Revista Educación en Ingeniería
ISSN 1900-8260
Junio de 2011 • N°. 11 • Pp 55-65 • Publicada en línea por la Asociación Colombiana de Facultades de Ingeniería -ACOFI- www.acofi.edu.co
Enviado: 30/04/2010 • Aprobado: 18/04/2011

MEDUSA, MÉTODO PARA DESARROLLAR SOFTWARE

MEDUSA, METHOD FOR DEVELOPING SOFTWARE

Lucy Nohemy Medina Velandia
Universidad Sergio Arboleda, Bogotá (Colombia)

Resumen

El método MEDUSA aquí presentado es el resultado de la investigación que se enmarca dentro de la ingeniería de *software*, disciplina que se encarga de difundir técnicas y métodos para desarrollar software de calidad. Fue concebido a raíz de la información recolectada en cincuenta empresas desarrolladoras de *software* en Bogotá, y en cursos de 9º y 10º semestre de la Facultad de Ingeniería de Sistemas de la Universidad Sergio Arboleda, Bogotá (Colombia), que elaboran proyectos de grado.

El método consta de tres fases muy sencillas para desarrollar *software* y se distingue de otros porque al finalizar cada etapa de su ciclo, da como resultado pequeñas tablas que albergan toda la información requerida en la etapa y que servirá como entrada a la siguiente, enfatizando siempre en la calidad.

En este documento se plasma el resultado del proyecto, teniendo presente las metodologías más utilizadas en Bogotá, Colombia.

Palabras clave: artefacto, calidad, medusa, método para desarrollar *software*, ingeniería de software

Abstract

MEDUSA method presented here is the result of research that is part of the software engineering discipline that is responsible for disseminating techniques and methods for developing quality software. It was conceived as a result of the information collected in fifty software development companies in Bogota and in the 9th and 10th courses half of the Systems Engineering Faculty of the Universidad Sergio Arboleda, Bogota (Colombia), who developed grade projects.

The method consists of three very simple steps to develop software and is distinguished from others because at the end of each stage of its cycle, resulting in small boards that house all the information required at this stage and will serve as input to the next, always emphasizing quality.

This document embodies the result of the project, considering methodologies used in Bogota, Colombia.

Keywords: artifact, quality, medusa, method to develop software, software engineering

Introducción

En los albores del desarrollo de *software* (1945 - 1955) programar no era una trabajo como se conoce hoy. Esa actividad se consideraba un arte; no había planificación, cada programa se hacía según la necesidad, no existía dirección de proyectos y mucho menos un método que guiara el desarrollo. Pero con el correr del tiempo, aparecieron nuevas tecnologías y, por ende, nuevos programas para cubrir los requerimientos del usuario que reflejaron la necesidad de organizar el desarrollo de software para abarcar todos los entornos. De ahí que la industria del *software* desempeñe un papel determinante en la economía mundial. Para cubrir las necesidades del desarrollo de software aparecen metodologías y estándares de todo tipo, que guían a los programadores hacia el buen desarrollo.

Han surgido problemas que no se han podido superar del todo, en la falta de aplicación de esos métodos para desarrollar *software*. En unos casos, las empresas desarrolladoras de *software* crean o modifican, según su necesidad, alguna de las metodologías existentes; en otros, las siguen al pie de la letra (lo que según el concepto de algunos desarrolladores se torna muchas veces tedioso y muy largo). Pero en la mayoría de estas empresas no se sigue ningún método, sino que cada cual se inventa el propio; otras fusionan algún método que conocieron en la etapa estudiantil con su experiencia; o por el contrario, sólo lo desarrollan como lo han hecho siempre. Estas diversas formas de hacer *software*, llevan a que no exista uniformidad en el trabajo, lo que se ve reflejado en su calidad.

A partir de los problemas mencionados, han surgido nuevas metodologías para que, al aplicar la Ingeniería de *Software*, se permita enfocar el problema de manera íntegra y que el producto del trabajo tenga la calidad requerida. Pero abarcar todas las fases de desarrollo que cubren las diferentes metodologías ha traído como consecuencia que los gestores de proyectos reduzcan costos y plazos; esto, como es de esperarse, se refleja en el producto final, y se tiene como resultado un producto sin calidad. Esto deja ver la importancia de adoptar una disciplina de ingeniería para desarrollar *software* que sea sencilla y fácil de ejecutar.

Por lo anterior, y luego de indagar entre alguna población estudiantil universitaria y más de 50 desarrolladores profesionales de *software* de distintas empresas dedicadas a dicha labor, sobre los problemas que se les presentan, se realizó un método para este fin, denominado MEDUSA, el cual se elaboró con el objetivo de permitir el desarrollo de software de una manera sencilla y en este documento se explicará en qué consiste.

Estado del arte

Muchos han sido los autores que desean mejorar el proceso del *software* y todo lo que concierne a las tecnologías de ingeniería de *software*. Roger Pressman ha sido uno de ellos; durante muchos años ha permitido conocer a través de su libro *Ingeniería del software*, un enfoque práctico (Pressman, 2010), sobre las generalidades utilizadas en el proceso de *software*. En él orienta sobre cómo se deben seguir las actividades genéricas que establecen el marco de un trabajo como son comunicación, planeación, modelado, construcción, validación, despliegue y el esqueleto arquitectónico.

Libros clásicos, como el de los autores Kendall & Kendall, intentan guiar a quien lee su libro *Análisis y diseño de sistemas* (Kendall & Kendall, 2006) para fundamentar el análisis de sistemas, el análisis de requerimientos de información, que conduce a entender el proceso de análisis y diseño y a implementar el software. Otro clásico es el libro de Bernd Brugge y Allen H. Dutoit, *Object-Oriented Software Engineering: Using UML, Patterns and Java* (Bernd, *et al*., 2006) que explican cómo obtener requerimientos para un sistema que se va a construir; cómo realizar análisis y diseño del sistema; el diseño de objetos; la administración de la fundamentación; la configuración del *software*; las pruebas y la administración de un proyecto de *software*.

Quizá de los autores que no deben dejarse de lado, por cuanto han sido los grandes creadores de sus propios métodos son: Booch (2000) con su libro *Análisis de diseño orientado a objetos con aplicaciones*, donde propone su metodología y utiliza diagramas para describir las decisiones de análisis y diseño, tácticas y estratégicas que deben hacerse

cuando se crea un sistema orientado por objetos. Ivar Jacobson (2000) autor de seis libros best-seller, líder del pensamiento en el mundo del software; padre de los componentes y de la arquitectura de componentes, del desarrollo de software orientado a aspectos de la ingeniería moderna de negocios, junto a Bocch y Rumbaugh crearon el lenguaje unificado de Modelaje (UML - *Unified Modelling Language*) y del Proceso Unificado (UP – Unified Process). Por su parte, Jacobson desarrolló el método para proyectos orientados a objetos, denominado *Object Oriented Software Engineering* (OOSE – Ingeniería del software Orientada a Objetos), especialmente fuerte porque proporciona un buen soporte para los casos de uso como forma de dirigir la captura de requisitos, el análisis y el diseño de alto nivel; también creó el OMT-2, muy útil para el análisis de un sistema y para sistemas de información con gran volumen de datos.

Aunque Grady Booch (método Booch), Ivar Jacobson (método OOSE) y James Rumbaugh (método OMT) eran reconocidos mundialmente por sus métodos orientados a objetos para desarrollo de software, iniciaron en 1994 la formación de un nuevo método que está estandarizando mercados orientados a objetos. Al unirse y crear la metodología UML intentaron introducir mejoras en los métodos de cada uno, capturaron lecciones aprendidas y resolvieron problemas que los otros tres métodos no habían visto.

No puede dejar de mencionarse al autor Craig Larman (2000), quien en su libro *UML y Patrones*, permite aprender a realizar análisis y diseño orientado a objetos, pero dirige su libro a diseñadores con experiencia y a programadores en lenguajes orientados a objetos también con experiencia.

La propuesta que hizo Reenskaug (1996), con el Modelo-Vista-Controlador (MVC), expuesto en su libro *Working with objects The OOram Software Engineering Method*, utiliza la construcción de sistemas *software* con interfaz gráfica y tiene dos objetivos principales: uno, reducir la distancia entre el modelo mental y el modelo computacional; y otro, facilitar los cambios de la interfaz de usuario. Ambos objetivos están relacionados de una forma muy directa.

Bennet, *et al*. (2008) en su libro *Análisis y diseño orientado a objetos de sistemas*, desarrolla un enfoque interactivo e incremental basado en el proceso unificado estándar de la industria, sitúan el análisis y el diseño de sistemas en el contexto del ciclo de vida completo de los sistemas; y proponen un nuevo contenido sobre desarrollo basado en componentes, desarrollo de software Agile, Programación eXtreme, Arquitectura dirigida por modelo, y otros recientes avances en el proceso de desarrollo de *software*.

Lo expuesto hasta aquí está congregado en la ingeniería de *software*, disciplina de la informática compuesta por una serie de pasos que abarcan métodos, herramientas y procedimientos, a los que se denominan paradigmas de la ingeniería de *software* y que contribuyen a mantener *software* de calidad y proveen el control del proceso de su desarrollo. Algunos de sus postulados indican que existen tres elementos primordiales para lograr este cometido: los métodos, las herramientas y los procedimientos. Los métodos permiten construir *software* de forma técnica e incluyen formas para planificar, estimar, analizar, diseñar, codificar, probar y mantener *software*. Las herramientas pueden ser automáticas o semiautomáticas, utilizadas para apoyar el empleo de los métodos; es el caso de las herramientas CASE – Ingeniería de *Software* Asistida por Computador y los procedimientos; son los que marcan la secuencia en la aplicación de los métodos, como por ejemplo las entregas, los controles de calidad, las guías para evaluar el proyecto, entre otros.

Metodología

En el desarrollo de MEDUSA fue necesario realizar dos tipos de investigación, de una parte, la investigación básica que abarcó teorías y métodos existentes para el desarrollo de *software*, y a partir de las cuales se estableció una nueva forma para desarrollar *software* de manera ordenada, simplificada y clara. De otro lado, se tomó la investigación aplicada, que buscó consolidar el método resultante (MEDUSA), como modelo en los procesos de desarrollo de *software* dentro de la Universidad Sergio Arboleda y en aquellas empresas que lo quieran adoptar.

Descripción de la metodología

Fase 1. Fase exploratoria. Se realizó la consulta y selección cuidadosa de los autores, la bibliografía y

los materiales útiles que abordan las metodologías y métodos utilizados en proyectos de desarrollo de software.

Fase 2. Fase de descripción y explicación. Se revisaron, identificaron, clasificaron cronológicamente y seleccionaron las diferentes metodologías y métodos que deberían ser tenidos en cuenta en el desarrollo del proyecto.

Fase 3. Fase de encuestas. Se identificó la población objetivo para aplicar la encuesta, tanto en el sector de la industria del software de la ciudad de Bogotá, como en la comunidad académica de la Universidad Sergio Arboleda. Esta encuesta se orientó a conocer las dificultades que se presentan al emprender un proyecto de desarrollo de software. Se prepararon los instrumentos para su aplicación, teniendo en cuenta los aspectos propuestos en el problema de investigación y lo arrojado por el análisis de la Fase 2. Se aplicó el instrumento a la población objetivo, se realizó el análisis de los datos y, con base en los resultados obtenidos, se procedió a realizar un estudio detallado de las metodologías y los métodos más utilizados en las empresas encuestadas.

Fase 4. Fase de construcción de la propuesta. Con base en los resultados de las fases 2 y 3, se establecieron los lineamientos, pautas o criterios que sirvieron como soporte para construir del método para el desarrollo de *software*, que se convertiría en el resultado de la investigación. Se construyó la propuesta metodológica y se preparó el documento resultado con la propuesta para el desarrollo de proyectos de *software* a través de MEDUSA, que se ajusta a empresas de cualquier tamaño y sector económico, incluyendo el académico y particularmente a la Facultad de Ingeniería de Sistemas y Telecomunicaciones de la Universidad Sergio Arboleda.

Fase 5. Fase de documentación. Se preparó un documento general con la descripción del trabajo, la especificación de la metodología propuesta y el aporte que desde la línea de Desarrollo de *Software* se está difundiendo entre las empresas desarrolladoras de *software* y los estudiantes que quieran acoger o aceptar la metodología para realizar sus proyectos de software.

Fase 6. Fase de difusión. MEDUSA, como resultado de la investigación, se ha presentado en eventos académicos internos en la Universidad Sergio Arboleda, y se han escrito artículos sobre el método.

Resultados

En dónde inicia MEDUSA

Para la concepción de MEDUSA se hicieron tres tipos de análisis: en el primero, se realizó un barrido por más de 19 modelos de desarrollo de *software*, y entre ellos se escogieron los nueve más representativos; el segundo, se hizo con estudiantes de último año de la Facultad de Ingeniería de Sistemas y Telecomunicaciones de la Universidad Sergio Arboleda, con el fin de conocer las dificultades que se les presentan al enfrentarse a un proyecto de desarrollo de *software*; y el tercero se realizó con cincuenta desarrolladores profesionales, con el fin de identificar el conocimiento y uso de las diferentes metodologías para el desarrollo de *software*.

En la etapa de análisis de los diferentes métodos y modelos se seleccionaron aquellos que han tenido representación importante en el mundo del desarrollo del *software*, entre los que se encuentran: modelo en cascada, también llamado modelo lineal secuencial o ciclo de vida clásico; los modelos incremental, prototipos, espiral, modelo V, diente de Sierra, desarrollo concurrente, técnicas de 4ª generación, DRA, XP, paradigma orientado a objetos, desarrollo basado en componentes, desarrollo orientado a aspectos, modelo de métodos formales, proceso unificado, paradigma orientado a servicios y Microsoft Solution Framework (MSF). Luego de revisar los anteriores modelos, se descartaron aquellos que se consideraron menos relevantes, y quedaron nueve, con el fin de realizar un cuadro comparativo (los métodos escogidos para la comparación fueron: DRA, Prototipo, Espiral, XP, RUP, Microsoft Solution Framewrok (MSF)). Para ello se identificaron los elementos claves de comparación, con el fin de establecer cuál de los modelos tenía más fortalezas. (Ver tabla 1 y tabla 2).

Tabla 1. Modelos y características importantes

Modelo / Características	DRA	XP	RUP	MSF(Microsoft Solution Framewrok)	Espiral	Prototipo
¿Combinado con otro?	Si (Cascada, lineal secuencial)	Se basa en la simplicidad, la comunicación y la realimentación o reutilización del código desarrollado	RUP es una implementación del Desarrollo en espiral. Desarrollo iterativo. El ciclo de vida RUP es una implementación del Desarrollo en espiral.	El modelo de proceso MSF combina los mejores principios del modelo en cascada y del modelo en espiral.	SI (prototipos en las iteraciones y cascada en aspectos sistemáticos)	Prototipado Incremental
Ciclo de desarrollo	Corto	Planificación incremental e interactiva	Cada ciclo de iteración, se hace exigente el uso de artefactos	Estrategia iterativa	Corto	Corto
Enfoque de construcción Basado en	Componentes	iteraciones, evolutivo	Se caracteriza por ser iterativo e incremental, estar centrado en la arquitectura y guiado por los casos de uso	Se centra en modelos de proceso y de equipo, dejando en un segundo plano las elecciones tecnológicas.	Evolutivo e incremental, cíclico alrededor de un espiral. Combina los elementos controlados y sistemáticos del modelo de cascada con la filosofía interactiva de construcción de prototipos	Iterativo, evolutivo e incremental
Trabaja por	Escalas y equipos	Tasa de errores del producto final es más baja. Utiliza las pruebas unitarias como eje de todo desarrollo	Ciclos interactivos		Actividades cumplidas en ciclos	iteraciones y pertenece a los modelos evolutivos
Lo que más consume	Recurso humano, tanto desarrollador como clientes	Recurso humano y tiempo	Tiempo y costos			Tiempo y recurso humano
El cliente puede ver avances del proyecto	Sí	Sí		En el modelo de diseño de soluciones, los usuarios se ven involucrados	Sí	Sí
Tiempo para el desarrollo del Proyecto	Tiempo corto	Corto plazo	Largo plazo	Pequeños, micros, medianos y de largo plazo. Cualquier tecnología y cualquier dimensión	Corto	Largo por los ajustes que solicita el cliente en cada iteración.

El resultado de la comparación entre los métodos de desarrollo de software trajo como resultado, entre otros, que la mayor parte de las metodologías siguen un enfoque de construcción por iteracciones, ciclos secuenciales e incrementos; el 44% de modelos escogidos incluye el método Cascada y el 22% incluye dentro de sus fases el método Espiral, que se refiere al desarrollo de un proyecto; el 44% de los modelos comparados se dirige a desarrollar proyectos cortos, mientras que el 22% a proyectos de largo plazo; el 11% es para trabajar todo tipo de proyectos, mientras que el uso del 11% de los modelos depende de las tareas que constituyen una iteracción.

Analizando el tamaño de los proyectos, el 22% de los modelos estudiados son utilizados en proyectos pequeños, el 22% en proyectos grandes y el 22% en todo tipo de proyectos, lo que quiere decir que todos los modelos son utilizados sin preferencia en especial; se excluye el método cascada y DRA, que son utilizados casi en su totalidad para el desarrollo de proyectos pequeños. De otra parte, al analizar el consumo de recursos, el 44% lo hace en tiempo, el 22% en dinero y el 33% en recurso humano.

El 66.66% permite que los clientes puedan ver avances del proyecto. Esto se considera como positivo, debido a que es precisamente el cliente quien puede hacer la retroalimentación y exigir lo que se requiere para entregar un producto de calidad. Algunos de los métodos limitan la intervención del cliente a fases exclusivas, como el modelo MSF, en el que el usuario participa en el proceso de diseño, o el modelo incremental, en el cual el usuario participa en cada iteracción, o el de prototipos o espiral, en los cuales el usuario es el gran participante del proceso.

Las actividades que se siguen para el desarrollo de las metodologías escogidas varían en número y acciones, aunque predominan las fases comunes del proceso de desarrollo, a saber: análisis, diseño, desarrollo, implementación y pruebas. El uso de algunas metodologías es inapropiado cuando no se tienen perfectamente establecidos los requisitos; cuando no se cuenta con los suficientes recursos técnicos o el software depende en gran medida de otro software o del hardware. En otras metodologías debe preveerse la identificación de los riesgos técnicos. Cuando se inicia un proyecto de software, las metodologías se deben usar casi en su totalidad cuando los requisitos estén muy bien definidos y los procesos se puedan controlar.

El siguiente análisis fue realizado con estudiantes de último año de la Universidad Sergio Arboleda de la ciudad de Bogotá, Colombia. Se puede indicar, que las metodologías de desarrollo de software, que más conocen son, sin duda, los métodos tradicionales, como el método cascada, lineal o clásico y el RUP, con un 100%; el espiral y el orientado a objetos, con 93.21%; XP en 53.84%; el modelo de prototipos e incremental 69.23%; el modelo por etapas, en 61.54% y el modelo de técnicas 4G, en 23.08%.

Otro aporte importante fue el que resultó del análisis de datos de las empresas versus los estudiantes; se dio el reporte a la Decanatura del Programa con el objeto de tomar los correctivos necesarios, lo que condujo a que, aprovechando el proceso de autoevaluación del programa, se decidiera replantear el plan de estudios, incluyendo gran parte de las conclusiones generadas en la investigación.

Una conclusión relevante es que el sector de las empresas denominadas micro está un tanto desprotegido y se ve como un buen nicho para enfocar en el futuro el desarrollo de proyectos académicos. En cuanto al análisis relacionado con los desarrolladores profesionales, que prestan sus servicios en empresas bogotanas renombradas; se encontró que el 100% de estas empresas son de carácter privado, de las cuales 58.69% son empresas medianas, 4.34% micro empresas, 10.86% grandes empresas y 20.08% empresas catalogadas como pequeñas empresas.

En cuanto al sector al que se dirigen las empresas desarrolladoras de *software* para satisfacer necesidades, se evidenció que el 82.60% está orientado a las grandes empresas, el 41.30% a empresas medianas, el 17.39% a las pequeñas empresas y el 15.231% a las empresas micro. Aunque el portafolio de servicios de las empresas encuestadas para satisfacer necesidades del mercado es muy diverso, se distingue que el desarrollo de *software* a la mediana (73.91%) cubre gran parte de este mercado; el 45.65% se orienta a la consultoría, el 32.60% al mantenimiento de *software*, el 30.43% a la gerencia de proyectos de desarrollo de *software*, el 23.91% a la administración de IT y al

aseguramiento de calidad, el 21.73% a la inteligencia de negocios y sólo el 13.04% orienta sus servicios a la seguridad informática.

Los principales sectores a los que dirigen la actividad del desarrollo de *software* las empresas encuestadas son, en su orden: al sector bancario el 43.47%, al estatal un 39.13%; a la industria el 34.78%; al financiero el 32.60%; a los sectores salud y asegurador (empresas aseguradoras) un 17.39%, mientras que a los sectores educativo y de seguridad social sólo los cubren en un 15.21%. Es evidente que estos dos últimos sectores serán eventualmente tenidos en cuenta para orientar proyectos desde la academia, puesto que sus necesidades son muy grandes y la oferta puede tener una gran respuesta de su parte.

Se tuvieron en cuenta para este estudio elementos que, de una u otra manera, inciden en el rendimiento del *software*, como las plataformas, bases de datos y sistemas operativos, que arrojaron los siguientes datos: la plataforma que dirigen los desarrollos de *software*, la plataforma .NET es una de las más utilizadas, con un 69.56%; la J2EE, con un 65.21%, mientras que las plataformas, SAP, AS/400, S/390 y otras no registran gran utilización en el mercado.

En cuanto a las bases de datos, la herramienta Oracle, es utilizada para la gestión de bases de datos por la industria en un 71.73%; el SQL Server, lo usan el 60.86%, seguidos por MySQL, con un 39.13%, y PosgresSql, con 19.56%. Otros sistemas, como SyBase, DB2, Access, Informix, SQLite, Apache Derby, Paradox, tienen una representación muy baja en el desarrollo de software dentro de las personas encuestadas.

Con referencia a los Sistemas Operativos utilizados en las empresas, éstas se encuentran en su gran mayoría utilizando el Windows, con 82.60%, seguido por Linux, con 50% de uso. Contrario a lo que muchos ingenieros imaginan, los desarrolladores encuestados no utilizan el Unix sino en un 15.21%; esto, debido a que el Unix es un sistema operativo que se caracteriza por ser multiusuario y multitarea, utilizado principalmente en grandes empresas con sistemas de computo muy robustos y en la población estudiada sólo el 58.69%, de las empresas son medianas y únicamente el 10.86% son grandes empresas.

Al iniciar un proyecto de desarrollo de software en las empresas, sólo el 60.86% de los encuestados sabe qué metodología utilizar; el 32.60% no lo sabe, esto da un buen margen para promover el uso de MEDUSA entre esta población que pierde tiempo al considerar qué método seguir. Sólo el 50% de ellas, utiliza metodologías conocidas y el 41.30% tiene la propia, que han establecido de la fusión de otras conocidas, como por ejemplo: Xp y Rup. El porcentaje restante se dividen entre los que utilizan certificaciones ISO, los que emplean guías establecidas por la empresa y los que se rigen por lo que el cliente exija.

En cuanto a las metodologías utilizadas para el desarrollo de *software*, un 45.65% se guía por RUP, un 43.47% por el análisis y diseño orientado a objetos, un 15.21% por XP y el 2.17% por MSF. En cuanto a paradigmas de desarrollo de software gran parte de las personas encuestadas no los emplean (45.65%), pero un 41.30% sí lo hace; esto da a entender que gran parte de las empresas no tienen parámetros fijos, ni estándares de calidad establecidos para ese desarrollo. Algunas aplican el modelo de prototipos (11%), el modelo en espiral y desarrollo por etapas en un 8%, el modelo en cascada, el desarrollo iterativo y creciente, y el RAD (*Rapid Application Development*) en un 3%, lo que muestra que sin importar el tipo de desarrollo que se haga, adoptan métodos que, como en el de Cascada, deben ordenar rigurosamente las etapas del ciclo de vida del software, de manera que al iniciar cada una de ellas se debe esperar a que finalice la inmediatamente anterior. Algunos autores critican esto porque acarrea mayores costos y esfuerzos. Como se mencionó en este escrito, se ratifica con las cifras que RUP se usa de forma indiscriminada, pues se dirige a todo tipo de desarrollo, (micro, pequeños, medianos o grandes); no obstante, la misma metodología RUP aconseja que se utilice en grandes desarrollos.

Para el modelado de las aplicaciones, el UML es el más utilizado, pues un 82.60% de los encuestados modela el *software* por medio de dicho lenguaje, y sólo el 4.34% lo hace con Enterprise Architect y el 2.17% con Arquitectura Dirigida por Modelos.

Luego de analizar de forma general todo lo expuesto en este numeral, se decide realizar un método que facilite el desarrollo de software, y así nace MEDUSA.

En qué consiste MEDUSA

El origen del nombre del método MEDUSA es: **MÉ**todo de **D**esarrollo **U**niversidad **S**ergio **A**rboleda.

Los lineamientos tenidos en cuenta para construir el método, dicen que debe:

Ser sencillo (fácil de aprender, modificar y ejecutar), iterativo, incremental (hacer pequeñas entregas de *software*), cooperativo (clients y desarrolladores trabajarán juntos), adaptable (permite realizar cambios), tener calidad como eje transversal, construido por modelado de tablas, tener en cuenta las metodologías y paradigmas desarrolladas desde 1955, tener en cuenta los resultados obtenidos de las encuestas realizadas a 50 desarrolladores de diferentes empresas, enmarcarse dentro de la Teoría General de Sistemas, considerar tres estructuras básicas: la empresa, el *hardware* y el *software*, considerar para MEDUSA tres etapas: conceptualización y análisis, diseño, implementación y complementos. Estar compuesta por etapas y procesos, que a su vez tendrán actividades, compuestas por tareas y siempre darán como resultados entregables o artefactos. Considerar en la primera etapa la compresión de los procesos de la organización, con el fin de proseguir sin dificultades. Seguir el enfoque de prototipos, con el objeto de cumplir con lo que el usuario necesita y que los requisitos sean cumplidos a cabalidad. Realizar pruebas para cada uno de los componentes, antes de tener el sistema completo. Documentar solamente lo necesario. Utilizar las mejores prácticas al construir aplicaciones de este estilo, según estándares mundialmente aceptados.

Marco general de MEDUSA

El método está soportado en tres elementos indispensables para su correcto desarrollo; estos son, la empresa, el *hardware* y el *software*. La empresa, porque para el desarrollador de software es importante no perder de vista el modelo de estructura de la empresa y enterarse de cómo son las relaciones entre los distintos órganos que la componen, cómo es la delegación de autoridad del órgano supremo, cómo es en realidad el nivel jerárquico de la empresa, de esta manera poder diferenciar los roles y actividades que desempeñan los posibles usuarios del sistema. La estructura de *hardware* y *software* son esenciales para entender cómo se están realizando los procesos que la empresa está utilizando con algún *software*, revisar plataformas, dispositivos, almacenamiento. Conviene también revisar programas con los cuales el sistema va a interactuar, de esta forma, ir evaluando el software que haga falta y se deba adquirir.

Como se mencionó en la definición de los lineamientos para el desarrollo de MEDUSA, éste se enmarca dentro de la Teoría General de Sistemas (TGS), por cuanto, cada uno de los elementos que conforman el método se mira como un sistema o subsistema, en donde todos los componentes están interrelacionados para lograr el objetivo. Cada una de las tres etapas que conforman a MEDUSA consta de procesos (instancia del programa) que a su vez se conforman por actividades o agrupaciones de tareas (acciones u operaciones que hacen parte del proceso, apuntan al cumplimiento de las metas del proceso, y son una subdivisión de los objetivos, que permiten agrupar compromisos de forma lógica para establecer un orden y realizar seguimientos). También las actividades se componen de tareas (las actividades o acciones específicas que deben ejecutarse para obtener un resultado concreto, que pueden desagregarse para dividir eficientemente el trabajo, para aprovechar el tiempo. Por medio de las tareas se definirá el alcance del proyecto, para de responder al qué de la misión del proyecto). Todo lo anterior, considerando que la ingeniería de software es *"un conjunto de etapas parcialmente ordenadas con la intención de lograr un objetivo"* (Jacobson, 1999).

Etapas de desarrollo de MEDUSA

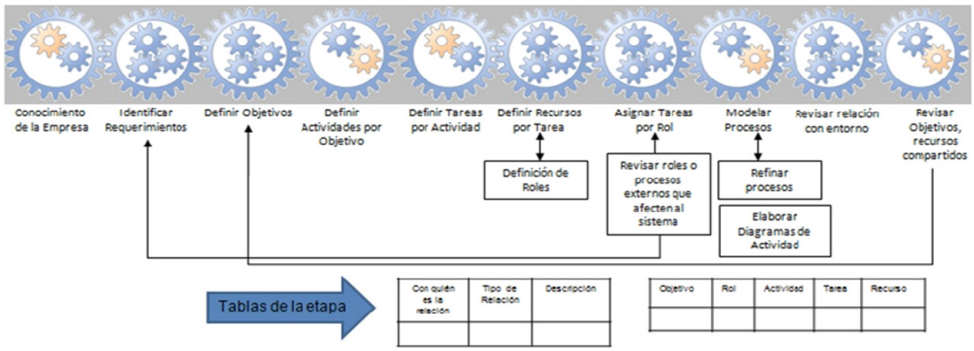
Para facilitar el control y la gestión de los proyectos que se desarrollen con MEDUSA, se concibieron tres etapas que constituirán al desarrollo eficiente del ciclo de vida o proceso de desarrollo de *software*, como comúnmente se denomina. Las tres etapas descritas a grandes rasgos son: la de conceptualización y análisis, la de diseño y la etapa de implementación y complementarios.

En la primera etapa denominada de conceptualización y análisis, el desarrollador o grupo de

desarrolladores, deberán: conocer la empresa; conocer a los empleados que trabajan con el sistema actual o los que van a trabajar con el sistema propuesto; revisar las relaciones de la empresa con el medio; enterarse del tipo de trabajo de los usuarios o los posibles usuarios del sistema e identificar las tareas que cada uno de ellos hace; definir objetivos y prioridades; conocer las necesidades de software para implementar u ordenar.

En las figura 1, puede observarse la estructura que rige la etapa de conceptualización y análisis.

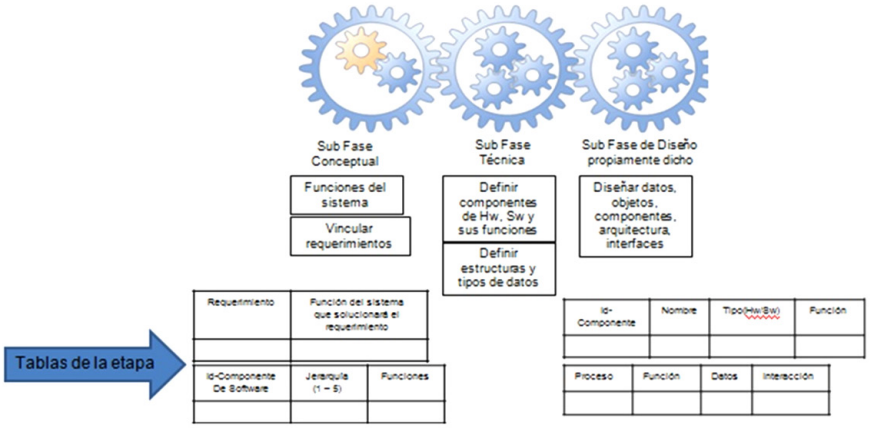
Figura 1. Etapa de conceptualización y análisis



En la etapa de diseño, se deberá: implementar cada uno de los requisitos encontrados en la etapa de conceptualización y análisis; diseñar el sistema de tal manera que éste sea comprensible por los *stakeholders*; no inventar en esta etapa nada que no se haya presupuestado en la etapa anterior; diseñar el sistema de manera uniforme; minimizar errores conceptuales; supervisar el proceso de construcción.

En la figura 2, se muestran paso a paso cada una de las actividades que componen la etapa de diseño de MEDUSA.

Figura 2. Etapa de diseño



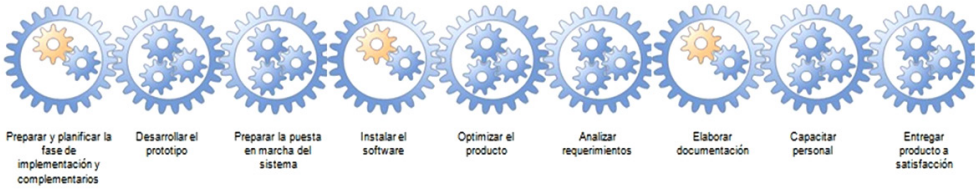
La etapa de implementación y complementarios, es una de las etapas denominada de trabajo colaborativo, por cuanto: desarrolladores y usuarios estarán más en contacto; se realiza el montaje del software desarrollado; se prueba el sistema con los tipos de prueba que existen; se realizan los ajustes a que dieren lugar; se imparte la instrucción para manejo del sistema; se realizan pruebas con el usuario o cliente; se escriben los manuales que requiera la empresa (de usuario, técnico, de instalación), se pueden presentar también,

documentos del desarrollo del software, documentos de la gestión del proyecto, documento con el modelado del software y documento de pruebas.

En la figura 3 se observan los elementos constitutivos de la etapa de implementación y complementarios.

En cada una de las etapas que componen MEDUSA, deben realizarse tres pasos importantes: Tener presente al cliente, hablando con él y revisando los pasos que se han dado; resaltar las tablas que deben elaborarse en cada paso y elaborar los artefactos que el método propone.

Figura 3. Etapa de implementación y complementarios



Discusión

MEDUSA es un método joven que se está probando para desarrollar proyectos de grado en la Universidad Sergio Arboleda. A los estudiantes se les presta asesoría y seguimiento con el fin de realizar un análisis y determinar el grado de dificultad de su aplicación. Terminada la aplicación, se procede a efectuar su validez a través del establecimiento de métricas, que permitan probar su complejidad y eficiencia. No hay estandarización de las métricas, y se construirán las propias. Una vez concluida la prueba, se podrá comprender la calidad del método, estimar la efectividad del proceso y mejorar la calidad del trabajo realizado.

Conclusiones

En este trabajo se ha delineado un método para desarrollar *software* con el objetivo de facilitar su construcción a programadores, sean estos estudiantes o profesionales. Si en su uso se comprueba la sencillez, adaptabilidad y facilidad para construir aplicaciones a través de la propuesta sobre la generación de tablas en cada fase que compone el método, propuestas para entender el sistema, se habrá logrado la meta concebida en esta investigación.

MEDUSA ha sido un proyecto que deja satisfacciones y expectativas, debido a que falta aún contrastar y verificar su aplicabilidad. Esta es apenas una primera etapa en la que se propone el método; para las pruebas se podrán realizar valoraciones matemáticas que ayuden a demostrar su eficiencia y eficacia, a través de la definición de métricas para su aplicabilidad. Con este método que se ha propuesto desarrollar *software* de una forma sencilla, y los que lo usen podrán, a criterio propio, valerse o no de un lenguaje de modelado, por cuanto las tablas que se proponen a través de las diferentes etapas que lo componen lo hacen comprensible.

Referencias

Bennet, S.; Farmer, R. & Mcrobb, S. (2008). Análisis y diseño orientado a objetos de sistemas. McGraw-Hill. Booch, G.; Rumbaugh, J. & Jacobson, I. (2008) El lenguaje unificado de modelado: guía del usuario. Madrid: Addison.

Guerrero, P. (2007). Elementos básicos de ingeniería de software. Instituto Tecnológico Metropolitano. Medellín, Colombia

Jacobson, R.; Booch, G. y Raumbaugh. (2000). El proceso unificado de desarrollo de software. Addison Wesley.

Larman, C. (2000). UML y Patrones, 2ª edición, México: Prentice Hall.

Lione, l. C. B.; Yvan, L. & Johanne, L. (2006). Toward the reverse engineering of UML sequence diagrams for distributed Java software, IEEE Transactions on Software Engineering. Vol. 32 N°.9, pp. 642-663.

Marcos, E. (2005). Investigación en ingeniería del software vs. Desarrollo software. Grupo KYBELE, Universidad Rey Juan Carlos.

Marius. M.; Arie, V. D. & Leon, M. (2007). Identifying crosscutting concerns using fan-in analysis, ACM, Transactions on Software Engineering and Methodology (TOSEM). Vol.17, N°. 1, pp.1-37.

Pressman, R.S. (2010). Ingeniería del software, un enfoque práctico. 7ª edición, México: McGraw-Hill.

Reenskaug, T. y Wold, P. Lehne, O.A. (1996). Working with objects The OOram Software Engineering Method. Taskon, Gaustadalléen

Software Engineering Institute (SEI). http://www.sei.cmu.edu/publications/documents

Sommerville, I. (2005). Ingeniería del software. 7ª edición, España: Pearson-Addison Wesley.

Weitzenfeld, A. (2008). Ingeniería de software orientada a objetos con Uml, java e internet, México: Thomson.

Sobre la autora

Lucy Nohemy Medina Velandia

Ingeniera de Sistemas. Especialista en Pedagogía. Magíster en Ingeniería de Sistemas. Estudio doctoral: Ingeniería Informática – Ingeniería de Software. Docente investigador en la Universidad Sergio Arboleda, Bogotá, Colombia. Calle 74 No. 14. lunome@gmail.com • lucy.medina@usa.edu.co