejor perfil de productividad. Los equipos ágiles tienden a ser muy productivos desde la primera iteración hasta el lanzamiento y ritmo, se tienen que gestionar de modo que no se produzca agotamiento. Los equipos ágiles que mantienen este código de trabajo con cada iteración, permiten realizar pruebas de rendimiento y sistemas desde el principio, es decir, en las primeras iteraciones. De este modo, defectos críticos, como problemas de integración, se descubren antes, la calidad general del producto es mayor y el equipo funciona de manera más productiva durante todo el ciclo de desarrollo.

C. Aplicación del Agilismo en un Contexto Académico

El estudio sobre las experiencias de formación en metodologías ágiles, realizado en los años 2015-2016 con los estudiantes del curso de Análisis y Diseño de Sistemas e Ingeniería del Software del Programa de Ingeniería de Sistemas (UFPS) [8], se apoyó en una investigación de campo; uno de los aspectos tenidos en cuenta tuvo que ver con la validez para garantizar la calidad del estudio. En la ejecución del procedimiento metodológico cuantitativo se aplicó la prueba piloto, cuestionario de satisfacción ya validada a estudiantes que no formaban parte de la muestra, pero que presentaban las mismas características de los sujetos evaluados. Para hallar el coeficiente de confiabilidad se procedió de la siguiente manera:

- Aplicación de la prueba piloto a un grupo de 30 sujetos (estudiantes) pertenecientes a la muestra de estudio, con características equivalentes a la misma.
- Codificación de las respuestas; transcripción de las respuestas en una matriz de tabulación de doble entrada con el apoyo del equipo estadístico de la Universidad.
- Interpretación de los valores tomando en cuenta la escala de Likertt.
- Determinación de resultados con tabulación simple y tabulación cruzada.

Conocido el método de investigación, se procedió a desarrollar el estudio, y al iniciar cada semestre académico, en el aula de clase se impartieron los conceptos y talleres para resolver un problema real, y se organizaron los trabajos por equipos conformados por cuatro o cinco estudiantes.

Durante el desarrollo del proyecto se organizaron sesiones semanales para el trabajo en equipo, la documentación y el desarrollo. En estos proyectos se evidenció que los estudiantes ven de manera positiva y favorable no sólo la estrategia implementada, sino el hecho de poder aplicar una metodología de desarrollo ágil en un caso práctico. Para los estudiantes fue un aspecto positivo, y se destaca que el trabajo realizado les permitió aplicar las mejores prácticas del desarrollo de software, así como hacer el seguimiento al proyecto (donde el cliente y el profesor guían el trabajo).

Lo anterior permitió conocer no sólo apreciaciones de los estudiantes, sino la aplicación de las mejores prácticas de la ingeniería de software y el uso de metodologías ágiles en proyectos reales; era importante resolver el problema que se venía presentando en cuanto a unificar los conceptos teóricos con la práctica, aspecto que se había evidenciado en semestres anteriores.

Las metodologías XP y SCRUM fueron las más seleccionadas (75%); para los estudiantes era atractivo mejorar y agilizar el proceso de desarrollo del software. Sin embargo, consideraban que se presentaban conflictos entre los miembros del equipo, bien por el ritmo de trabajo o por capacidades, al responder con el trabajo efectuado. Igualmente, la exigencia era mayor si no conocían de antemano el proceso de desarrollo. Asimismo, durante el proceso de desarrollo los estudiantes presentaron dificultades en cuanto a las pruebas, el *refactoring* e incluso al integrar las pruebas, donde la experiencia y la disciplina fueron aspectos relevantes.

Otro aspecto positivo del estudio fue el uso de las metodologías ágiles. En el curso de Ingeniería de Software se evidenció que, en el mantenimiento correctivo de los proyectos realizados en el contexto planteado, no se presentaban los requerimientos esperados por los clientes, y además los hitos se retrasaban, lo que ocasionaba mayor presión al equipo de desarrollo, al exigírsele mejorar los tiempos y mantener los hitos. Por otro lado, se observó que el cliente no estaba satisfecho con la entrega de las mejoras y/o correcciones, ya que se entregaban en fechas posteriores a las pactadas.

Para solucionar y mejorar los aspectos anteriormente descritos, se determinó implementar la metodología SCRUM en los proyectos de curso de las asignaturas Análisis y Diseño e Ingeniería de Software, esto permitió mejorar la satisfacción del cliente al informarle sobre los avances realizados

y evitar así malentendidos; mantener motivados a los equipos de desarrollo por los resultados obtenidos; conocer los avances del proyecto cuando los interesados lo solicitaran; ahorrar tiempo y costos cuando se presentaran cambios en los requisitos del producto, y cumplir con los tiempos establecidos para la entrega.

III. METODOLOGÍA

Para responder a las preguntas de investigación, se condujo un estudio descriptivo exploratorio, el cual permitió conocer las percepciones y las prácticas de los profesionales que desarrollan software, no se limitó sólo a recolectar los datos, sino a identificar las relaciones entre dos o más variables. Posteriormente, se recogieron los datos sobre la base de la hipótesis y se resumió de manera cuidadosa la información para ser analizada con el fin de entregar resultados que contribuyeran al estudio.

Luego se aplicó el instrumento de la encuesta, que fue diligenciada por los desarrolladores pertenecientes a la población objetivo. Las respuestas complementaron los hallazgos del estudio. La encuesta contenía preguntas relacionadas con las prácticas ágiles de las diferentes metodologías existentes, su utilización, sus características y sus ventajas en el desarrollo del software. Luego se procedió a realizar el análisis de los datos, con la técnica de comparación de conceptos, y se intentó crear una metodología alrededor de las lecciones aprendidas por los investigadores y de los resultados del proceso de desarrollo con el fin de adoptar una metodología ágil, y de esta manera responder a las preguntas de investigación formuladas. Por último, se consolidaron las conclusiones encontradas en el estudio.

Recolección de datos: En el artículo se señalan los datos obtenidos y la naturaleza de la población de la cual fueron extraídos. La población fue de 1.234 sujetos con una muestra de 104 sujetos (profesionales egresados del programa de Ingeniería de Sistemas de la UFPS) [9]. Una vez identificada la población, se decidió si se recogían los datos de la población total o de una muestra representativa de ella. El método se eligió en función de la naturaleza del problema y de la finalidad para utilizar los datos.

Investigación descriptiva: El instrumento tipo encuesta se diseñó en dos secciones, y permitió encontrar soluciones que surgieron al aplicarlo. Se efectuó una descripción minuciosa de la temática por tratar, a fin de justificar las percepciones de los desarrolladores de software y las practicas que efectúan. Su objetivo no se limitó solo a determinar el problema, sino también a comparar entre las diferentes metodologías ágiles, su uso y aplicación en el proceso de desarrollo. El estudio permitió determinar un alcance de acuerdo con la población objetivo y la información recogida, se realizó el número de secciones del instrumento en relación con el objeto de estudio y el problema dado.

La muestra calculada es probabilística simple, en ella todos los individuos tienen la misma probabilidad de ser elegidos para conformarla y, consiguientemente, las posibles muestras de tamaño no tienen la misma probabilidad de ser seleccionadas. Este método de muestreo probabilístico nos asegura la representatividad de la muestra extraída [10].

IV. RESULTADOS

Una de las herramientas más importantes y puesta en práctica en la mayoría de las organizaciones exitosas del mundo de hoy es la incorporación de metodologías para la gestión de los proyectos. En [2] se indica que la aparición de metodologías ágiles ha sido constante desde cuando se planteó el concepto. Las metodologías ágiles se emplean para la gestión de proyectos de cualquier disciplina, y han sido apropiadas por la industria del software. Se mencionan las metodologías ágiles más utilizadas por los equipos de trabajo en el desarrollo de software [10].

Como se observa en la Tabla l, las metodologías ágiles más utilizadas por los equipos de trabajo son Scrum (43,42 %), XP (10,50 %) y Crystal (9,20 %), las cuales se toman en cuenta en los resultados del presente artículo.

Tabla 1. Metodologías ágiles

Metodología	Porcentaje resultado del estudio
Adaptive Software Development (ASD)	5,3
Scrum	43,4
Cristal Methods (CM)	9,2
DSMD	3,9
eXtreme Programming (XP)	10,5
Feature-Driven Development (FDD)	1,3
Lean Development (LD)	3,9
Otras	2,6

Fuente: Elaborado a partir de [9], [10]

Las metodologías ágiles son un marco de trabajo de la ingeniería de software que define iteraciones en el desarrollo a lo largo del ciclo de vida del proyecto. La mayoría de las metodologías ágiles pretenden minimizar los riesgos de desarrollo del producto en corto tiempo. En la Tabla 2 se especifican las variables analizadas en el estudio realizado en [10].

Tabla 2. Categoría y variables identificadas en las metodologías ágiles

Categoría	Variable (características)
Generales	Ciclo de vida
	Entregas parciales
Equipo de trabajo	Estilo de comunicación
	Tamaño
	Jerarquía del equipo
Gestión del proyecto	Cultura de gestión
	Planificación
	Retroalimentación
El cliente	Disponibilidad del cliente
	Acceso a usuarios expertos
Procesos y herramientas	Tiempo de la iteración
	Adaptabilidad

Fuente: Tomado de [10].

En la Tabla 3 se describen las características evaluadas en las metodologías Scrum, XP y Cristal.

Tabla 3. Evaluación de características en XP, Scrum y Crystal

Características	XP	SCRUM	CRISTAL
Ciclo de vida	Iterativo e incremental	La metodología seleccionada para el desarrollo del producto es la que determina el flujo de trabajo en el *sprint*	Iterativo e incremental
Entregas parciales	Sujetas a cambios durante el proyecto	No están sujetas a cambios	Realiza entregas frecuentes:
Comunicación equipo	Comunicación fluida entre los miembros del equipo	Comunicación fluida entre los miembros del equipo	Osmótica
Tamaño equipo	Entre 3 - 10	Max 8	Depende de la complejidad y tamaño del proyecto.
Jerarquía del equipo	No existe	Pequeña jerarquía definida.	Estructurada
Cultura de gestión	No existe	Existe	Existe
Planeación	No	Sí	Sí
Retroalimentación	Sí	Sí	Sí
Disponibilidad del cliente	Permanente	Permanente	Periódico

Características	XP	SCRUM	CRISTAL
Acceso a usuarios expertos	Comunicación continua	Integración permanente del experto	Reuniones con periodicidad mayor
Adaptabilidad	Previsión en el diseño del software	A los cambios entre iteraciones	A los cambios entre iteraciones
Tiempo de la Iteración	De 1 a 3 semanas	De 1 a 4 semanas, conocidas como *Sprint*	Depende cómo se definan las entregas

Fuente: Elaboración propia.

A continuación, se realiza el análisis de las características según las categorías definidas en la Tabla 2.

Generales. Las tres metodologías tienen un ciclo de vida interactivo e incremental, y se diferencian particularmente en el número de etapas que cada una de ellas tiene definidas:

a) XP: Desarrolla el producto basado en seis (6) etapas: Exploración, Planificación de la Entrega, Iteraciones, Producción, Mantenimiento y Muerte del Proyecto. Esta metodología se caracteriza por hacer entregas parciales al finalizar la iteración, la cual puede estar sujeta a cambios en las siguientes iteraciones.

b) Scrum: Su ciclo de vida contempla cuatro (4) etapas: Planificación del *sprint,* Etapa de desarrollo, Revisión del *sprint* y Retroalimentación. A diferencia de XP, Scrum se caracteriza por realizar entregables en los *sprints,* los cuales son aprobados por el cliente y no están sujetos a cambios en los siguientes *sprints*.

c) Crystal: El número de etapas del ciclo de vida depende del proyecto; emplea una planeación adaptativa que se fundamenta en el proceso de construir el producto teniendo en cuenta dos factores importantes: la complejidad y el tamaño del producto. Se caracteriza por que al finalizar cada iteración entrega un producto que puede ser funcional para el cliente o un insumo para la siguiente iteración.

Equipo de trabajo. Para las tres metodologías los equipos de trabajo son importantes. El estilo de comunicación de XP, Scrum y Crystal es fluido, mantiene una comunicación constante, rápida y oportuna entre los miembros del equipo de trabajo. A diferencia de XP y Scrum, Crystal tiene un alto costo de *feedback,* debido a la reiteración de la ejecución de los procesos de acuerdo con la evaluación de los entregables. El tamaño de los equipos de trabajo en XP oscila entre 3 a 10 miembros, en Scrum entre 3 y 8, y en Crystal [11] depende de la complejidad y el tamaño del proyecto: de 1 a 6, 7 a 20, 21 a 40, 41 a 80 y de 81 a 200 personas. En cuanto a la jerarquía del equipo de trabajo, en XP no se define una estructura porque todos los miembros del equipo están en un solo nivel para ejercer la toma de decisiones, lo contrario de Scrum y Crystal; Scrum (referencia) tiene una jerarquía definida en: *stakeholders*, *Product owner,* Scrum máster, equipo autoorganizado, y en Crystal la estructura del equipo depende de la cantidad de miembros que lo componen.

Gestión del proyecto. De las tres metodologías analizadas, en XP no se evidencia cultura de gestión, por el contrario, Scrum y Crystal presentan un grupo de procesos como planificación, ejecución, seguimiento y control que permiten la administración del proyecto, y Crystal tiene además un proceso para hacer cierre. Aunque XP no tiene cultura de gestión, algunos equipos de trabajo planean actividades superficialmente, lo contrario ocurre con Scrum y Crystal, que desarrollan una planeación exhaustiva. Las tres metodologías hacen retroalimentación, la cual es costosa en Crystal por el hecho de hacer el proceso reiteradamente.

El cliente. Para XP y Scrum, la disponibilidad del cliente en el desarrollo del producto debe ser permanente, mientras que para Crystal el contacto con el cliente se puede hacer periódicamente dependiendo de la magnitud del proyecto en tiempo y complejidad. Igualmente, en XP y Scrum la comunicación del equipo de trabajo con los expertos requeridos para el desarrollo del producto debe ser continua, mientras que en Crystal no se requiere al experto de forma permanente sino en periodos definidos.

Procesos y herramientas. El tiempo de la iteración en XP oscila entre 1 a 3 semanas; Scrum determina que las iteraciones son de máximo 4 semanas; mientras que en Crystal el tiempo de la iteración depende de cómo se hayan definido las entregas parciales de los entregables. Respecto a la adaptabilidad de los procesos para ejecutar los cambios, XP permite cambios a partir del diseño; Scrum y Crystal permiten cambios entre las iteraciones.

V. CONCLUSIONES

En un proyecto de desarrollo de software no existe una metodología universal, la elección de ésta depende de la complejidad y el tamaño del proyecto. La elección de metodologías ágiles como XP y Scrum no depende de la complejidad y el tamaño del proyecto, mientras que para Crystal estas dos características son fundamentales.

Al hacer las comparaciones de las metodologías descritas en el artículo en cuanto a las etapas que cubren el ciclo de vida, las tres metodologías seleccionadas basan su desarrollo en un modelo iterativo/incremental, siendo Scrum la que utiliza el concepto de *sprint*.

Las iteraciones definidas en las tres metodologías tienden a desarrollarse en periodos fijos de tiempo; los cambios constantes son permitidos sólo en XP. En ellas, el cliente y el experto tienen constante comunicación con el equipo de trabajo, excepto en Crystal que mantiene una periodicidad mayor en la comunicación.

El tamaño del equipo de trabajo tiene una relación directa con el tamaño del proyecto.

Finalmente, los resultados obtenidos del análisis de las características de las metodologías ágiles, permiten a los líderes de los equipos de desarrollo estructurar una metodología que se ajuste a las necesidades del proyecto, teniendo en cuenta los activos del negocio y los factores ambientales.

REFERENCIAS

[1] Instituto Nacional de Tecnologías de la Comunicación (Inteco), *Ingeniería del Software: metodologías y ciclos de vida*. España: Laboratorio Nacional de Calidad de Software, 2009. Disponible en: http://www.inteco.es/ (Accedido el 10 de enero de 2014).

[2] H. F. Castro, T. Velásquez and M. P. Rojas. "Adoption of project management methodologies in Colombia project manager's perspective", *J. Phys. Conf. Ser.,* vol. 1126, Conference 1, 2018. Available: https://iopscience.iop.org/article/10.1088/1742-6596/1126/1/012032

[3] Project Management Institute (PMI), *A Guide to the Project Management Body of Knowledge* (PMBOK® Guide) – Sixth Edition, Official Spanish Translation, Paperback PMI, 2017.

[4] M. Fowler, The New Methodology, Technical report, 2005. [Online]. Available: http://www.martinfowler.com/articles/newMethodology.html

[5] L. E. Gimson Saravia, *Metodologías ágiles y desarrollo basado en conocimiento*. Buenos Aires: Universidad Nacional de la Plata, Facultad de Informática, 2012.

[6] A. English, "Extreme Programming: It's Worth a Look", *IEEE IT Pro,* vol. 4, No. 3, pp. 48-50, May/June 2002.

[7] S. Ambler, *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process.* New York: John Wiley & Sons, Inc., 2002.

[8] J. P. Rodríguez, "Estudio sobre las experiencias de formación en metodologías ágiles realizado en los años 2015-2016 en los cursos de Análisis y Diseño de Sistemas e Ingeniería del Software del programa de Ingeniería de Sistemas de la Universidad Francisco de Paula Santander". Documento de Trabajo, 2017.

[9] C. J. Parada, M. P. Rojas and F. H. Vera, "Study of the use of agile methodologies in the development of software construction projects in Colombia", *J. Phys. Conf. Ser.,* vol. 1126, Conference 1, 2018. Available: https://iopscience.iop.org/article/10.1088/1742-6596/1126/1/012056/pdf

[10] C. J. Parada, "Primer Informe Técnico Proyecto Metodología Ágil aplicada a los proyectos de Ingeniería de Sistemas de la UFPS que involucre el desarrollo de Aplicaciones Móviles", 2014.

[11] A. Cockburn, *Agile software development*. Addison Wesley, 2001.

Capítulo II
Guía Técnica para la Especificación de Requisitos de Software en el Desarrollo de Aplicaciones para Ciudades Inteligentes

Fabio Alberto Vargas Agudelo - fvargas@tdea.edu.co
Docente-investigador, Tecnológico de Antioquia - I. U., Medellín, Colombia
Jaime Alberto León Rincón - jimmyleon7@gmail.com
Docente-investigador, Tecnológico de Antioquia - I. U., Medellín, Colombia
Darío Enrique Soto Durán - dsoto@tdea.edu.co
Docente-investigador, Tecnológico de Antioquia - I. U., Medellín, Colombia
Juan Camilo Giraldo Mejía - jgiraldo1@tdea.edu.co
Docente-investigador, Tecnológico de Antioquia - I. U., Medellín, Colombia

I. INTRODUCCIÓN

Las ciudades inteligentes (*Smart cities*) son un fenómeno que crece con el avance de la tecnología, usan componentes tecnológicos y se constituyen de diversas características que las definen. Para que dichas ciudades sean una realidad se requiere un conjunto de elementos entre los que se destacan el hardware y el software, siendo el software uno de los componentes principales y complejo a la hora de desarrollarlo, si se tiene en cuenta que para su funcionamiento tiene que interactuar con diversos componentes e infraestructuras tecnológicas. Esto plantea grandes retos para los desarrolladores en la especificación de los requisitos, lo que obliga a explorar y proponer nuevas formas de llevar a cabo esta actividad.

Se evidencia un conjunto limitado de metodologías para la fase de especificación de requisitos en el desarrollo de estas aplicaciones, por tal motivo se hace necesario proponer una guía técnica para esta fase, la cual influirá de manera positiva en las fases posteriores del desarrollo de software, para lograr un producto de calidad que satisfaga las necesidades cambiantes y exigentes de los habitantes de estas ciudades.

En el capítulo se presenta una guía técnica para la especificación de requisitos de software para ciudades inteligentes, que recoge los elementos y características esenciales de este tipo de aplicaciones, procurando una especificación de requisitos pertinente y no ambigua. Todo ello basado en un conjunto de formatos que compendian la información necesaria en este tipo de aplicaciones.

El capítulo se presenta de la siguiente manera: en la sección 1 se contextualiza el término *ciudad inteligente*; en la sección 2, se busca generar un inventario con los componentes y características necesarios para el desarrollo de aplicaciones en

ciudades inteligentes; en la sección 3, se presentan metodologías de ingeniería de requisitos utilizadas en aplicaciones para ciudades inteligentes; en la sección 4 se exponen los resultados, mediante la guía técnica para la especificación de requisitos funcionales y un ejemplo de aplicación; por último, se presentan las conclusiones.

Este capítulo es producto de la investigación para trabajo de grado del estudiante Jaime Alberto León Rincón, para optar al título de Ingeniero en Software, y de la experiencia investigativa de los autores en el área.

II. METODOLOGÍA

Este trabajo se llevó a cabo a partir de una revisión de literatura sobre el concepto de ciudad inteligente, el inventario de componentes y características propias de este tipo de aplicaciones, y el análisis de metodologías de ingeniería de requisitos aplicables para ciudades inteligentes.

A. Ciudad Inteligente

Según [1], las ciudades inteligentes responden a proyectos piloto basados en el uso de las TIC, para la transformación de la ciudad en un entorno más habitable y competitivo a nivel mundial.

Según [2], las ciudades inteligentes son aquellas que aplican las Tecnologías de la Información y la Comunicación (TIC), en procura de que sus habitantes mejoren la calidad de vida, y alcancen un desarrollo sostenible utilizando sus recursos de la mejor forma posible.

De acuerdo con [3], una ciudad es inteligente cuando al menos dispone de una iniciativa que aborde las siguientes dimensiones: *Smart Economy*, *Smart People*, *Smart Mobility*, *Smart Environment*, *Smart Governance* y *Smart Living,* mediante el uso de las tecnologías para mejorar la competitividad y asegurar un futuro más sostenible.

Según [4] las ciudades inteligentes utilizan y aplican las nuevas tecnologías en la administración de los servicios urbanos: Servicios sociales, distribución de energía y gobernanza.

En [5] se define una ciudad inteligente como una ciudad innovadora que utiliza Tecnologías de la Información y Comunicación (TIC) y otros medios para mejorar la toma de decisiones, la eficiencia de las operaciones, los servicios urbanos y su competitividad; a la vez que se garantiza la atención de las necesidades a las generaciones actuales y futuras en relación con los aspectos económicos, sociales y medioambientales.

B. Características y Componentes

En la Tabla 1 se relacionan las características y componentes propios de aplicaciones para ciudades inteligentes, de acuerdo al análisis de un conjunto de sistemas desarrollados.

Tabla 1. Características y componentes de las aplicaciones para ciudades inteligentes

Sistema	Componentes	Características
Parqueaderos inteligentes (*Smart Parking*) [6]	- Sensores de proximidad inductivos - Detector de metales - *Loop* de piso - RFID - Pulsadores - Placa electrónica Arduino - Reloj en tiempo real - Wifi - PLC (Programmable Logic Controllers) - LED - Sensores de presencia - Circuito detector de peso - Sensores de tope - Semáforos	- Infraestructura de comunicaciones fija y móvil - Interfaz natural e intuitiva - Sensible al contexto - Fiabilidad y seguridad - Multimodal - Ubicuidad - Procedimiento concurrente - Fiabilidad - Capacidad de evolución - Tiempo real - Adaptativo - Invisible - Embebido - Predictivo - Personalizado - Dispositivos invisibles - Infraestructura de comunicaciones fija y móvil - Redes dinámicas de dispositivos distribuidos - Interfaz natural e intuitiva - Fiabilidad y seguridad - Computación ubicua - Comunicación ubicua - Interfaces inteligentes
Señalización inteligente del tráfico [7]	- Cámaras SmartView - Redes de sensores inalámbricas - CHIP - Raspberry Pi - Bluetooth - ZigBee - Wifi - Ethernet - GPRS/3G - Middleware Servicios Web - API REST - Magnetómetro para detección de vehículos - Semáforos - Paneles informativos	
Iluminación inteligente de las calles [8]	- Luminarias LED - Sistema de conversión AC/AC - Optoacoplador - Sensor de luz - Wifi - Plataforma Arduino - Arduino Shield Ethernet - Arduino Shield Wi-fi - Arduino Shield Yun - Drivers para módulos LED - Pulsadores - Sensores o detectores - Unidades de control - Repetidores - Herramientas de configuración y de monitorización	
Detección de disparos [9]	- Sensores acústicos - Máquina de clasificación - Notificaciones push - Micrófonos - Alarmas - Cámaras	

Sistema	Componentes	Características
Calidad del aire/partículas en aire [10]	- Transmisores de temperatura - Conjunto integrado de sensores: Pluviómetro, anemómetro, temperatura, humedad, radiación solar y UV - Sensor Dräger X-am 5000 - PM10-Bombas para material particulado - Equipos de posicionamiento – GPS - Purificadores de aire - Sensores inteligentes para monitoreo de calidad del aire - Sensor de polvo - LED infrarrojo y un fototransistor - Sensores de CO2 - Sensores de Telaire - Tecnología infrarroja no dispersiva (NDIR) - Fuente infrarroja - Guía de ondas - Filtros infrarrojos - Detector termopila micromecanizado	- Infraestructura de comunicaciones fija y móvil - Interfaz natural e intuitiva - Sensible al contexto - Fiabilidad y seguridad - Multimodal - Ubicuidad - Procedimiento concurrente - Fiabilidad - Capacidad de evolución - Tiempo real - Adaptativo - Invisible - Embebido - Predictivo - Personalizado - Dispositivos invisibles - Infraestructura de comunicaciones fija y móvil - Redes dinámicas de dispositivos distribuidos - Interfaz natural e intuitiva - Fiabilidad y seguridad - Computación ubicua - Comunicación ubicua - Interfaces inteligentes
Gestión energética [11]	Sensores con el sistema LCN (Red de Control local) Sensor de temperatura y receptor infrarrojo Sensor de humedad Detector de movimiento y sensor de luminosidad Detector de dióxido de carbono Sensor de luminosidad exterior Sensores para la gestión de la energía Sensores de luz Detectores de movimiento y de presencia Sensores de temperatura	

Fuente: Elaboración propia.

C. Metodologías de Ingeniería de Requisitos Aplicables

En [12] se propone una metodología de requisitos de software para aplicaciones de *AmI (Ambient Intelligence*), la cual sugiere una fase de modelado de negocios, una fase de modelado de tecnología, una fase de modelado de interacción y una fase general de modelado, y en esta última fase se proporciona información acerca de las metas que se desea lograr.

Los autores en [13] proponen la metodología de especificación *Agile Requirements de SCRUM,* aplicable a soluciones móviles, la cual plantea un documento de *back log* que define las diferentes funcionalidades y requerimientos del producto que desea el *stakeholder*. Una vez se reúnen estas funcionalidades del producto, se inicia la fase de priorización.

En [14] se presenta una metodología denominada DoRCU (Documentación de Requerimientos Centrada en el Usuario), caracterizada por su flexibilidad y orientación al usuario, la cual se apoya en diversos métodos, técnicas y herramientas ya desarrollados por otros autores, pero sin comprometerse con las tendencias de un modelo en particular.

Una metodología para la ingeniería de requisitos de sistemas ubicuos - ABC-Besoins, es la propuesta en [15], la cual está orientada por metas y agentes y proporciona formalismos para representar el conocimiento básico de un dominio y proceder a identificar los requisitos; brindando continuidad en el proceso de Ingeniería de Requisitos (IR) desde la elicitación de requisitos hasta la generación de un modelo conceptual y de un modelo de diseño.

En [16] se propone la ingeniería de requisitos para ambientes inteligentes (R4IE), la cual establece objetivos de alto nivel, determina el ámbito de trabajo, identifica las partes interesadas, las tareas y funcionalidad y las cualidades de rendimiento del sistema, y determina los perfiles del interesado.

En la Tabla 2 se presenta un resumen de las metodologías analizadas con respecto a la guía técnica presentada en el capítulo, teniendo en cuenta las características propias de cada metodología en relación con el desarrollo de aplicaciones para ciudades inteligentes. Se destaca el valor agregado de la propuesta presentada con respecto a la especificación de requisitos. Las características que se comparan en la Tabla 2 son las siguientes.

Uso de formatos: La Utilización de formatos o plantillas para la especificación de requisitos.

Interacción humana – tecnología: la capacidad de la tecnología de dar respuesta a impulsos dados por los humanos y que deben reflejarse en los requisitos.

Interacción tecnología – tecnología: la capacidad de responder a solicitudes dadas entre tecnología y tecnología para llevar a cabo una tarea determinada.

Componentes tecnológicos: la participación de componentes tecnológicos necesarios en la definición de los requisitos.

Características de las ciudades inteligentes: Las metodologías contemplan particularidades explícitas de las ciudades inteligentes.

Tabla 2. Resumen de las metodologías analizadas con respecto a la guía técnica presentada en el capítulo

	Uso de formatos	Interacción humano – tecnología	Interacción tecnología – tecnología	Componentes tecnológicos	Características de las ciudades inteligentes
Metodología para modelar la inteligencia ambiental aplicaciones que usan i * framework.	No	Sí	Sí	Sí	Sí
Metodología de Especificación Agile Requirements de SCRUM	Sí	No	No	No	No
Metodología DoRCU para la Ingeniería de Requerimientos	Sí	No	No	No	No
Ingeniería de requisitos para Ambientes Inteligentes (R4IE).	Sí	Sí	Sí	Sí	Sí
Modelo de contexto y de dominio para la ingeniería de requisitos de sistemas ubicuos	Sí	No	No	Sí	No
Guía técnica para la especificación de requisitos para aplicaciones en ciudades inteligentes (propuesta)	Sí	Sí	Sí	Sí	Sí

Fuente: Elaboración propia.

III. RESULTADOS

A continuación, se presenta la guía técnica para la especificación de requisitos de software en ciudades inteligentes, que recoge las buenas prácticas de algunas metodologías y los formatos aplicables. Todo esto con base en la revisión y el análisis realizados en las sesiones anteriores con respecto al concepto de ciudades inteligentes, sus componentes y características. En la figura 1 se presenta la visión general de la guía técnica.

Fig. 1. Visión general de la guía técnica

A. Fase 1: Definición del Entorno

Denomina al conjunto de características o normas que regulan, describen y definen el funcionamiento de los lugares en los cuales funcionará el sistema que se va a desarrollar (parqueaderos, centros comerciales, hogares, empresas, negocios, etc.). El entorno se caracteriza por un conjunto de factores que delimitan el ámbito en el que estos lugares actúan, y establece las condiciones en que estos se tienen que desenvolver. Los factores son: económicos, sociales, tecnológicos, ambientales, de producción, de estructura organizacional y de recursos humanos.

B. Fase 2: Entorno Tecnológico Actual

Describe todos los elementos tecnológicos que están presentes en el entorno y que son relevantes para su funcionamiento (máquinas, herramientas, servicios, sensores, computadores, Smartphones, impresoras, cámaras de TV, etc.). Estos elementos son primordiales para reconocer la capacidad inteligente del entorno.

Hardware: Descripción de los componentes tecnológicos que están disponibles en el entorno en que se desarrollará la aplicación, y que ésta puede utilizar, tales como circuitos de cables y circuitos de luz, placas, actuadores, cámaras SmartView, controladores, sensores, etc.

Software: Se describen los programas informáticos que están instalados y disponibles en el entorno en el cual se desarrollará la aplicación.

C. Fase 3: Necesidades del Entorno

Describe las carencias y posibles mejoras que puede tener el entorno y que impiden desempeñar adecuadamente las funciones que se llevan a cabo. Se describen también los procesos que se pretende optimizar y darles solución, al igual que se busca el aprovechamiento de factores ambientales, energéticos, espaciales, entre otros. Esto con el fin de crear bienestar entre las personas que interactúan en el entorno.

D. Fase 4: Funcionalidad para Implementar en el Entorno

Describe la funcionalidad general que tendrá el software para solucionar la necesidad identificada. La descripción de la función es importante para detectar posibles necesidades que se tendrán en cuenta en la especificación de requisitos para un correcto desarrollo del software.

E. Fase 5: Especificación de Requisitos

Para la especificación de requisitos se tendrá en cuenta el siguiente proceso:

Identificación del requisito: Es un código alfanumérico que identifica al requisito.
Nombre del requisito: Denomina el título del requisito.
Componente: Lista de elementos eléctricos, electrónicos, electromecánicos y mecánicos con los cuales está asociada la funcionalidad del requisito.
Característica asociada: Se define la cualidad del requisito. Algunas cualidades son:

- Embebido: Dispositivos integrados en otros productos, y que a su vez controlan determinadas funciones en un sistema de computación en tiempo real.
- Natural: Ausencia de interacción con los sistemas informáticos.
- Invisible: Los dispositivos se deben integrar en el entorno de manera natural y discreta.
- Ubicua: Los dispositivos están presentes en cualquier parte del entorno inteligente que se va a desarrollar, deben interactuar entre sí y con el usuario, permitiendo al sistema acceder a la información que estos ofrecen desde cualquier lugar, mediante la implementación de redes inalámbricas.

Descripción del requisito: Exposición en detalle de la necesidad que se va a solucionar con el software, debe ser adecuada y precisa. Es importante que esta descripción contemple los siguientes elementos: Función que va a cumplir; objeto sobre el que se aplicará la función; acción asociada al componente inteligente empleado. Por ejemplo: Encender/ apagar las lámparas automáticamente al detectar movimiento vehicular o humano.

Función	Objeto	Función asociada
Encender / apagar	Las lámparas	Detectar movimiento

Prioridad: Se define el nivel de importancia del requisito, de acuerdo a factores como aspectos técnicos, de recursos humanos, etc. Se asigna un valor numérico o un valor enumerado: prioridad alta, media y baja.

Restricciones: Indicar las limitaciones para tener en cuenta en el diseño y/o desarrollo de la solución asociadas al requisito, tales como: normas aplicables, estándares relacionados, operación, implementación, etc.

Interacción humano - tecnología: El usuario interactúa directamente con los componentes tecnológicos asociados a la solución del requisito de la aplicación que se va a desarrollar.

Interacción tecnología - tecnología: Interacción de los componentes tecnológicos sin la intervención humana.

En la Tablas 3 y 4 se muestran las plantillas aplicables.

En la Tablas 5 y 6 se muestra un ejemplo de aplicación de las plantillas en un parqueadero inteligente.

Tabla 3. Plantilla para la definición del entorno

FASE I	Definición del entorno
FASE II	Descripción del entorno tecnológico actual
Hardware	
Software	
FASE III	Establecer las necesidades del entorno
FASE IV	Descripción de la funcionalidad para implementar en el entorno

Fuente: Elaboración propia.