Revista Educación en Ingeniería
ISSN 1900-8260
Julio a Diciembre de 2015, Vol. 10, N°. 20, pp. 98-109 • © 2015 ACOFI • http://www.educacioneningenieria.org
Recibido: 13/10/2015 • Aprobado: 12/11/2015

ESCOGER UNA METODOLOGÍA PARA DESARROLLAR SOFTWARE, DIFÍCIL DECISIÓN

CHOOSING A SOFTWARE DEVELOPMENT METHODOLOGY, HARD DECISION

Lucy Nohemy Medina Velandia
Fundación Universitaria Los Libertadores, Bogotá (Colombia)

Wílmer Mesías López López
Fundación Universitaria Los Libertadores, Bogotá (Colombia)

Resumen

Esta investigación tuvo como objetivo diseñar un método para desarrollar *software* sencillo, fácil de aprender, modificar y ejecutar; iterativo, incremental, cooperativo, adaptable, que como eje transversal aplicara la calidad. Está dirigido a empresas medianas y pequeñas, así como a estudiantes que llevan a cabo proyectos. Se utilizó investigación básica y aplicada. El instrumento principal fue la encuesta, realizada a 50 empresas desarrolladoras de *software*, repartidas en grandes, medianas y pequeñas, y a estudiantes de los dos últimos semestres de Ingeniería de Sistemas. A partir de los resultados se identificaron las necesidades existentes en el sector para desarrollar *software* de una manera más sencilla y corta. Se encontró que los desarrolladores de *software* no ven con buenos ojos las metodologías llamadas duras o tradicionales porque deben seguir una secuencia en cada etapa y reunir una voluminosa documentación. Por otra parte, si utilizan metodologías ágiles se presentan inconvenientes entre los grupos de trabajo, debido a la diferencia de criterios, a las constantes reuniones que no las ven como ventaja sino como pérdida de tiempo, además del estrés que se genera dentro por la cantidad de tareas que deben cumplir en tan corto tiempo para desarrollar el *software*. Frente a los resultados generados, se elabora el método Winner, que se trabaja en cuatro etapas sencillas y se caracteriza por emplear tablas, las cuales se factorizan y sólo resultan catorce en todo el proceso.

Palabras claves: metodologías ágiles; métodos de desarrollo de *software*, Winner.

Abstract

This research aimed to design a method for developing software that is simple, easy to learn, easy to modify and run, iterative, incremental, collaborative, adaptable, applied as a transverse axis to quality.

It is aimed at small and medium enterprises, as well as students in developing their projects. From this objective, surveys were conducted to companies engaged in software development and university students in the last semesters. Thereafter, the needs that have to develop quality software more easily identified and faster. The research metodology used was basic and applied. The main instrument was a survey applied to fifty software development companies spread over large, medium and small; students in the last two semesters of the Engineering Systems also surveyed. The results showed that software developers do not look favorably methodologies called hard or traditional, because they have to follow many sequential steps at each stage and should make voluminous documentation; if they use agile methodologies appearproblems between the working groups due to the disagreement, the constant meetings that do not see as an advantage and a waste of time, in addition to the stress that is generated within the team by the presents many activities in such a short time to develop the software. Compared to the results generated, the Winner method works in four easy steps and characterized by use tables, which are factored and are only fourteen in the whole process of software development.

Keywords: agile methodologies, development methods software, Winner.

Introducción

Siendo las metodologías para desarrollar *software* los marcos que permiten la estructuración, planificación y control de un proyecto de *software*, se requiere que éstos sean concisos, ordenados, claros y específicos. Todo proyecto de *software* deberá planearse, estructurarse y desarrollarse con habilidad, paciencia y conocimiento. No obstante, son múltiples los problemas que se presentan al realizar dicha labor. Por ejemplo, cuando un equipo está trabajando y el jefe del grupo realiza excesiva presión para ejecutar la tarea; son muy frecuentes los cambios en los requerimientos o especificaciones, o estos no existen; muchos equipos de trabajo no documentan correctamente los proyectos, además de realizar numerosos cambios superficiales; y es muy común que se solicite agregar funcionalidades al *software* que no se consideraron al iniciar la labor. Uno de los problemas más frecuentes al desarrollar *software* es la carencia de un método científico durante la elaboración del proyecto.

El desarrollo de *software* requiere orden, disciplina y una excelente gestión para que dicha tarea sea eficiente y de calidad. Infortunadamente, las metodologías existentes hacen que el proceso se vuelva lento debido a tantas actividades que abarcan, lo que lleva a que el ritmo sea lento pues no están diseñadas para trabajar con incertidumbre. Lo anterior hace que en diversas oportunidades la planificación del proyecto no sea óptima y lleve a retrasos, y que los sistemas nazcan deteriorados en razón de la cantidad de mantenimiento y parches que se efectúan en el proceso, como consecuencia de los defectos que se presenten. Por otra parte, puede suceder que los requisitos hechos por el usuario no se comprendan bien, o no se especifiquen antes de iniciar un proyecto, o por el cambio de personal dentro del equipo.

Numerosos han sido los autores de metodologías para desarrollar *software*. Las han creado ágiles, incrementales, iterativas, por etapas, evolutivas y secuenciales, entre otras. Todo estos marcos de trabajo se han utilizado para estructurar, planificar y controlar proyectos de *software*; pero en muchos casos, en lugar de facilitar las actividades requeridas para culminar el proyecto se hacen complejas y retrasan los proyectos por la cantidad de entregables e hitos que se deben cumplir al culminar cada etapa. Las razones anteriores llevan a pensar que escoger una metodología para desarrollar *software* no es fácil, es una dura tarea que en múltiples ocasiones quita tiempo y causa discusiones entre los integrantes de los equipos de trabajo. Otro problema es que cuando se aplican ciertas metodologías para desarrollar *software*, puede ocurrir que los proyectos se entreguen a destiempo, es decir, que cuando el *software* esté listo, la razón inicial para haberlo adquirido puede que haya cambiado y todos los esfuerzos realizados sean inútiles, es decir, se haya perdido, tiempo, dinero y esfuerzo.

Lo que se propone es un método para desarrollar *software* que proporciona agilidad en el desarrollo, control de calidad en el producto final, sencillez en

su uso, facilidad para comprenderlo y utilizarlo, y que además sea corto y manejable en equipos de trabajo grandes o pequeños.

En términos generales, la presentación de este escrito es la siguiente: en la primera parte se expone el problema que se despejará y la metodología por utilizar; enseguida, se presenta el estado del arte, que describe para qué se hace y cuál es la intención de la propuesta planteada. A continuación se plasman las fases que se siguieron para desarrollar el método y las actividades propuestas. Luego se muestran los resultados y el análisis de la investigación que sirvió como base para desarrollar el método.Se culmina con la explicación del método propuesto y las conclusiones.

Punto de partida

Cuando se desarrolla *software* se deben seguir actividades que forman parte de cada etapa. Si se realizan ordenadamente, el producto final será legible, sencillo y de fácil manejo. La propuesta de un método nuevo, llamado Winner, no implica olvidar que han sido muchas las metodologías realizadas, bien sean las tradicionales, también llamadas "pesadas", o las ágiles. Para construir el método propuesto se tomaron diecinueve (19) metodologías entre pesadas y ágiles, con el fin de realizar un análisis preliminar. De éstas, se escogieron sólo nueve que se estudiaron y compararon entre ellas profundamente, con el objeto de que sirvieran como apoyo para la propuesta que se presenta.

Winner parte del establecimiento de necesidades de los sectores que utilizan metodologías para desarrollar *software*. Se realizaron dos tipos de encuestas; por una parte, a los desarrolladores que trabajan en empresas dedicadas a dicha tarea, y por otra, a estudiantes de últimos semestres, quienes también deberán utilizar metodologías para realizar sus proyectos de grado. Las personas que crean *software* en casas de desarrollo indicaron que no toman en cuenta todos los pasos de las metodologías escogidas para culminar sus proyectos, dado que las actividades propuestas en cada fase son muy complejas y robustas para cumplir con toda su aplicación. Para suplir lo anterior, aplican la seleccionada, según su criterio, o combinando etapas y actividades de unas y otras metodologías.

A partir de este problema, se decide proponer un método sencillo, de fácil uso y con pasos reducidos, manteniendo la calidad del *software*. Igualmente, se reflexiona sobre los sectores y tipos de empresas a las que se dirige el método una vez concluido.

Como se mencionó, se tuvieron en cuenta dos tipos de encuestas como punto de partida, dirigidas a dos sectores diferentes, el educativo y el empresarial. En este último se consideraron variables como las plataformas, el tipo de empresa en la que se contesta la encuesta, el tamaño, los servicios que presta, el sector al que dirige su actividad, el sistema operativo que utiliza, las bases de datos y la metodología de desarrollo empleada para desarrollar *software*.

Por su parte, la encuesta dirigida a estudiantes de últimos semestres tenía como propósito conocer qué metodologías de las vistas durante sus estudios escogían, por qué y qué dificultad se les presentaba. Con los insumos obtenidos se inició el trabajo para conseguir el método propuesto.

El problema

Las encuestas se dirigieron a 50 empresas bogotanas desarrolladoras de *software* y a estudiantes de los dos últimos semestres de ingeniería de sistemas. Una vez analizados los datos, se establecieron formalmente los problemas que tienen las personas cuando siguen las metodologías que se encuentran en el mercado para desarrollar *software*. Un hallazgo importante es que las empresas desarrolladoras de *software* utilizan metodologías acomodadas, dependiendo de la necesidad del desarrollador, el analista, el diseñador o la misma empresa. Esta actividad hace que en muchos casos los productos carezcan de refinamiento y, por ende, su calidad no sea la mejor.

Otro problema que surge de las dos encuestas fue no seguir la secuencia de la metodología escogida y la adecuación según las necesidades de quien la practica. Por lo anterior, afirman los encuestados, cuando culminan los proyectos los resultados no reflejan la calidad esperada, se presentan errores o insuficiencias en el *software*, lo que lleva a corregir las aplicaciones en caliente, pues el producto está en manos del cliente y hay que crear parches en lo programado, lo que vuelve frágil el sistema.

Con respecto a estos inconvenientes, surgen las siguientes preguntas: ¿qué se espera solucionar con Winner?, ¿este método facilitará la construcción de

software y podrá utilizarse en desarrollos concretos con un alto grado de probabilidad de éxito, calidad y uso de todos los pasos que conforman sus etapas?

Qué han hecho los demás

En este aparte se trata de recuperar algo del conocimiento acumulado sobre el objeto de estudio de este escrito, como son las metodologías de desarrollo de *software*.

Parte de la historia de la ingeniería de software, en la cual están incluidas las metodologías de desarrollo de *software*, se inicia en 1965, cuando Edsger Dijkstra utilizó la famosa frase "crisis del *software*" por la presencia de muchos y variados problemas. Por nombrar algunos, la poca flexibilidad de los programas, su alto costo, la falta de comprensión y de verificación de los mismos, la complejidad, la continua adaptabilidad del *software* dependiendo de las necesidades de los usuarios, la poca estimación del tiempo y costos que se tenían para el desarrollo del proyecto; en fin, la calidad era muy baja, por cuanto las aplicaciones no cumplían con las especificaciones. A partir de dicha crisis surge la ingeniería de *software* renovada y con metodologías que intentan subsanar el problema expuesto; de todos modos, hoy persisten errores, así como la no concreción de tiempos y costos fiables en la culminación del proyecto.

Sin embargo, los problemas actuales son otros, como el poder computacional que a diario se incrementa y cambia constantemente; los bajos costos del *hardware,* lo cual hace que la computarización de los negocios y empresas aumente, el gran número de usuarios heterogéneos que concurren a la misma información o al mismo sitio, las nuevas necesidades y costumbres de los usuarios, el personal de desarrollo y mantenimiento con conocimientos diversos, la complejidad en las arquitecturas de *hardware* y de *software* y la diversidad de lenguajes de programación, entre otros.

Cockbun (2001) señala que en 1985 empiezan a surgir metodologías, tecnologías y herramientas que pretendían solucionar definitivamente los problemas expuestos, en especial la identificación concreta de costos, la planificación y la calidad de los productos desarrollados. Infortunadamente, no se utilizan las metodologías y la parte gerencial del proyecto como lo especifica la ingeniería de *software*, sino que las han desarrollado de acuerdo con las necesidades específicas del momento; de hecho, no existe un estándar para aplicarlo en todos los procesos de desarrollo de *software*.

Las metodologías surgen para ordenar y promover el desarrollo de *software* de calidad, por ello se han dispuesto numerosos modelos, entre los que se encuentran: clásicos, evolutivos, basados en componentes, métodos formales, modelos de calidad del *software*. También se han desarrollado diversas metodologías: estructuradas, orientadas a objetos y de desarrollo ágil, entre otras (Coad, Lefebvre y De Luca, 1999). Stapleton (1997) hace un recuento sobre diferentes tipos de modelos, entre otros:

El modelo prototipo, utilizado en los años noventa, consiste en realizar parcialmente el sistema e irlo compartiendo con el usuario para que experimente con él y de esta forma pueda determinar los requerimientos antes de desarrollar el producto final. No siempre es bueno utilizar este modelo debido a que el usuario cree que es el producto final y en ocasiones no le gusta. Es un modelo recomendado cuando no se conocen ciertamente las necesidades, pues guía al programador según las sugerencias del cliente.

El modelo espiral, creado por Boehm (1998), es evolutivo. Sus actividades se representan por medio de un espiral dividido en cuatro regiones o cuadrantes, alrededor de las cuales las interacciones se ejecutan dentro de los bucles o interacciones. Al terminar un giro e iniciar otro, se construye un nuevo modelo del sistema. El espiral puede combinarse con otros modelos, está orientado al desarrollo de sistemas grandes, pero con personal especializado que deberá decidir cuántas interacciones se requieren hasta terminar.

De este modelo existen dos variantes: el espiral de seis regiones y el espiral Winwin. En el primero, las actividades que se llevan a cabo son la comunicación con el cliente, la planificación, el análisis de riesgos, la ingeniería, la adaptación y construcción, y la evaluación del cliente.

Por su parte, Winwin adopta una comunicación con el cliente más activa, pues constantemente se le deben mostrar los requisitos solicitados y a la vez éste entrega los detalles necesarios para continuar. Lo más sobresaliente de este modelo e la identificación del sistema y los subsistemas, la designación de

condiciones de victoria de los dirigentes y el conjunto de condiciones de victoria de los directivos que se reúnen para establecer negociaciones.

Otro grupo de modelos que forma parte de la ingeniería son los llamados evolutivos, los cuales, como su nombre lo indica, son cambiantes, sobre todo en los requisitos de usuario y producto. Estos tipos de modelos se consideran iterativos y evolutivos, pues permiten realizar versiones de *software* cada vez más complejas y completas, hasta conseguir el logro planteado. De este tipo de modelo forman parte el espiral y los llamados iterativos e incrementales.

Uno de los principales distintivos de los modelos basados en componentes es la reutilización de código, el cual lleva a mejorar la calidad de las aplicaciones y a reducir el tiempo de desarrollo de los sistemas. Es tan interesante este tipo de modelo que se ha establecido la ingeniería de *software* basada en componentes (ISBC), dedicada a desarrollar sistemas que reutilizan componentes previamente diseñados y construidos. Según Abrahamsson et al. (2002), las ventajas que tiene el ISBC son: la calidad de las aplicaciones mejora sustancialmente, son más sencillos el mantenimiento y las pruebas del sistema y, por supuesto, la reutilización del *software*. Comprarlo, en lugar de desarrollarlo, permite que las etapas de elaboración sean más cortas, por supuesto hay mayor retorno de inversión y se mejora la funcionalidad.

Los métodos formales son necesarios porque los sistemas informáticos son muy grandes y complejos y si no se desarrollan eficazmente fallarán inevitablemente. Las correcciones certificadas al *software* representan dinero y deberán utilizarse en la industria para lograr la calidad esperada, pues cuando se usan estos modelos lógicos, matemáticos, desde las fases tempranas, los proyectos serán exitosos (Cockbun, 2001).

Por otra parte, las metodologías orientadas a objetos realizan el modelado del sistema a partir del dominio del problema y la construcción de objetos que se interrelacionen. Los primeros métodos que surgieron, fueron el Object Modeling Technique (OMT), desarrollado por James Rumbaugh; el Objetory, RUP o Métrica 3, entre otros. Las principales características de la metodología orientada a objetos es la reutilización del código. Los objetos diseñados y construidos con esta metodología permiten la computación distribuida, la cliente-servidor y la migración de aplicaciones, entre otras posibilidades.

Por último, se tratarán las metodologías ágiles, estudiadas por muchos autores. Herrera & Valencia (2007), en su artículo "Del manifiesto ágil. Sus valores y principios", realizan una síntesis de lo que ha sido la computación e indican el problema surgido de la popularización de las computadoras y la tecnología en general, de lo cual han surgido necesidades que han hecho avanzar también la forma como se desarrolla el *software*. A lo largo del tiempo, surge una nueva necesidad, y es la de trabajar proyectos de *software* cortos, pequeños, rápidos, por lo que las metodologías tradicionales no se ajustan a dicha tarea, lo que trae como consecuencia el surgimiento de las metodologías ágiles, que permiten acelerar los procesos, disminuir los tiempos y las etapas. Las actividades se convierten en ligeras, se organizan grupos de trabajo reducidos, pero cumpliendo con la misma calidad de las metodologías tradicionales.

Canós, Letelier & Penadés (2012) mencionan algunas características principales de las metodologías ágiles: aumentan la productividad, bajan costos y mejoran la atención al cliente; los equipos trabajan de forma eficiente y rápida, se proponen estrategias parciales y utilizables en corto tiempo, por la interacciòn con el usuario se extrema la calidad del producto. Además, facilitan la pronta corrección de errores técnicos.

Algunas metodologías ágiles

Scrum, propuesta por Ken Schwaber, Jeff Sutherland y Mike Beedle (2001), orientada a proyectos en los que cambian rápidamente los requisitos. El desarrollo de un proyecto se hace por medio de iteraciones, llamadas *Sprint*, las cuales duran 30 días, luego de lo cual se le presenta al cliente. Cada *Sprint* incrementa el desarrollo. También se efectúan reuniones constantes de quince minutos dentro del equipo de trabajo hasta que termine el proyecto.

Lean Development, desarrollada por Poppendieck & Poppendieck (2003), introduce un componente que implementa los cambios, pues éstos son considerados riesgos y por tal acción se vuelven oportunidades.

Crystal Methodologies, realizada por Alistair Cockburn (2001), se trata de un conjunto de metodologías ágiles

orientadas al desarrollo de *software*; su característica principal es que todas las actividades se centran en el equipo de trabajo, pues de él dependerá el éxito del producto, pero deberá invertir sus mejores esfuerzos, habilidades y destrezas, y regirse por las políticas de trabajo definidas y repartidas por colores. También se tiene en cuenta la disminución de artefactos producidos. Los recursos son los que limitan la invención y la comunicación del equipo.

Feature Driven Development, metodología establecida por Coad, Lefebvre % De Luca (1999), es un proceso iterativo de cinco pasos, cada uno con una duración máxima de dos semanas. Centrada en la fase de diseño e implementación del sistema, sus premisas para lograr el éxito se reúnen en una lista de características del producto final.

Dynamic Systems Development Method es un proceso iterativo e incremental en el que trabajan juntos equipo y usuario. Consta de cinco fases: estudio de viabilidad, estudio del negocio, modelado funcional, diseño, construcción e implementación. Existe retroalimentación entre las fases (Stapleton, 1997; Plonkaet al., 2014).

Adaptive Software Development, propuesta por Jim Highsmith y Orr K (2000), es un método iterativo que tolera cambios y está centrado en componentes de *software*, no tanto en las tareas. La componen tres fases: especulación (se inicia el proyecto, se planifican las características del producto), colaboración (se desarrolla) y aprendizaje (se revisa la calidad y se entrega al cliente). Esta última etapa es la de aprendizaje de los errores y se vuelve a iniciar el ciclo hasta culminar.

Extreme Programming (XP), creada por Kent Beck, es una metodología liviana para desarrollar *software*. Se caracteriza porque planifica, analiza y diseña en un mismo momento y durante el desarrollo. De esta forma se decide si se va por buen camino, y se evita el retroceso en etapas adelantadas, es decir, se trabaja a partir de prueba y error. El equipo está formado por entre dos y doce personas que trabajan en parejas. Es un método simple que desarrolla sólo lo que requiere, permite la retroalimentación constante, la toma de decisiones difíciles y remediar los errores tan pronto como se detectan; existe comunicación continua entre los clientes y el grupo de trabajo (Beck, 2000; Newkirk & Martin, 2001; Wake, 2001).

Fueron muchas las metodologías investigadas que sirvieron para que Winner tuviera una base y se ideara un aspecto diferenciador.

Metodología y fases del proyecto

El proyecto se desarrolló utilizando la metodología mixta, compuesta por la básica y la aplicada. Básica, porque se partió del marco teórico y los antecedentes, que sirvieron como punto de apoyo y partida para construir Winner. Aplicada, porque parte de lo encontrado en la investigación básica para generar el nuevo método, cuyo objetivo principal es lograr que se utilice en la industria del *software* para el desarrollo de sus proyectos y por los estudiantes para sus trabajos de clase y de grado.

Teniendo la metodología de investigación como guía, se desarrollaron las fases reales de construcción de Winner.

1. Fase de reconocimiento: se abordó toda la indagación preliminar, se trabajó sobre antecedentes, marco teórico, selección de bibliografía, materiales y autores que tratan las metodologías para desarrollar *software*.
2. Fase de revisión: se analizaron todas las metodologías encontradas, se clasificaron cronológicamente, se separaron las que servían como base para las preguntas de las encuestas y de acuerdo con la importancia de las metodologías indagadas se tuvieron o no en cuenta en el proyecto.
3. Fase de estudio observacional: se trabajó sobre el sector objetivo de la industria del *software* en Bogotá para encuestarlo. A la vez se consideraron los cursos universitarios superiores sobre los que se aplicarían las encuestas. Se prepararon los instrumentos para aplicarlos.
4. Fase de construcción de la propuesta: de acuerdo con los resultados obtenidos en el análisis de información de las encuestas, se elaboraron los lineamientos metodológicos para construir Winner y se concluyó la etapa.
5. Fase de documentación: desde el inicio del proyecto se construyeron documentos que respaldaran la investigación, dentro de la cual se describieron los pasos que se siguieron para elaborar Winner y las partes metodológicas de las que consta.
6. Fase de difusión: Winner, como resultado de investigación, se ha presentado en eventos académicos

y se ha aplicado en el desarrollo de proyectos de grado. Es uno de los métodos que se estudian dentro de la asignatura Ingeniería de Software I y II. Se ha expuesto en encuentros en Madrid, España, Venezuela, Brasil y Bogotá.

Resultados y análisis

Como se mencionó, fueron 19 las metodologías escogidas para analizar en principio. De esas sólo se seleccionaron nueve. A partir de su estudio se buscó la justificación para realizar Winner. Se efectuaron las encuestas a 50 desarrolladores profesionales de *software* en empresas dedicadas a dicha labor, y a estudiantes de los cursos de los semestres noveno y décimo de Ingeniería de Sistemas. Se analizaron los datos recogidos y se establecieron similitudes entre los dos tipos de encuestas. Una de ellas fue la falta de una metodología sencilla y de fácil uso para los desarrolladores de proyectos pequeños y medianos, que no generara tanto estrés como las metodologías ágiles; que no se tuvieran que seguir tantos pasos secuenciales; que no acarreara numerosa documentación y reuniones extenuantes cada día, mencionaron algunos desarrolladores, además del problema que se presenta en el grupo de programadores por roces continuos o discrepancias en las apreciaciones sobre un mismo tema.

En cuanto al tamaño de las empresas en las que se realizó la encuesta, hubo 10 grandes (20 %), 25 de mediano tamaño (50 %) y 15 pequeñas (15 %). Lo anteiror muestra que las grandes empresas generan menos permisos para conocer sus proceso y son las que menos se han creado en Bogotá, mientras que las medianas facilitaron que se conocieran sus procedimientos para elaborar *software* y son muchas más las que existen en la ciudad de Bogotá. Un fenómeno curioso es que las empresas pequeñas son las más numerosas, pero también las que tienen más dificultad para ser identificadas y acceder a sus procesos, debido a que sólo trabajan entre tres y cinco desarrolladores en todas las actividades de creación de software.

En cuanto a los sectores a los cuales dirigen los servicios las empresas desarrolladoras de *software* escogidas para la evaluación, el 59 % de los esfuerzos de las casas que hacen *software* va a las grandes y medianas empresas, mientras que la pequeña y la micro se encuentran desprotegidas o simplemente no acuden a las casas fabricantes, pues según opinan los encuestados es "muy costoso", por lo que acuden a los desarrolladores independientes. Sólo el 41 % de estas empresas está cubierto por las casas desarrolladoras de *software*.

El portafolio que ofrecen las empresas desarrolladoras de *software* encuestadas se caracteriza por ofrecer servicios para satisfacer necesidades de desarrollo a la medida, consultorías, mantenimiento de *software*, gerencia de proyectos, aseguramiento de la calidad. Los resultados muestran que los dos servicios que más se requieren son los desarrollos a la medida y las consultorías (55 %), mientras que el mantenimiento de *software* equivale a un 15 %, porcentaje realmente alto debido a que se realiza *software* a la medida, lo que incrementa la necesidad de hacer mantenimiento individual.

En muchas ocasiones, manifiestan los encuestados, se requieren personas con experiencia en dirección de proyectos de *software*, por lo que acuden a las casas desarrolladoras para contratar servicios en la gerencia de proyectos (sólo el 14 % lo requiere). Aunque todo proyecto que se emprenda debe contar con aseguramiento de la calidad dentro de su diario quehacer, las empresas sí contratan los servicios en esta línea, con el objeto de que sus propios desarrolladores o el *software* adquirido no se salga de esta característica tan importante.

La actividad de las empresas desarrolladoras de *software* está dirigida principalmente al sector de la banca (20 %), que es mucho más fuerte que los demás. Por ejemplo, al sector estatal sólo le corresponde el 18 %, a la industria el 16 %, al sector financiero el 15 %, al de seguros y de salud el 8 % y al educativo y de seguridad social el 5 %. Es comprensible que los sectores mejor cubiertos sean los de la banca y el Estado, pues en ellos se encuentra la mayoría de empresas de la capital de la República de Colombia. Por otra parte, los desarrolladores encuestados afirmaron que los problemas presentados al enfrentarse a un proyecto dependen del tipo de metodología que adopten para desarrollar. Si se trata de una tradicional, deben soportar la gran cantidad de actividades, la elaboración de una documentación voluminosa y presentar el consecutivo de algunas actividades, por lo que se generan tiempos muertos y en ocasiones la no interacción con el usuario, lo que lleva a que si los requerimientos no quedaron plenamente establecidos

desde el comienzo se deba volver atrás, lo que implica más tiempo y recursos, además de la prolongada etapa de consecución de requerimientos. Si se trata de una metodología ágil, afirman que la presión que produce la entrega de pequeños adelantos funcionales es muy grande; los roces que surgen en el equipo por diferencia de criterios, las constantes reuniones que para algunos son innecesarias, la frecuente interacción con los usuarios, que en ocasiones pretenden saber más que los desarrolladores o, por el contrario, no saben nada, complica la ejecución del proyecto.

Método Winner

A partir de las necesidades, problemas y conclusiones encontradas en el numeral anterior, se crea el método Winner para desarrollar *software*. Está organizado para que su uso sea sencillo y lo puedan utilizar empresas medianas y pequeñas, estudiantes o desarrolladores. Algunos criterios convenidos para construirlo fueron: sencillez; de fácil aprendizaje, modificación y ejecución; iteratividad; incrementalidad; cooperatividad; adaptabilidad. Además, como eje transversal se tuvo en cuenta uno de los factores más importantes: la calidad.

Una de las principales características de Winner es que se basa en tablas, principales artefactos al concluir cada una de las etapas. Para que el método sea completo, se enmarca dentro de la teoría general de sistemas, por cuanto debe seguirse un desarrollo ordenado e integrado y las partes deben interactuar buscando siempre el orden de las cosas y objetivos regulados.

A. Etapas de Winner

Figura 1. Etapas de Winner. Fuente: los autores

Las cuatro etapas que conforman Winner (figura 1) están enmarcadas en la teoría general de sistemas (TGS), que facilita el estudiarlas y analizarlas sin perder de vista la perspectiva global del proyecto que se esté ejecutando, manteniendo durante su desarrollo el orden, la regularidad y la carencia de azar en cada paso.

En la primera etapa, de comprensión y conocimiento, se trata de conocer a fondo la situación actual de la empresa y todo lo que se refiere a infraestructura de *hardware* y *software*, así como el trabajo que ejecutan las personas involucradas en el desarrollo del trabajo, por supuesto las necesidades y problemas por solucionar.

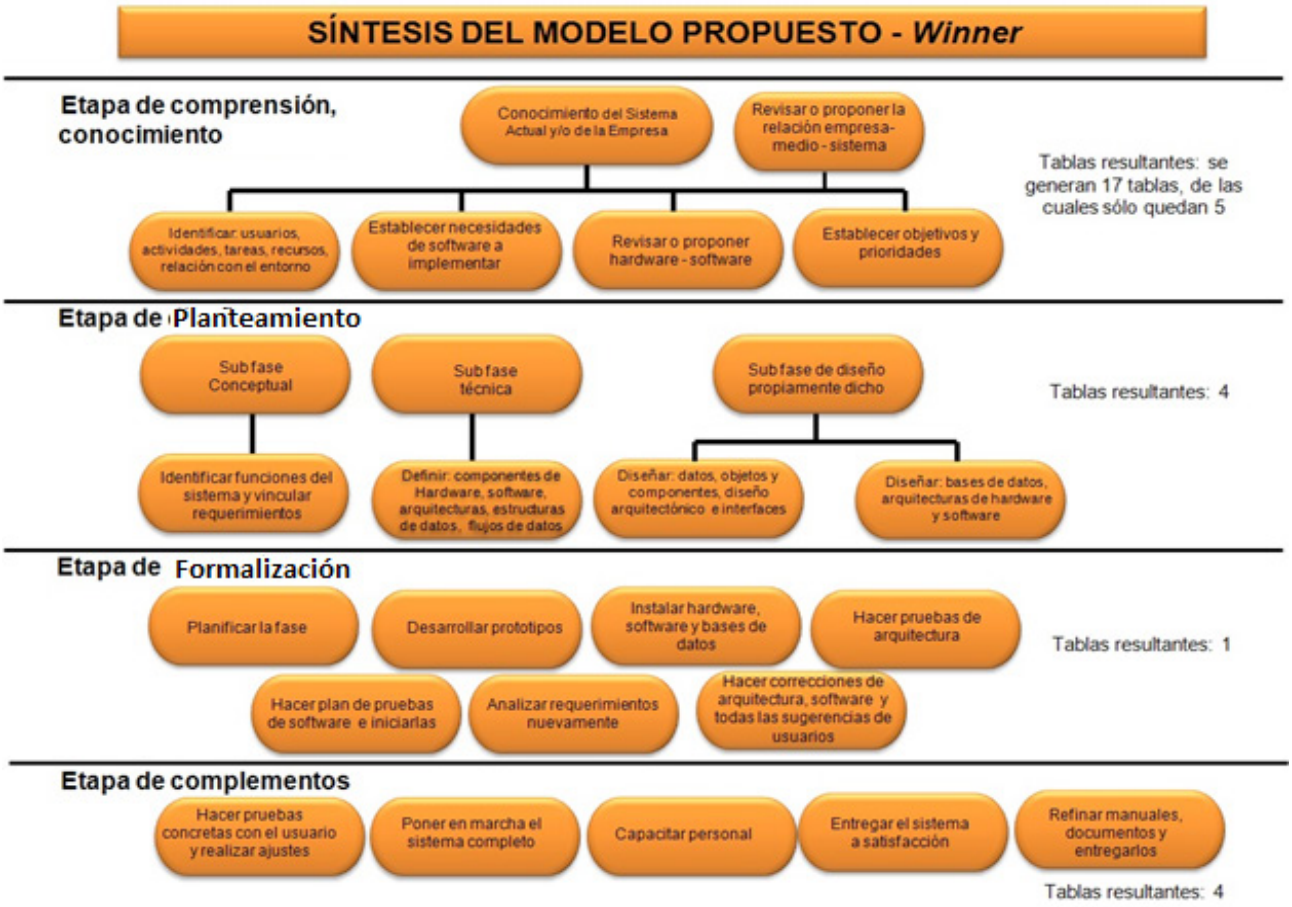
En el planteamiento se identifican las funciones del sistema y su vinculación a cada uno de los requerimientos encontrados en la primera etapa; se diseñan los datos y los flujos de datos, y los objetos; se definen los componentes de *hardware* y *software*, y las estructuras de componentes; se hace el diseño arquitectónico y de las interfaces del sistema; finalmente, se diseña la base de datos y la arquitectura de *hardware* y *software*.

En la formalización, de especial cuidado y vital para la entrega del proyecto, se desarrollan todos los prototipos planeados; se instalan el *hardware*, el *software* y la base de datos; se hacen pruebas de funcionamiento de la arquitectura y se continúa con

las pruebas del *software* de la mano del usuario o cliente, se analizan los requerimientos nuevamente y si se requieren ajustes deberán realizarse de acuerdo con las sugerencias sobre los posibles errores.

La etapa de complementos es de trabajo colaborativo. En ella los usuarios y el equipo de trabajo deben afrontar y abordar diferentes tareas juntos; se pone en marcha el sistema completo; se hacen pruebas concretas con el usuario; se capacita al personal requerido; se refinan los manuales y los documentos. Se proporciona dicho material y se da a conocer el plan de mantenimiento que se ejecutará. Finalmente, se entrega a satisfacción el sistema completo.

Figura 2. Síntesis del método Winner. Fuente: los autores



Se presenta la síntesis del método Winner (figura 2), con sus cuatro etapas y cada uno de los procesos, actividades y tareas que componen cada fase.

Se obtienen, como consecuencia, diversos componentes físicos de información, que para el método que se presenta son tablas que se van creando a medida que avanza cada una de las etapas. Las tablas se van refinando, de tal forma que al terminar cada fase y luego de factorizarlas, se obtienen sólo unas pocas. Las tablas que se obtienen en cada fase se presentan a continuación (cuadros 5, 6, 7 y 8).

Tabla 5. Resultados de la etapa de comprensión y conocimiento.

Nombres de las tablas de resultado	
Tabla descripción de objetivos	Tabla tipo de usuario
Tabla de usuarios	Tabla de requisitos
Tabla de actividades	Tabla objetivos del papel
Tabla objetivo-papel-actividad	Tabla descripción de tareas
Tabla descripción de recursos	Tabla tareas y recursos
Tabla relaciones	Tabla resultado fase uno
Tabla papel de usuario	Tabla de recursos
Tabla objetivo-actividad	Tabla papel-objetivo-actividad-tarea
Tabla actividades y tareas	

Tabla 6. Resultados de la etapa de planteamiento.

Nombres de las tablas	
Tabla de requerimientos y función	Tabla tipo de componente
Tabla jerarquía componente	Tabla requerimiento y función
Tabla tipo componente	Tabla jerarquía componente
Tabla de procesos	

Tabla 7. Resultados de la etapa de formalización.

Nombre de la tabla
Tabla humano-responsabilidad-recurso Tabla tipo de componente

Tabla 8. Resultados de la etapa de complemento.

Nombres de las tablas	
Tabla documentación	Tabla pruebas
Tabla mantenimiento	Tabla capacitación

Luego de tener las tablas en cada fase, se relacionan y al final del proyecto sólo quedan 14 tablas en total que se muestran a continuación (tabla 9).

Tabla 9. Tablas finales resultantes.

Etapa	Nombre de la tabla
Comprensión, conocimiento y análisis	Tabla de usuarios Tabla de requisitos Tabla de recursos Tabla de resultados fase uno Tabla de relaciones
Diseño	Tabla de requerimientos y función Tabla tipo dcomponente Tabla jerarquía componente Tabla de procesos
Ejecución	Tabla humano-responsabilidad-recurso
Complementos	Tabla pruebas Tabla documentación Tabla capacitación Tabla mantenimiento

Conclusiones

El método Winner se ha probado en el desarrollo de trabajos de grado. Con él se han concluido diez proyectos dirigidos al sector industrial, específicamente a la venta y reserva de boletas de eventos, y de servicios (salud, arte, educación y seguridad).

A los estudiantes que desarrollan proyectos de grado con este método se les ofrece asesoría permanente y se les hace seguimiento con el fin de realizar un análisis y determinar el nivel de dificultad en el momento de la aplicación. También han manifestado su agrado al utilizar el método, pues lo ven sencillo, rápido y manejable por medio de las tablas que se crean como artefactos al finalizar cada etapa; a la vez, expresaron que no se requieren muchos pasos para lograr el objetivo.

Terminada la aplicación del método en por lo menos diez proyectos de grado dirigidos a diversos sectores de la industria y el comercio, éste se validará mediante el establecimiento de métricas que permitan probar su complejidad y eficiencia. Aunque hoy existen numerosas métricas, es evidente que no se han estandarizado y como no hay acuerdo sobre este asunto, se construirán las propias. Este es el siguiente proyecto por realizar.

Winner es un método creado para servir al desarrollo de *software* de forma sencilla, de tal manera que quienes lo usen podrán, según su criterio, valerse o no de un lenguaje de modelado, puesto que las tablas que se proponen en cada etapa hacen que se comprenda fácilmente, por cuanto las tablas se van relacionando unas con otras hasta obtener una sola, sin perder de vista ninguna de las creadas, pues éstas se utilizan en las etapas posteriores.

Se está trabajando con algunas empresas desarrolladoras de *software*, en las cuales se implementará el método para establecer si efectivamente es de ayuda para las personas que lutilizan y las que crean *software*.

Referencias

Abrahamsson, P., Salo, O., Ronkainen, J. & Warsta, J. (2002). "Agile software development methods Review and analysis". *VTT Publications*.

Boehm, Barry. (1988). "A Spiral Model for Software Development and Enhacement". *Computer*, 21(5).

Canós, J.H., Letelier, P. & Penadés, M. C. (2012). "Metodologías ágiles en el desarrollo de *software*". España: Universidad Politécnica de Valencia.

Coad, P., Lefebvre E. & De Luca J. (1999). J*ava Modeling In Color With UML: Enterprise Components and Process*. Prentice Hall.

Cockbun, A. (2001). *Agile Software Development*. Addison Wesley.

Herrera, U. E. & Valencia, A. L. (2007). "Del manifiesto ágil sus valores y principios". *Scientia et Technica*, 13(34). Universidad Tecnológica de Pereira.

Newkirk, J. & Martin, R.C. (2001). *Extreme programming in practice*. Addison-Wesley.

Poppendieck, M. & Poppendieck, T. (2003). *Lean software development: an agile toolkit for software development managers*. Addison Wesley.

Plonka, L. et al. (2014). "UX Design in Agile: A DSDM Case Study". Agile Processes in Software Engineering and Extreme Programming. Springer International Publishing, 1-15.

Schwaber, K., Beedle, M. & Martin, R.C. (2001). *Agile Software Development with Scrum*. Prentice Hall.

Stapleton, J. (1997). *DSDM. Dynamic Systems Development Method: the method in practice*. Addison-Wesley.

Wake, W.C. (2001). (sf). "Extreme Programming Explored". Addison-Wesley.

Sobre los autores

Lucy Nohemy Medina Velandia

Ingeniera de sistemas, especialista en Pedagogía y magíster en Ingeniería de Sistemas. Profesora de planta de la Fundación Universitaria Los Libertadores (Colombia). Coordinadora de investigación del Programa de Ingeniería de Sistemas. Miembro activo del grupo de investigación Gridntic. Líder del semillero de investigación Sofía.
lunome@gmail.com

Wílmer Mesías López López

Profesional en ingeniería de sistemas, especialista en Gerencia de Proyectos de Telecomunicaciones y estudiante de la Maestría en Dirección Estratégica de Tecnologías de la Información. Director del Programa de Ingeniería de Sistemas de la Fundación Universitaria Los Libertadores. Editor de la revista *Coningenio*. Miembro del grupo de investigación Gridntic. Profesor universitario en la línea de Gerencia de Proyectos y Sistemas de Información.
wilmesiss@gmail.com