Modelo para el desarrollo rápido en la plataforma Java EE 6 para aplicaciones empresariales en la nube.

Marcos Adrián Jara Rodríguez
Facultad Politécnica, Universidad Nacional del Este.
Ciudad del Este, Paraguay.
marcosjara@yahoo.es

Resumen

Debido al auge actual de la Computación en la Nube, muchas personas y empresas van cambiando su percepción en cuanto al uso del software. Este trabajo de investigación está enfocado a ofrecer alguna utilidad a las empresas de la región de Ciudad del Este y alrededores que se dedican al desarrollo de software, ya que el mismo presenta un modelo para desarrollo de software, que permite la construcción rápida de aplicaciones corporativas en la nube (SaaS), utilizando la Tecnología Java EE 6, basada en las metodologías de desarrollos ágiles. Con esto se pretende lograr una disminución en el tiempo de desarrollo y puesta en marcha de las soluciones, minimizando los costos de implementación y aumentando la rentabilidad.
Descriptores: computación en la nube, SaaS, desarrollo ágil.

Abstract

Due to the current rise of Cloud Computing, many people and companies are changing their perception about software usage. This research aims at offering some usefulness for businesses in Ciudad del Este and surrounding region, engaged in software development; it presents a model for software development, which allows rapid construction of enterprise cloud applications (SaaS), using Java EE 6 technology, based on agile development methodologies. It pretends to reduce development and implementation time of solutions, minimizing deployment costs and increasing profitability.
Keywords: cloud computing, SaaS, agile development.

1. Introducción.

El presente proyecto presenta un modelo para el desarrollo de software de peso ligero (liviano), que permite la construcción rápida de aplicaciones corporativas en la nube (SaaS), utilizando la Tecnología Java EE 6, basada en las metodologías de desarrollos ágiles de más auge en la actualidad.

El objetivo del desarrollo de la metodología consiste en lograr una disminución en el tiempo de desarrollo, minimizando los costos que derivan de largos periodos de implementación aumentando así la rentabilidad [1].

Como resultado de la elaboración del modelo propuesto se obtuvo una solución fundamentada en 3 (tres) pilares para el desarrollo rápido: (a) *la generación automática de código inicial*, (b) *la utilización de plantillas predefinidas* y (c) *la ejecución de scripts automáticos*; que según la experiencia del autor y varias otras fuentes bibliográficas, son la base para el desarrollo rápido optimizando costo, tiempo y recursos.

2. Objetivos.

2.1. Objetivo general.

Realizar el diseño de una metodología de desarrollo rápido de software con enfoque en procesos ágiles utilizando el estándar Java EE para su ejecución en la nube.

2.2. Objetivos específicos.

- Investigar las tendencias de las aplicaciones en la nube así como el estudio de las diversas acciones realizadas en las diferentes fases de análisis, construcción, pruebas, despliegue y mantenimiento.
- Identificar las características principales de las aplicaciones Web en la nube, de forma a estandarizarlo como patrones y obtener el beneficio de la reutilización.
- Estudiar las herramientas Java EE disponibles de forma a identificar cuál de ellas, sola o en conjunto, se perfilan hacia el objetivo de obtener el producto final de forma rápida,

teniendo en cuenta las diferentes etapas de desarrollo, de forma a exponerla como sugerencia en esta propuesta.

– Exponer un conjunto de normas y procedimientos en base a herramientas y metodologías ágiles con el fin de que tanto desarrolladores como empresas puedan seguirla para agilizar su proceso de desarrollo abaratando los costos y asegurando la calidad.

3. Los pilares del desarrollo rápido.

Surgió la necesidad de atacar los puntos débiles que reducen el tiempo de desarrollo desde diferentes frentes, lo que conllevó al surgimiento de 3 elementos principales que se convirtieron en pilares.

De esta forma, la metodología propuesta está enfocada en 3 pilares fundamentales, que se presentan en la figura 1.

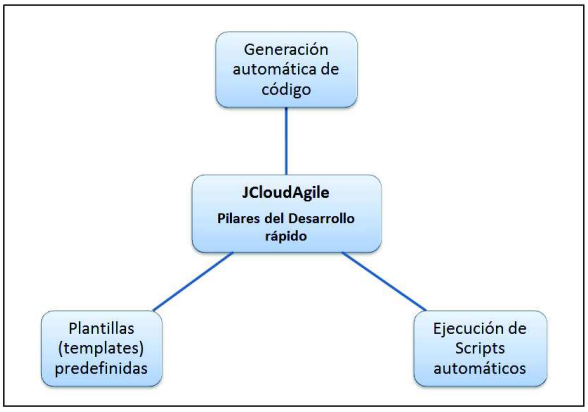

Figura 1. Pilares del Desarrollo rápido (Fuente propia).

3.1. Pilar Nº 1. Generación automática de código inicial.

El uso de herramientas de generación automática de código permite agilizar el desarrollo de sistemas de software e incrementar su confiabilidad [27].

Según el autor del libro *Code Generation in Action* de Jack Herrington [27], el código fuente manual debe ser respetado, pero rechazado. Se debe respetar porque hay casos especiales en el código que requieren una lógica específica realmente compleja o fuera de lo común. Cuando se reemplaza el código manual por código generado, es necesario estar seguro de haber contemplado esos casos especiales. En cambio se debe rechazar el código manual cuando fuere posible porque el tiempo de desarrollo es extremadamente valioso, y gastarlo en tareas repetitivas es casi criminal.

La autogeneración de código es un tema que muchas veces se encuentra lejos de los dominios de cualquier empresa de desarrollo. Muchas empresas no se introducen al tema, considerando la errónea idea de que trabajar por ello es muy complejo y costoso o que los hace perder mucho tiempo.

3.2. Pilar Nº 2. La utilización de plantillas predefinidas.

Según va aumentando la complejidad de los programas, así como de los problemas a los que se enfrenta el equipo de desarrollo, se van también identificando algunos elementos cuya estructura y comportamiento se repiten una y otra vez, como así también las soluciones que se adoptan para su resolución.

Las plantillas probablemente sean los elementos más presentes hoy en día en la creación de aplicaciones [14]. Son utilizados en las diferentes etapas de desarrollo y se encuentran en diferentes formatos y a veces con diferentes identificaciones, como por ejemplo: templates, patrones, estándares, modelos, componentes o clases [28].

Nótese que también un comportamiento bien definido al ser encapsulado en una función o procedimiento, puede servir como plantilla o modelo, para otros casos similares donde se requiera de la misma acción. Lo mismo ocurre con las actividades de las personas y roles que al cumplir siempre una misma rutina, al escribirlas y documentarlas también se convierten en un tipo específico de plantillas [29].

En [28, 30], se describen algunas plantillas genéricas realizadas para cubrir problemas específicos de Casos de Uso del Tipo ABMC (Altas, Bajas, Modificaciones y Consultas) y cómo el resultado del uso de estas plantillas pasa a servir de materia prima para las plantillas de la siguiente etapa del proyecto.

El uso de plantillas dará la posibilidad de que los procesos y actividades que se ejecutan en cada fase, sean más estándares y claros para todos los miembros del equipo y más eficientes en la hora de ejecutarlos.

3.3. Pilar Nº 3. Ejecución de scripts automáticos.

La utilización de scripts de automatización de tareas ha sido siempre una de las más poderosas herramientas que disponen los equipos de desarrollo a lo largo de la historia. El enfoque en el que se basa este pilar tiene estrecha relación con los dos pilares anteriores, y de cierta forma es aquí donde la aplicación de plantillas a y la auto-generación se encuentran para dar seguridad y firmeza a los proyectos desarrollados en la empresa.

Muy a menudo se tiene la idea que la utilización de scripts es solamente necesaria, o al menos más utilizada en la hora del despliegue del Siste-

ma. Según [14] los scripts de despliegue son necesarios para la instalación del producto sirviendo para tareas de empaquetamiento, versionado, compilación. Pero según [22] los scripts deben ser creados para prácticamente todo, tratando de automatizar lo máximo de tareas que se puedan y evitando las acciones innecesarias.

En la actualidad existen herramientas muy flexibles adaptables a cada situación, que permiten prácticamente diseñar los scripts, de acuerdo a cada necesidad específica.

La creación y utilización de scripts, van más allá de su apoyo en el marco de la construcción del *software* ya que proyectan también para la solución de innúmeras necesidades que surgen en la etapa de construcción y mantenimiento o de ejecución de sistemas en la nube.

4. Enfoque hacia la automatización.

Los enfoques metodológicos, principalmente aquellos los basados en *Xp* y *Scrum* [14, 24, 32], aprovechan del concepto de la automatización sólo para algunos usos concretos, como por ejemplo, la ejecución automática de scripts solamente se considera para el proceso de pruebas y el de las automatizaciones para el despliegue.

En la metodología aquí propuesta tiene un enfoque muy orientado hacia la automatización a través de la generación y el uso de las herramientas en las diferentes etapas del desarrollo.

5. De lo prescriptivo a lo adaptativo.

Las metodologías pueden compararse viendo cuántas reglas proporcionan.

En el Libro *Kanban y Struts - Obteniendo lo mejor de ambos* [32], se define claramente que prescriptivo significa "más reglas a seguir" y adaptativo significa "menos reglas a seguir". 100 % prescriptivo significa que no te deja usar el cerebro, hay una regla para todo. 100 % adaptativo significan "Haz Lo que Sea", no hay ninguna regla o restricción. Como se puede ver, los dos extremos de la escala son de alguna manera ridículos.

Los métodos ágiles se denominan a veces métodos ligeros, en concreto, porque son menos restrictivos que los métodos tradicionales.

6. Visión de producto y mercado.

Es importante destacar que esta propuesta contempla un apartado breve pero complementario sobre la estructura organizacional mínima relacionada al modelo de negocios de una empresa de desarrollo de *software* con enfoque SaaS, que ayuda a tener clara la visión del producto y mercado con el cual se pretende interactuar.

En la mayoría de los casos el tratamiento dado a la estructura organizacional de una empresa de desarrollo no se incluyen como parte de la metodología en sí, ya que solamente se tienen en cuenta el ciclo de vida relacionado al producto.

Los estudios realizados sobre la organización de las empresas de desarrollo dejan a relucir una realidad donde se observa que por lo general la mayoría de las que inician sus actividades se enfocan directamente en el producto que será desarrollado, prestando poca o nada de atención a su relación con el mercado [33], es decir cómo la empresa será estructurada, cómo atraerán clientes, qué canales de distribución tendrán o cómo será realizado el estudio de mercado y las ventas [34].

7. Modelo de una empresa con enfoque SaaS.

Una empresa que nace con el objetivo de desarrollar software, pasará por diferentes etapas hasta convertirse en adulta, donde estará posicionada para ganar la confianza de los clientes y del mercado.

Antes de planear un proyecto se deberían establecer los objetivos y el ámbito del producto [18], considerar soluciones alternativas e identificar.

Según [14], la organización inicial de las empresas de software suele realizarse al azar esperando que las situaciones vayan apareciendo y respondiendo bajo demanda según ciertas circunstancias. Muchas iniciarán sin tener claro su rol en el mercado [35].

En la figura 2 se presenta una propuesta interesante extraída del modelo Lean Canvas, que va a ayudar a lograr éste objetivo.

Figura 2. Bloques del modelo Lean Canvas [16].

La propuesta para el modelo de negocio de ejemplo posee recursos y elementos del Lean Startup, cuyos fundamentos están orientados al desarrollo rápido de empresas de tecnologías [16], realizándose las adaptaciones requeridas para una

empresa de desarrollo de sistemas con enfoque SaaS, teniendo en cuenta los tipos de procesos y actividades que se requieren.

Es un modelo simple y práctico de elaborar como el que se ejemplifica en la figura 3 siguiente:

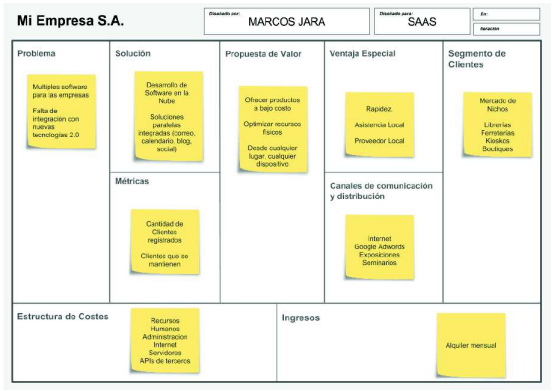

Figura 3. Modelo contextual de la idea del producto (Fuente propia).

8. Ciclo iterativo para el desarrollo del producto.

En la figura 4 se puede observar el ciclo constante por el cual debe atravesar un producto teniendo en cuenta las iteraciones para su desarrollo, desde que el mismo se inicia.

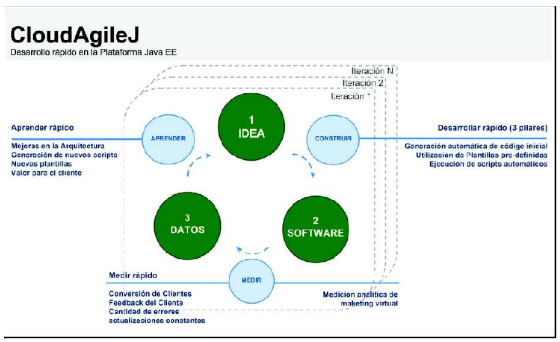

Figura 4. Ciclo iterativo para el desarrollo del producto (Fuente propia).

En la figura 5 se desarrolla el nodo construir, donde se presenta una leve adaptación al modelo Lean, siendo su diferencia principal el uso de los 3 pilares presentados en apartados anteriores, en este caso para lograr el desarrollo rápido (ágil).

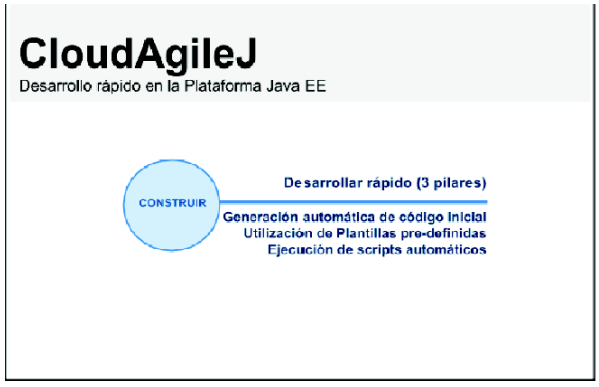

Figura 5. Nodo construir basado en 3 pilares (Fuente propia).

La principal característica del modelo presentado consiste en un circuito que gira en torno a un mismo conjunto de conceptos (idea, *software* y datos) que se van iterando de forma indefinida, con lo que se logra en primera instancia implantar el software en la nube, pero que no termina de iterarse, ya que el mismo concepto será utilizado también para las continuas actualizaciones y mejoras que se vayan realizando.

9. Ciclos breves para la obtención de cada elemento.

La idea.

Cuando se va a desarrollar un *software*, lo primero que debe quedar bien claro es el resultado del analisis. Generalmente cuando se inicia el desarrollo de un nuevo sistema la constumbre es realizar un relevamiento de datos y un posterior análisis de los mismos, con el concepto de la idea lo que se pretende es iniciar el trabajo mucho antes. Definir concretamente el problema y la solución, el segmento del cliente a quien va dirigido el producto y los canales para llegar a ellos así como encontrar una ventaja competitiva y la propuesta de valor son fundamentales en este concepto y debe ser realizado en cada momento que se pasa por el, en los ciclos.

La construcción.

Es el nodo más complejo y el que se tratará de ágilizar lo máximo que se pueda. Se trata en esta parte de contar con herramientas que hagan el trabajo estandar, como ya se mencionó en el pilar número 1 - Generación automática de código inicia.

Se debe tener desarrollado las plantillas para los tipos de formularios o programas que se van a utilizar, en un primer momento, si no se dispone de profesionales, o de mucho tiempo, estas plantillas pueden ser elaboradas en simples archivos de textos, teniendo como premisa de que para el que lo vaya a utilizar pueda ser sencillo buscar y reemplazar nombres o variables.

10. Herramientas de scripts.

En el mercado existen una serie de herramientas que posibilitan la generación de scripts, una de ellas y la más utilizada en la actualidad para proyectos Java y Java EE es el Ant de la fundación Apache.

Pero el Ant, no es la única herramienta de la que dispondrá el programador o la empresa para crear scripts o rutinas automatizadas, ya que en este contexto también se pueden citar a Maven, que es otra de las promesas en cuanto a necesidades de automatización y gestión de dependencias se refiera.

La empresa que desee crear software ágil, debe necesariamente conocer y hacer uso y abuso de estas herramientas, explotarlas al máximo para sacarle partido, ya que son herramientas disponibles y no tienen ningun costo por ser *software* libre.

11. Generación automática de código.

El líder del proyecto debe tener la visión sobre el re-aprovechamiento y optimización de actividades, de la misma manera como el desarrollador maneja los conceptos de reutilización código.

En ese sentido se deben evitar realizar actividades repetitivas como por ejemplo la implementación de funcionalidades mecánicas que ya todo el mundo sabe cómo realizarlas y que ya no requiere de la necesidad del programador para pensar por el hecho de que la forma de hacer es prácticamente automática, como copiar y pegar, luego cambiarlos de nombres.

Debe tenerse claro los objetivos y las metas de la empresa, para que las actividades de creación de herramientas no solapen a las actividades habituales de desarrollo, pero también deben ser establecidas metas claras en relación al tiempo de la puesta en producción de estas herramientas que se van a desarrollar, como ya se ha dicho lo ideal es que la maduración se de a los 12 meses siguientes de iniciación de actividades.

De esta forma, al cabo de un año, la empresa ya contará con una buena base de datos de rutinas automatizas y generación automática de código que puede ir empequetándolas o integrándolas para que finalmente con una sola orden puedan producirse funcionalidades más complejas de una sola vez.

12. Fase de desarrollo de un sistema.

En cuando a las fases propias del desarrollo de un sistema, la propuesta es simple y contiene elementos tomados de modelos ágiles donde se sugiere el viejo estilo de fases clásico, que consta de un momento inicial de la construcción la cual se denomina definición de funcionalidades, seguidamente de una fase de desarrollo propiamente, donde realmente se requiere de agilidad, posteriormente la fase de pruebas y finalizando con la fase de despliegue.

A continuación en la siguiente figura 6 se puede observar el modelo de fases propuesto:

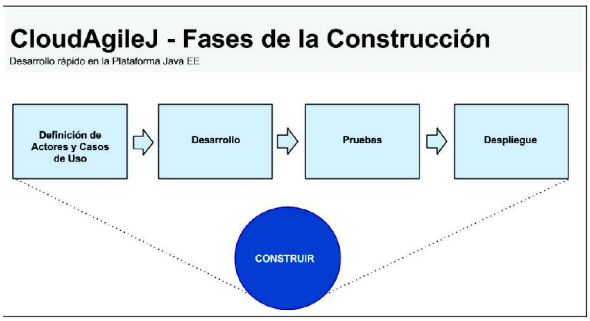

Figura 6. Fases del desarrollo de un sistema (Fuente propia).

Las fases del desarrollo ágil de un sistema son la clave para el desarrollo rápido de aplicaciones, a continuación se describen cada una de ellas:

Definición de actores y casos de uso: Para éste modelo se trata de simplificar todo lo que sea posible en cuanto a las tareas de esta fase, ya que se sugieren acciones livianas que aceleran la asimilación de la comprensión conceptual de lo que debe ser desarrollado.

Esta fase está dividida en dos partes que tienen el objetivo de obtener dos tipos de elementos principales, los actores y los casos de uso.

Para listar los actores del sistema, puede utilizarse como ejemplo el contenido de la tabla 1.

Tabla 1. Definición de los actores (Fuente propia).

Lista de actores Iteración: N Fecha: dd/mm/aaaa		
Nro.	Nombre del Actor	Tipo
1	Secretaria	Otro sistema mediante una API
2	Gerente	Otro sistema mediante protocolo
3	Administrador	Usuario por medio de una inferfaz
N		–

Seguidamente se tiene la segunda parte, la cual consiste en especificar los casos de uso del sistema. En cada iteración, los casos de uso del producto se van refinando hasta acercarse cada vez más a la funcionalidad real.

Las funcionalidades que se vayan a especificar en la primera iteración deben estar compuestas de las funciones mínimas que el cliente más lo establece como prioritario dentro de sus necesidades.

Es importante hacer constar un listado de las funcionalidades mínimas acompañada de un prototipo inicial que explique al usuario su finalidad y objetivo, así como la interacción que ésta tendrá con él.

En la tabla 2 se muestra una plantilla sencilla que servirá para este fin:

Tabla 2. Definición de la lista de Casos de uso (Fuente propia).

Lista Casos Uso Iteración: N Fecha: dd/mm/aaaa			
Nro.	Descripción de Casos de Uso	Complejidad	Prioridad
1		3 - Difícil	3 - Alto
2		2 - Mediano	2 - Medio
3		1 - Simple	3 - Alto
N		2 - Medio	3 - Alto

Ya en una siguiente iteración, las funcionalidades pueden aumentar, disminuir, modificarse o cambiar drásticamente. Se debe estar abierto a ello, pues no debería tener problemas para realizar cambios ya que se disponen de las herramientas para hacerlo.

Recuerde que cada ítem de la lista de funcionalidades debe estar acompañado de un prototipo o modelo de pantalla.

Esta fase requerirá más dedicación del equipo de trabajo y del cliente, pues esta fase no puede ser automatizada.

Desarrollo: El desarrollo consiste en la construcción del producto funcional para el usuario, en el software tangible, en el menos tiempo posible, utilizando para ellos herramientas desarrolladas dentro de la empresa, para la empresa.

Para lograr una rápida producción del producto mínimo funcional, se debe hacer uso del autogenerador de código, el cual es uno de los pilares para el desarrollo rápido, tratando de re crear los prototipos hechos en papel, en el producto *software*.

Pruebas: la fase de prueba es otra de las fases, al igual que el de desarrollo, donde se deben automatizar lo máximo que se pueda. Pruebas unitarias y funcionales pueden realizarse a través de la ejecución de scripts automáticos.

Es importante también en esta fase, montar una infraestructura de integración continua, de manera que todos los cambios que vayan siendo realizados y confirmados por los desarrolladores, lance un evento de generación y empaquetado de código donde se arrojen como resultados las pruebas realizadas y un resumen de las funcionalidades que se compone en el módulo.

Despliegue: Finalmente la fase de despliegue, que consiste en la disponibilización del producto terminado en la nube, listo para ser accedido por el equipo de pruebas para realizar los tests necesarios como así también para el cliente final, de manera que éste pueda hacer una revisión de las funcionalidades.

Esta fase también posee un alto grado de automatización, ya que hoy en día existen herramientas disponibles o para desarrolladores que permiten el acceso a servidores remotos, logrando realizar tareas desde la ubicación local, para parar o iniciar servidores, enviar y recibir archivos, ejecutar comandos, realizar backups y tareas programadas, todo sin la intervención humana sino que con la simples llamada a un comando.

13. Roles.

Conviene asignar roles a los profesionales de acuerdo al perfil determinado que éste posee [14], pero un profesional no debe cerrarse a desempeñarse siempre en el mismo rol, sino que deben existir reglas claras para que pueda darse una rotación de roles entre las personas e incluso entre equipos.

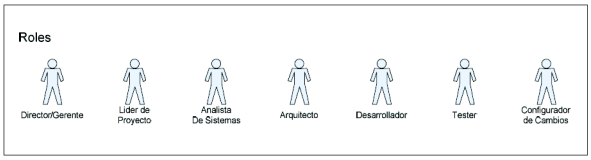

Figura 7. Roles sugeridos por el modelo (Fuente propia).

En la figura 7 se puede observar que en total son 7 los roles que sugieren esta propuesta para lograr la mejor sincronía entre las actividades que se desarrollan y a continuación se presenta un compendio relacionado a los roles que serán propuestos por esta metodología.

13.1. Rol de Director/Gerente.

Es importante la figura del Gerente o de Director dentro de una Empresa que desarrolla Software. La persona que cubrirá este rol debe poseer un perfil más comercial que técnico, pero también debe conocer bien lo concerniente a la parte técnica para que su labor no se vuelva algo superfluo para sus contactos tanto internos como externos a la organización. Este Rol representa a la persona vista como el primer contacto con el cliente que está necesitando de la solución de *software*, aunque también debe intermediar en la solución de problemas de los sistemas ya existentes. Es la cabeza visible de la organización o empresa.

13.2. Rol de Líder de Proyecto.

El líder del proyecto es el responsable por la ejecución y acompañamiento del proyecto a su cargo, realiza las tareas de organización y control del proyecto en sus diferentes etapas y fases, prevé situaciones venideras reflejos de la gestión y/o trabajo del Equipo bajo su cargo.

13.3. Rol de Analista.

La persona o personas que se encuentren bajo este rol deben saber todo acerca del funcionamiento de la empresa de sus clientes, realizando para ello las etapas primordiales del análisis de sistemas en sus diferentes fases de relevamiento, análisis y diseño del sistema, es el que conoce a fondo la lógica de negocios y constantemente se encuentra interrogando sobre la mejor forma de llevar a cabo los procesos para obtener mejores y más rápidos resultados.

13.4. Rol de Arquitecto.

El Rol de Arquitecto está reservado para el erudito en las tecnologías que son utilizadas por la empresa y para aquella que está muy imbuida en las innovadoras tecnologías que están por llegar, se mantiene siempre a la vanguardia ante los cambios y se prepara para lo que ha de venir, tratando siempre de implementarlas en la empresa.

13.5. Rol de Desarrollador.

Como se sabe, un proyecto de sistemas no puede realizarse sin un programador que lo desarrolle, la persona o personas que cumplan este rol serán las encargadas de llevar en frente la construcción del sistema.

13.6. Rol de Tester.

Mientras los mismos desarrolladores se encargan de la automatización de las pruebas unitarias, los testers se encargan de realizar las pruebas funcionales a niveles más genéricos, por ejemplo, probando e integrando las diferentes funcionalidades del sistema de manera a lograr un cierto grado de calidad en los productos construidos así como también para lograr el máximo acercamiento sobre los requerimientos solicitados por el cliente.

13.7. Rol de Gerente de Configuración y Cambios.

Se puede decir sin lugar a dudas que éste es uno de los roles indispensables requeridos para el desarrollo de los sistemas en la nube, ya que la persona dedicada a este rol debe velar por que el sistema se encuentre correctamente actualizado tras una nueva versión disponible, y evitar de que el sistema caiga ante cualquier inconveniente.

13.8. Integración de los diferentes roles.

Son varios los roles que intervienen en el proceso de desarrollo de un sistema informático. La manera ideal de llevar a cabo una implementación sería que cada rol sea cubierto por un individuo en particular, e incluso existen roles que deberían ser cubiertos por más de una sola persona, como es el caso del analista, arquitecto y desarrollador dependiendo de la cantidad de proyectos que se están ejecutando.

Ahora bien, si lo que se busca es utilizar una metodología ágil, abarantando los costos de desarrollo y aumentando la productividad, no se puede recomendar el inicio de las actividades de la empresa ya contando con 7 profesionales, uno por cada rol, ya que muy probablemente esto demandará mucha inversión al empezar.

Como es normal en toda organización, lo ideal es ir cubriendo los recursos de a poco y de forma gradual de acuerdo a la demanda de los servicios, para lo cual se debe prestar atención en cubrir todos los roles independientemente de si no se cuenta aún con todos las personas suficientes.

14. Magnitud del software.

Las metodologías ágiles en general mencionan que no es conveniente realizar una estimación del proyecto por completo antes de iniciarlo, ni que tampoco se podría saber la dimensión real hasta que el proyecto no estuviera completamente concluido.

Lastimosamente en cualquier proyecto, casi siempre es necesario conocer de ante mano la duración estimada del mismo de forma a poder estimar también el costo que demandará su desarrollo. La estimación no es una restricción fija sino simplemente una aproximación inicial [9] y gradual. En fin, obtener de alguna forma la dimensión del sistema en alguna unidad de medida estándar que ayude a definir los parámetros de tiempo y costo para los proyectos que serán elaborados es de suma importancia.

14.1. Factores que influyen en la estimación de proyectos de software.

La estimación y planificación de cualquier proyecto es por lo general una tarea compleja. Cada proyecto viene acompañado de sus propios detalles, compromisos, equipos, tiempos, costos, problemas y riesgos.

Es importante tener en cuenta que el trabajo realizado en esta etapa para obtener una unidad de medida del software, se basa únicamente en cuanto a las funcionalidades de implementación requeridas para el proyecto, principalmente en cuanto a los requisitos funcionales y no funcionales a ser implementados en los casos de uso.

Para realizar una estimación del *software* que esté acorde a éste modelo se hace uso de una aproximación a la metodología de estimación de puntos por casos de uso o *Use Case Point* (UCP) por sus siglas en inglés, cuyo fundamento se expone brevemente en el siguiente apartado.

14.2. Estimación de puntos por casos de uso.

El método de puntos en casos de uso es un enfoque bien documentado para estimar las actividades de desarrollo de *software* [38, 39]. Sin embargo, ningún método de estimación se debe usar de manera aislada, sino que se lo debe equilibrar con otros métodos.

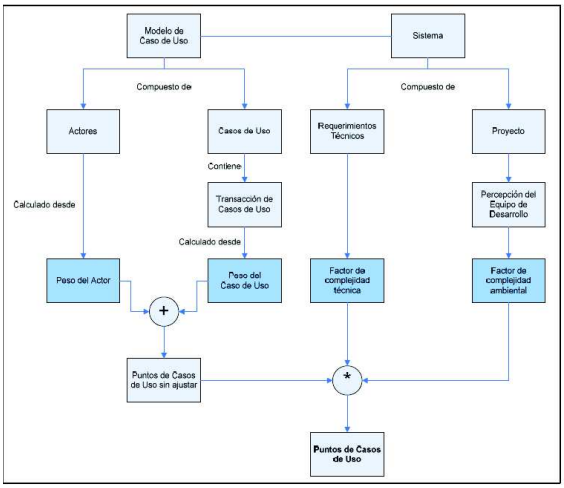

Figura 8. Puntos de casos de uso (Fuente propia en base a [38]).

La figura 8 presenta sus fundamentos principales. En la parte superior izquierda se puede observar los actores y casos de uso. El número y el peso de los casos de uso identificados representan el componente más importante para el cálculo de los llamados puntos de caso de uso sin ajustar. El tamaño de un sistema se calcula a partir de los puntos de caso de uso sin ajustar, ajustándolos según el factor de complejidad técnico obtenido tras considerar las propiedades técnicas del sistema y el factor ambiental donde se tienen en cuenta factores externos que afectan al desarrollo del sistema.

El peso de un caso de uso está dado por el número de transacciones de casos de uso diferentes en la interacción entre el actor y el sistema que se ha de crear.

Según el método de puntos en casos de uso [39], los criterios para asignar peso a un determinado caso según la cantidad de transacciones son:

- Caso de uso simple - de 1 a 3 transacciones, peso = 5.
- Caso de uso medio - de 4 a 7 transacciones, peso = 10.
- Caso de uso complejo - más de 7 transacciones, peso = 15.

Por consiguiente, las suposiciones sobre la naturaleza de una transacción y la estrategia usada para contar las transacciones influyen notoriamente sobre la estimación.

14.3. Transacción de casos de uso.

Las transacciones de casos de uso ayudan a determinar la longitud y concisión que generalmente se asigna a los casos de uso, de manera a poder obtener una cantidad estimada de las actividades que se tiene que realizar.

El concepto de transacciones dependerá de lo que el equipo de desarrollo considere relevante como actividad para el caso de uso, y una vez definido debe ser utilizado en conjunto con la planilla de casos de uso para estimar una cantidad. En la bibliografía existen variaciones que no dejan bien claro su concepto, pero Ivar Jacobson, inventor del caso de uso, describe una transacción de caso de uso como un "viaje de ida y vuelta" que va desde el usuario hasta el sistema para luego volver al usuario; una transacción está terminada cuando el sistema espera un nuevo estímulo de entrada y una vez recibida, la responde de vuelta al usuario [38].

14.4. Ecuación del punto por caso de uso.

Es importante tener en cuenta que la transacción de caso de uso es sólo una de la variable que se tiene en cuenta para determinar la magnitud, a continuación se citan otras variables que entran en juego:

- Factor de complejidad técnica.
- Factor de medio ambiente.
- Factor de Productividad.

Todos estos factores que representan el peso se ven reflejados en la siguiente ecuación:

$$\mathbf{UCP} = UUCP \times TCF \times ECF \qquad (1)$$

Donde:

- **UUCP** = Puntos de caso de uso sin ajustar (*Unadjusted Use Case Points*).
- **TCF** = Factor de complejidad técnica (*Technical Complexity Factor*).
- **ECP** = Factor de complejidad del medio ambiente (*Environment Complexity Factor*).

Cada uno de los factores están dados por la suma de diferentes pesos, los cuales derivan de otras variables e indicadores que deben ser tomados en cuenta, así, el primer factor UUCP está dado por:

1. Suma de los pesos de los casos de uso sin ajustar.

2. Suma de los pesos de la complejidad de los actores.

Las sumas de los casos de uso sin ajustar se ejemplifica en la tabla 3 y la suma de los pesos de la complejidad de los actores en la tabla 4:

Tabla 3. Factor de complejidad de los casos de uso [38].

1. Suma de los pesos de la complejidad de los casos de uso sin ajustar				
Tipos de Caso de Uso	Descripción	Peso	Cantidad de Casos de Uso	Sub Total
Simple	De 1 a 3 transacciones Hasta 5 clases	5	4	20
Medio	4 a 7 transacciones De 6 a 10 clases	10	6	60
Complejo	Más de 7 transacciones Más de 10 clases	15	3	45
			Total	**125**

Hay que tener en cuenta que la primera, segunda y tercera columna está establecida por el propio punto de caso de uso.

Esta tabla consiste en evaluar la complejidad de los actores con los que va interactuar el sistema.

Tabla 4. Factor de complejidad de los actores [38].

2. Suma de los pesos de la complejidad de los actores.				
Tipo de Actor	Descripción	Peso	Número de actores	Sub Total
Simple	Otro sistema que interactúa con el sistema mediante una API	1	1	1
Medio	Otro sistema que interactúa con el sistema mediante un protocolo (ej. TCP/IP) o una persona interactuando a través de una interfaz en modo texto	2	1	2
Complejo	Una persona que interactúa con el sistema mediante una interfaz gráfica (GUI)	3	2	6
			Total	**9**

El UUCP representa el valor más representativo de la formula, ya que determinará en gran medida la magnitud de cada producto, así mismo existen otros factores como el TCF y el ECP ya mencionado.

14.5. Determinar la productividad del Sistema.

La cantidad de puntos por casos de uso resultantes del cálculo de estimación de un sistema es una información muy útil para utilizarlo como referencia en el proyecto actual y principalmente para las futuras estimaciones que se vayan a realizar.

Obteniendo este valor referencial, es más sencillo para la empresa de desarrollo asignar un valor monetario para un punto de caso de uso, como así también un valor en el tiempo, es decir ¿en cuánto tiempo se desarrolla un punto de caso de uso?.

La empresa de desarrollo deberá basarse en la experiencia de proyectos anteriores para poder obtener el valor óptimo para éstas dos dimensiones (valor monetario y valor en el tiempo). En cuanto al valor en el tiempo para calcular las horas se recomienda utilizar un valor entre 15 y 30 para el factor de productividad, que se deberá ir probando y mejorando con el correr de los proyectos, si no se tiene un cálculo inicial se podría utilizar el valor de 20.

15. Conclusión.

Para concluir este trabajo de tesis, este capítulo se dedicará a mostrar las conclusiones y recomendaciones obtenidas a lo largo del proyecto.

15.1. Principales logros.

El objetivo de esta tesis era realizar el diseño de una metodología de desarrollo rápido de software con enfoque en procesos ágiles utilizando el estándar Java EE para su ejecución en la nube. Fue propuesto lograr este objetivo a primera instancia apoyado en las metodologías ágiles más utilizadas en la actualidad como son el SCRUM y el XP.

Al realizar una investigación más profunda se notó de la existencia de metodologías ágiles incluso más recientes que las anteriores, como los denominados Modelos Lean (Lean Startup, Lean Canvas, Lean Business Canvas, KanBan) orientados mayormente a empresas de tecnología Web y hacia productos tecnológicos e innovadores, cuyos conceptos fueron aprovechados principalmente para el diseño del modelo logrando finalmente la creación de un modelo compacto y aplicable en sencillos pasos.

Un aspecto importante del modelo creado es que el mismo se basa en la generación de herramientas de trabajo que se constituyen en los tres pilares del desarrollo rápido y que sustenta a toda la etapa principal de desarrollo en sus diferentes fases.

Uno de los elementos tomados en cuenta para la elaboración de la metodología consistió en verificar el cumplimiento mínimo que se requiere para su elaboración, donde se constató la validez de la mayoría de las variables, lo cual convierte a ésta metodología en robusta y confiable pudiendo ser utilizado para cualquier proyecto de envergadura.

15.2. Trabajos futuros.

En un trabajo como el realizado, siempre que se desea que haya una mejora continua en el mismo, se recomienda a maestrandos e investigadores

que tengan interés en la metodología, la complementación del proyecto con el estudio del desarrollo distribuido cubriendo la comunicación e iteración entre los miembros del equipo.

Otra recomendación sería la incorporación de una integración de este modelo con los procesos de desarrollos conocidos como CMMi para verificar en qué nivel podrían trabajar juntos y encontrar una serie de artefactos que se adapten para cubrir los dos objetivos de una sola vez combinando los diferentes enfoques.

También se recomienda la incorporación de procedimientos relacionados con la plataforma Java EE para su desarrollo como backend de una aplicación móvil, consideraciones y estándares a tener en cuenta y verificando el cumplimiento de los pilares para su implementación o la incorporación de otros elementos clave del modelo.

Referencias bibliográficas

[1] Adriana Echeverri, Leonardo Moreno, *Modelo Cloud Computing aplicable a PYMEs*, Universidad de San Buenaventura, Colombia, 2011.

[2] Instituto Nacional de Tecnologías de la Comunicación,*Resumen ejecutivo del Estudio sobre el Cloud Computing en el Sector Público en España*, Inteco, España, 2012.

[3] Siete gigantes se pelean por la nube, Disponible en `http://www.iprofesional.com/notas/135272-Siete-gigantes-se-pelean-por?ella?sepa-por-qu-la-nube-informtica-seducetanto-a-las-empresas-tecnologicas` al 25/04/2012,iprofesional.com, 2012.

[4] Iván Rubén Vela Izozorbe, *Tecnologías de Información y Cloud Computing como apoyo en la eficiencia de las MiPyMEs*,Universidad Veracruzana, México, 2012.

[5] Juan José Franklin Rodríguez Vila, *Utilización de tecnologías cloud computing para la innovación en organizaciones virtuales*. Universidad de San Martin de Porres, Perú, 2010.

[6] Rubén Toledo, *Servicios de Gestión Empresarial para PYMEs: Un caso práctico de SaaS (Software as a Service)*, Universidad Politécnica de Cartagena, Colombia, 2010.

[7] Guido Andrés Casco, *Modelo de Procesos de Desarrollo de Software Paraguayo "Ñanduti"*, Facultad Politécnica ? UNA, Paraguay, 2008.

[8] Dave Evans, *Internet de las cosas, Cómo la próxima evolución de Internet lo cambia todo*, Informe técnico Cisco, EE.UU, 2011.

[9] P. R. González, *Estudio de la Aplicación de Metodologías Ágiles para la evolución de Productos de Software*, Universidad Politécnica de Madrid, España, 2010.

[10] C. Pardo, J. A. Hurtado y C. A. D. Collazos, *Mejora de Procesos de Software ágil con Agile-SPI Process*, scielo.org, Colombia, 2008.

[11] Rachid Anane, *A DSL-based Approach to Software Development and Deployment on Cloud*, 24th IEEE International Conference, 2010.

[12] E. Mucci, *El impacto de la Nube en la productividad de la PYME*, Universidad Politécnica de Catalunya, España, 2010.

[13] Martin Fowler, The new methodology, disponible en `http://martinfowler.com/articles/newMethodology.html#LeanDevelopment`, Martin Fowler Blog, Inglaterra. 2005.

[14] Schenone Marcelo Hernán, *Diseño de una Metodología Ágil de Desarrollo de Software*, Fiuba, Argentina, 2004.

[15] Eric Ries, *The Lean Startup - How today's entrepreneurs use continuous integration to create radically sussccеful business*, Crown Bussiness, EE. UU, 2011.

[16] Ash Maurya, *Running Lean - A systematic process for iterating your web application from Plan A to a plan that Works*, 2010.

[17] A. Orjuela Duarte y M. Rojas C., L*as Metodologías de Desarrollo Ágil como una Oportunidad para la Ingeniería del Software Educativo*, redalyc.org, Colombia, 2008.

[18] Roger Pressman, *Ingeniería de Software - un enfoque práctico*, 6ta edición, Mc Graw Hill, España, 2005.

[19] Luis Merchán Paredes, Diego Gómez Mosquera, *Validación de un modelo liviano para pequeñas empresas de desarrollo de software*, Gestión de la Configuración, Entramado-Unilibre, Colombia, 2011.

[20] Claudia Ramírez, Jannet Aquino, Gabriel Pérez, *Tecnología Cloud Computing aplicada a empresas comerciales basadas en la plataforma Salesforce.com*, universia.net, España, 2010.

[21] E. Delgado Expósito, *Metodologías de desarrollo de software. ¿Cuál es el camino?*, redalyc.org, Cuba, 2008.

[22] Susana Chávez, Adriana Martín, Nelson Rodríguez, María Murazzo, Adriana Valenzuela, Metodología ágil para el desarrollo SaaS, *XIV Workshop de Investigadores en Ciencias de la Computación*, Venezuela, 2012.

[23] Karla Mendes Calo, Elsa Estévez, Pablo Fillottrani, *Un Framework para Evaluación de Metodologías Ágiles*, Universidad Nacional del Sur, Argentina, 2010.

[24] Luis Echeverry, Luz Delgado, *Caso Practico de la Metodología Ágil XP Al Desarrollo de Software*, Universidad Tenológica de Pereira, Colombia, 2007.

[25] Fowler M., Beck K, Brand J., *Refactoring: Improving the Design of Existing Code*, Addison-Wesley. EE.UU, 1999.

[26] Fabián Hernando López Higuera, Comparación de herramientas para el desarrollo de librerías enfocadas a Aplicaciones Web, *Revista Virtual Universidad Católica del Norte*, Nro. 34, Colombia, 2011.

[27] Jack Herrington, *Code Generation in Action*, Manning Publications Co, EE.UU, 2003.

[28] Marcela Daniele, Paola Martellotto, Daniel Romero, *Extendiendo las plantillas genéricas para la definición de casos de uso con un framework genérico distribuido*. Universidad Nacional de Rio Cuarto, Argentina, 2006.

[29] María Ines Lund, Cintia Ferrarini, Laura Aballay, Maria G. Romagnano, Ernesto Meni, *CUPIDo - Plantillas para documentar Casos de Uso*, Universidad Nacional de San Juan, Argentina, 2010.

[30] Ing. Marcela Daniele, AC. Daniel Romero, *Evolución de Plantillas Genéricas para la descripción de Casos de Uso a Plantillas genéricas para análisis y diseño*, SEDICI, Argentina, 2005.

[31] Peñalver G, Meneses A, García S, *SXP - Metodología Ágil para el Desarrollo del Software*, Universidad de las Ciencias Informáticas, Cuba, 2010.

[32] Henrik Kniberg, Mattias Skarin, *Kanban y Scrum- Obteniendo lo mejor de ambos*, C4MEDIA, infoq.com, EE.UU, 2010.

[33] Julián Andrés Collazos Alegría, Modelo de servicios de infraestructura de aplicaciones a través de Cloud Computing, Proyecto "Casa en el Aire", Universidad Icesi, Colombia, 2011.

[34] Henrik Kniberg, *Scrum y XP desde las Trincheras - Como hacemos SCRUM*, infoq.com, EE.UU, 2007.

[35] Eréndira M Jiménez-Hernández, *Metodología Híbrida para Desarrollo de Software en México*, CICIC, 2012.

[36] José Luis Isla Montes, Francisco Luis Gutierrez Vela, *Modelo Estructural de Patrones de Diseño*, Departamento de Lenguajes y Sistemas Informáticos, España, 2003.

[37] Silvia Gabriela Rivadeneira Molina, *Metodologías Ágiles enfocadas al modelo de requerimientos*, Universidad Nacional de la Patagonia Austral, Argentina, 2012.

[38] Remi-Armand Collaris, Eef Dekker, Estimación del costo del software usando puntuación en casos de uso: clarificar las transacciones de casos de uso. Disponible en `http://www.ibm.com/developerworks/ssa/rational/library/edge/09/mar09/collaris_dekker/` en 09/07/14, Developer Works, IBM, EE.UU, 2009.

[39] Mg. Gabriela Robiolo, Transacciones, Objetos de Entidad y Caminos: métricas de software basadas en casos de uso, que mejoran la estimación temprana de esfuerzo, SEDICI, Argentina, 2009.

[40] Méndez Nava, Elvia Margarita, *Modelo de Evaluación de Metodologías para el Desarrollo de Software*, Universidad Católica Andrés Bello, Venezuela, 2006.

[41] Profesor Rafael Barzanallana, Página del Profesor, Universidad de Murcia Disponible en `http://www.um.es/docencia/barzana/IAGP/Iagp3.html`, Universidad de Murcia, España, 2007.