UNIVERSIDAD NACIONAL DEL ESTE

FACULTAD POLITECNICA

Maestría en Informática y Computación

Modelo para el Desarrollo Rápido en la Plataforma Java EE 6 para Aplicaciones Empresariales en la Nube

Tesis de Maestría

Marcos Adrián Jara Rodríguez

Ciudad del Este, Paraguay

2014

Marcos Adrián Jara Rodríguez

MODELO PARA EL DESARROLLO RÁPIDO

EN LA PLATAFORMA JAVA EE 6 PARA

APLICACIONES EMPRESARIALES EN LA NUBE

Tesis presentada a la Universidad Nacional del Este como requisito parcial para la obtención del título de Magister en Informática y Computación

Tutor: Magister Oscar Adolfo Vallejos

Ciudad del Este, Paraguay
2014

Jara Rodríguez, M. A. (2014). Modelo para el Desarrollo rápido en la Plataforma Java EE 6 para el Desarrollo de Aplicaciones Empresariales en la Nube.

Marcos Adrián Jara Rodríguez. 150 páginas.

Tutor: Magister Oscar Adolfo Vallejos

Tesis académica de maestría en Ciencias Aplicadas –

Universidad Nacional del Este, 2014.

Marcos Adrián Jara Rodríguez

MODELO PARA EL DESARROLLO RÁPIDO

EN LA PLATAFORMA JAVA EE 6 PARA

APLICACIONES EMPRESARIALES EN LA NUBE

Esta tesis fue evaluada y aprobada para la obtención del título de Magister en Informática y Computación por la Universidad Nacional del Este.

Mesa Examinadora:

Dedicatoria

Esta tesis la dedico:

A mi amada esposa por su incondicional compañía en los momentos en que realizaba este trabajo.

A mis hijas Belén y Berenice, por los momentos alegres que me brindaban en las situaciones que más necesitaba.

A mi Padre y Madre, por ser ellos mis mejores ejemplos de vida y perseverancia ante las adversidades de la vida.

Agradecimiento

Muchas Gracias al Tutor de esta Tesis, Magister Oscar Vallejos por sus sabios consejos y recomendaciones.

Agradezco especialmente al Director de la Maestría Doctor David La Red por su incansable labor durante todo el proceso de elaboración de este proyecto y por su generosidad en la hora de transmitir sus conocimientos.

A los miembros de la dirección de Investigación de la Facultad Politécnica, por brindarme esta valiosa oportunidad de formación y apoyo continuo, así como a todos los demás directivos de la Universidad Nacional del Este.

Índice general

Índice de figuras

Índice de tablas

Capítulo I

Introducción

1.1. Resumen

El presente proyecto presenta un modelo para 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 y puesta en marcha de soluciones de

software para los clientes finales encarada por las Pequeña y Mediana Empresa (PYME) de desarrollo, consecuentemente minimizando los costos que derivan de largos periodos de implementación aumentando así la rentabilidad [1].

Para el diseño de este modelo se tuvieron en cuenta procesos y técnicas aplicables a las diferentes etapas del desarrollo del software en particular como son el análisis, construcción, pruebas y despliegue así como también se tuvieron en cuenta las fases previas de organización general que sirven para lograr el entendimiento global de los objetivos de producir el producto.

Durante el proceso de investigación se incursionó en el estudio de varias metodologías ágiles, de las cuales se tomaron como base los modelos con mayor fuente bibliográfica existente, entre las cuales se encuentran el SCRUM, el XP – (Extreme Programming) y otras metodologías derivadas del Lean Development, analizando cada principio del manifiesto ágil para verificar en qué medida cada metodología lo cumple y en qué grado, el estudio de mejores prácticas y revisión de patrones de diseño aplicables, encontrando puntos en común entre ellas que promuevan una estandarización en conjunto con la plataforma Java EE.

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 con lo cual la empresa puede ser capaz de sustentar el desarrollo de software empresarial en la nube utilizando la tecnología Java EE optimizando costo, tiempo y recursos.

Finalmente como resultado de todo el proceso se obtiene un modelo el cual es bautizado con el nombre de CloudAgileJ.

1.2. Introducción

1.2.1. Motivación

Este trabajo de investigación está enfocado a servir de utilidad para las Empresas de la región este del Paraguay (Ciudad del Este y alrededores) que se dedican al rubro de desarrollo de software, cuyos proyectos se destinan a clientes que corresponden a pequeñas y medianas empresas (PYMES), las cuales, en la mayoría de las ocasiones generan demandas de aplicaciones corporativas del tipo sistema de gestión (SG).

Debido al auge actual del Cloud Computing, muchas personas y empresas van cambiando su percepción en cuanto al uso del software. Un estudio sobre el Cloud Computing en el Sector Público de España [2] revela que el 74% de las organizaciones utilizan algún tipo de servicio cloud (Drive, Sky, Dropbox, correo electrónico, etc.), siendo las aplicaciones de correo electrónico y los programas de gestión los más migrados. Según el mismo estudio menciona que las administraciones públicas y el sector financiero son las que más están apostando por la adopción del cloud y que los ingresos mundiales por servicios cloud alcanzarán los 55.000 millones de dólares en 2014.

Por otro lado, según [3] el cloud computing seduce a los gigantes proveedores de software, hasta tal punto que hoy empresas como Samsung, IBM, Microsoft, Apple, HP, Google y Telefónica se disputan entre sí para obtener una mayor cuota de mercado y consumidores, ofreciendo para ellos diferentes tipos de soluciones. Como consecuencia, la tendencia es que cada vez más empresas de desarrollo locales y regionales buscan desarrollar soluciones propias también en ambiente Web ya preparadas para el cloud [4], buscando siempre ventajas competitivas en cuanto al costo y tiempo de puesta en marcha de sus soluciones en producción (Implantación) [5].

1.2.2. El desafío de pasar a la nube

Las pequeñas y medianas empresas, que a partir de ahora serán llamados de clientes o usuarios finales, muy a menudo requieren soluciones integrales para mejorar sus controles internos y externos, que necesitan con urgencia. Es muy común, cuando el analista realiza al cliente la pregunta de ¿para cuándo requiere la solución?, obtener una respuesta como: *es para ayer* indicando de esta forma que lo necesita con mucha urgencia.

Por esa razón, las empresas de desarrollo de software se encuentran con grandes desafíos a la hora de brindar sus servicios para este tipo de clientes, haciendo hasta lo imposible con el poco tiempo que se les deja y con el bajo presupuesto que se les asigna para lograr la implementación a tiempo y con la exigencia de calidad requerida [6].

Estos desafíos generalmente se dan por situaciones resultantes de los pesados procesos de desarrollo que van de la mano con tecnologías como Java EE, así como también por la alta curva de aprendizaje de tecnologías relacionadas a la plataforma, una falta de claridad en cuanto a los requerimientos solicitados y mucha complejidad en la implementación por la falta de adopción de frameworks que todos juntos se suman al poco tiempo disponible en realizar y culminar el producto solicitado.

Cabe destacar que estas empresas de desarrollo también pueden estar prestando sus servicios a otros tipos de clientes que sí disponen del tiempo y recursos necesarios para solicitar la elaboración de software con alguna metodología tradicional, la cual se exige toda la documentación del sistema como parte del producto terminado, como por ejemplo con la adopción del modelo RUP [7], no obstante este trabajo sólo contempló los tipos de servicios de desarrollo que son ofrecidos para el primer tipo de necesidad de cliente mencionado, aunque al lograr el objetivo del trabajo, esta metodología puede adaptarse para ambos tipos de clientes, sin importar el tipo de empresa y/o servicio ofrecido.

1.2.3. Internet de las cosas

El Cloud Computing y las Aplicaciones SaaS promoverán el uso del Internet de las Cosas.

Según se lee en [8] "Internet de las Cosas (IdC), lo cambiará todo, incluso a nosotros mismos". Desde sus inicios, Internet se utilizaba principalmente como herramienta para buscar información. En los últimos 10 años se ha estado viviendo una nueva forma sobre el uso de Internet, donde la mayoría se ha convertido en social, transaccional y móvil. En el año 2008, el número de "cosas" conectadas a Internet sobrepasó al número de habitantes del planeta y se estima que para el año 2020 habrá 50.000 millones de dispositivos conectados a Internet.

Según un estudio de Cisco, sólo en el año 2015 el volumen de negocios de las cosas conectadas a Internet ascenderá a 475.000 millones de euros.

El IdC representaría un negocio mundial de 10.900 billones de euros en 5 años por la mayor productividad, ahorro de costes y nuevos mercados para las empresas.

Con esto se crea una nueva posibilidad con lo que brinda Internet, para que todos, personas y cosas, estén permanentemente conectados y podamos recibir y procesar información en tiempo real.

1.3. Hipótesis y objetivos

1.3.1. Hipótesis

Es posible realizar el diseño de una modelo de desarrollo rápido de Software con enfoque en procesos ágiles utilizando el estándar Java EE para su ejecución en la Nube.

1.3.2. 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.

1.3.3. 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.

- Estudiar la factibilidad real de aplicar la metodología ágil en el desarrollo de software para la nube utilizando el estándar Java EE.

- Identificar las características principales de las aplicaciones Web en la nube, de forma a estandarizarlo como patrones y obtener el beneficio de la reutilización.

- Proponer arquitecturas de optimización para rápidos despliegues y control de versionamiento de aplicaciones Web integrada con la base de datos.

- Evaluar mecanismos eficientes para mantener una estructura semi-acoplada entre la aplicación y el diccionario de datos de forma a poder hacer un uso óptimo de la refactorización de componentes.

- 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.

- Analizar la factibilidad de reducir el tiempo de despliegue de aplicaciones Java EE en servidores de aplicación, a través de alguna solución resultante del estudio en este campo.

- Evaluar la rapidez del desarrollo de las aplicaciones Web en la nube y su repercusión en el tiempo, costo y calidad de desarrollo para las empresas y los clientes finales.

1.4. Antecedentes

Esta investigación implicó el estudio de varias fuentes de información, principalmente en lo relacionado a:

a) Sistemas preparados para el Cloud Computing.

b) Plataforma Java EE como alternativa para el desarrollo en la Nube.

c) Metodologías ágiles de desarrollo.

d) Herramientas disponibles para generación de código que aumentan la productividad del desarrollador.

e) Diferentes tipos de aplicaciones de gestión.

f) Próximas tendencias de la nube en cuanto a aplicaciones de software.

Como resultado de la investigación se han evidenciado una gran fuente de antecedentes, las cuales seguidamente se agruparon en Sustantivos y Metodológicos.

Si bien existe bastante antecedente en la bibliografía estudiada, no se encontraron indicios sobre los puntos b) y c) combinadas entre sí – aunque sí

de forma independiente, ya que la Plataforma Java EE está más orientada a seguir un riguroso proceso de desarrollo, por ese motivo el surgimiento de tecnologías más ágiles como Ruby on rails. Tampoco se halla una integración entre los puntos a) – f) para lograr el objetivo propuesto por este trabajo. Es por esa razón de que se considera que este trabajo reúne todas las características de originalidad siendo muy relevante académica y socialmente, ya que:

a) Se espera que el conocimiento científico producido resulte aplicable.
b) Se aprovecha el conocimiento del maestrando en la región, por las empresas del sector y específicamente por alguna Institución tecnológica del ramo de desarrollo de software

Se consideran que los antecedentes sustantivos más relevantes a este trabajo son los relacionados a la construcción de aplicaciones utilizando metodologías ágiles y a la plataforma Java EE 6 en Cloud Computing.

En ese contexto se pueden mencionar los siguientes antecedentes:

- J. M. A. Rodríguez, Desarrollo Ágil en JEE con herramientas OpenSource, 2010.
- Susana Chávez, Adriana Martín, Nelson Rodríguez, María Murazzo, Adriana Valenzuela, Metodología AGIL para el desarrollo SaaS, 2012
- Adriana Echeverri, Leonardo Moreno, Modelo Cloud Computing aplicable a PYMES, 2011

De forma análoga a los antecedentes sustantivos, también se puede decir que entre los antecedentes metodológicos se encontraron los siguientes

- Eréndira M Jiménez-Hernández, Metodología Híbrida para Desarrollo de Software en México, 2012

- Juan José Franklin Rodríguez Vila, Utilización de tecnologías cloud computing para la innovación en organizaciones virtuales. 2010
- Juan José Franklin Rodríguez Vila, Utilización de tecnologías cloud computing para la innovación en organizaciones virtuales. 2010

1.5. Marco teórico

Descripción del encuadre o marco teórico

Java EE

Es una plataforma creada por Sun en el año 1997, para el desarrollo de aplicaciones distribuidas orientadas principalmente a las corporaciones. En el desarrollo empresarial se ponen de manifiesto ciertos requisitos esenciales, no tan críticos en otras aplicaciones, que debe cumplir un desarrollo y que con Java EE se puede conseguir utilizando principalmente software libre [9], entre estos requisitos se pueden nombrar los siguientes:

- Escalabilidad, tanto horizontal como vertical.
- Fiabilidad.
- Facilidad de mantenimiento.
- Seguridad.
- Rendimiento.
- Extensibilidad.
- Flexibilidad.

El objetivo final es conseguir la máxima productividad del equipo del proyecto, por ello el uso de herramientas que en esta plataforma permita un desarrollo ágil resulta relevante para el logro del objetivo propuesto.

Si bien la plataforma Java EE presenta muchas ventajas, también trae consigo una serie de inconvenientes, como por ejemplo, la complejidad que dificulta su adopción por los desarrolladores menos experimentados así como también el hecho de que existen muchos modelos de desarrollo, frameworks y Apis que pueden confundir a los desarrolladores.

En conclusión, desde el punto de vista teórico, JEE es una especificación de especificaciones para distintos conceptos: XML, Servicios Web, EJB, etc. También se cuenta con un conjunto de buenas prácticas o guías que se denominan JEE Blueprints.

Metodologías y procesos de construcción de software
Proceso Unificado de Modelado (RUP)

Considerada como metodología tradicional, el proceso unificado de desarrollo (RUP) es una metodología para la ingeniería de software, que va más allá del mero análisis y diseño orientado a objetos para proporcionar una familia de técnicas que soportan el ciclo completo de desarrollo de software. El resultado es la documentación consistente de un proceso basado en componentes, dirigido por los casos de uso, centrado en la arquitectura, iterativo e incremental [10].

Plataformas como Java EE, están muy enfocadas a cubrir expectativas requeridas por el Proceso RUP.

Características principales de RUP

Centrado en los modelos: Los diagramas son un vehículo de comunicación más expresivo que las descripciones en lenguaje natural. Se trata de minimizar el uso de descripciones y especificaciones textuales del sistema.

Guiado por los casos de uso: Los Casos de Uso son el instrumento para validar la arquitectura del software y extraer los casos de prueba.

Centrado en la arquitectura: Los modelos son proyecciones del análisis y el diseño constituye la arquitectura del producto a desarrollar.

Iterativo e incremental: Durante todo el proceso de desarrollo se producen versiones incrementales (que se acercan al producto terminado) del producto en desarrollo.

Beneficios que aporta RUP

- Permite desarrollar aplicaciones sacando el máximo provecho de las nuevas tecnologías, mejorando la calidad, el rendimiento, la reutilización, la seguridad y el mantenimiento del software mediante una gestión sistemática de riesgos.
- Permite la producción de software que cumpla con las necesidades de los usuarios, a través de la especificación de los requisitos, con una agenda y costo predecible.
- Enriquece la productividad en equipo y proporciona prácticas óptimas de software a todos sus miembros.

Metodologías ágiles

El objetivo principal de las metodologías ágiles es el de esbozar los valores y principios que deberían permitir a los equipos de trabajo, desarrollar software rápidamente y responder a los cambios que pudieran surgir a lo largo del proyecto [11].

El Manifiesto ágil

El manifiesto ágil es, un documento que resume la esencia de las metodologías ágiles, y una serie de postulados que valora principalmente los siguientes aspectos:

1. Se presta más atención al individuo y las interacciones del equipo de desarrollo que sobre el proceso y las herramientas.

2. El objetivo principal es el de desarrollar software que funciona, más que conseguir una buena documentación.
3. En todo momento se trata de la colaboración con el cliente, más que la negociación de un contrato.
4. Responder a los cambios, más que seguir estrictamente un plan [12, 13].

Principios ágiles

De estos cuatro postulados de valores, se desarrollaron doce principios para el manifiesto ágil y que son:

1. La satisfacción del cliente a través de la entrega rápida y continua de paquetes de software útiles y de valor.
2. Nuevos requisitos son bienvenidos incluso en la etapa final del desarrollo.
3. Entrega con frecuencia de software que funcione, preferentemente en semanas en vez de meses.
4. El software que funciona es la prueba fehaciente de que se puede medir el progreso del proyecto.
5. Desarrollo sostenible, capaz de mantener un ritmo constante.
6. Trabajo cercano de forma cotidiana entre las personas de negocio y desarrollares.
7. La conversación cara a cara es la mejor forma de comunicación.
8. Los proyectos están construidos en torno de personas motivadas, a los cuales se les tiene que dar la confianza necesaria para que realicen la tarea.
9. Atención continua a la excelencia técnica y al buen diseño.
10. Simplicidad.
11. Equipos que se auto organizan.

12. Adaptación regular a las circunstancias cambiantes.

Comparación entre metodologías agiles y metodologías tradicionales como la del RUP

En la *tabla 1.5-a* se muestran las principales diferencias de las metodologías ágiles con respecto a las tradicionales. Estas diferencias hacen referencia de igual manera a aspectos organizacionales y al proceso de desarrollo [10].

METODOLOGÍAS AGILES	METODOLOGÍAS TRADICIONALES
Basadas en heurísticas provenientes de prácticas de producción de código.	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Reglas de trabajo impuestas internamente (por el equipo)	Reglas de trabajo impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas.
Flexibilidad en los contratos debido a la respuesta a cambios	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones en determinadas etapas del proceso
Grupos pequeños (menos de 10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos trabajando en diferentes tareas.
Pocos artefactos	Mas artefactos
Pocos roles	Mas roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos.

Tabla 1.5-a – Comparación de metodologías de desarrollo [10].

De acuerdo a la *tabla 1.5-a* se puede observar que las metodologías de desarrollo ágil son más orientadas a procesos de desarrollo de software de pocas semanas de desarrollo y con bajos niveles de formalización en la documentación requerida.

A continuación se describen con mayor nivel de especificación las principales semejanzas y diferencias de las metodologías ágiles y las metodologías tradicionales (pesadas) [10]:

Prácticas de desarrollo: En las metodologías de desarrollo ágil se procura realizar los procesos de software de acuerdo a las prácticas que le han dado

resultados al grupo. En las metodologías pesadas se desarrolla de acuerdo a las normas sugeridas por los estándares de desarrollo.

Adaptación al cambio: En las metodologías ágiles por la misma capacidad de reacción son más adaptables a los cambios, por el contrario en las metodologías pesadas por el nivel de formalidad en la fase de requerimientos son más resistentes al cambio.

Control: Por su capacidad de adaptación el proceso se hace menos controlado que en las metodologías tradicionales que ejercen mayor control en el proceso por su nivel de formalización.

Documentación: En las metodologías ágiles no se hace énfasis en la documentación ni en los artefactos a diferencia de las metodologías pesadas.

Equipo de trabajo: En las metodologías ágiles existen bajo número de participantes y roles, por el contrario en las metodologías pesadas se sugieren los roles que proporcionan las normas.

En conclusión, de acuerdo a la tabla se da prioridad: a los individuos y las interacciones más que a los procesos y a las herramientas, a los sistemas funcionando antes que a la documentación detallada, a la colaboración con el cliente antes que la negociación de contratos, a la respuesta al cambio antes que seguir el plan.

Principales metodologías ágiles

Las metodologías ágiles resuelven los problemas surgidos, posteriormente, a la masificación del uso del computador personal, dado que las expectativas y necesidades por parte de los usuarios se hicieron más urgentes y frecuentes [11]. Fue así como a comienzo de los 90 surgieron propuestas metodológicas para lograr resultados más rápidos en el desarrollo de software sin disminuir su calidad [9, 13].

A continuación se describen las características esenciales de las metodologías ágiles más utilizadas como son SCRUM y XP [14].

SCRUM

Desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle. Define un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos 10 años.

Está especialmente indicada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos. El desarrollo de software se realiza mediante iteraciones, denominadas sprints, con una duración de 30 días. El resultado de cada sprint es un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a lo largo proyecto, entre ellas destaca la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración.

XP

La programación extrema o eXtreme Programming (XP) es una metodología de desarrollo formulada por Kent Beck, autor del primer libro sobre la materia, Extreme Programming Explained: Embrace Change (1999). Es el más destacado de los procesos ágiles de desarrollo de software. La programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. Los defensores de la XP consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Creen que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos.

Otras metodologías

Lean Startup es una manera de abordar el lanzamiento de negocios y productos que se basa en aprendizaje validado, experimentación científica e Iteración en los lanzamientos del producto para acortar los ciclos de desarrollo, medir el progreso y ganar valiosa retroalimentación de los clientes. De esta manera las compañías, especialmente Startups pueden diseñar sus productos o servicios para cubrir la demanda de su base de clientes, sin necesitar grandes cantidades de financiación inicial o grandes gastos para lanzar un producto.

Originalmente desarrollado en 2008 por Eric Ries teniendo en mente compañías de alta tecnología, la filosofía Lean Startup se ha ampliado para aplicarse a cualquier individuo, grupo o empresa que busca introducir nuevos productos o servicios en el mercado [15, 16].

Evolución del producto software

La ISO/EIC 14764 (1999) define mantenimiento del software como *el proceso por el cual un producto sufre modificaciones en su código y documentación asociada para solucionar un problema o mejorar*. El estándar 1219 de IEEE (1998) lo define como *la modificación de un producto software tras ser liberado para corregir defectos, mejorar la ejecución de otros atributos, o adaptar el producto a un nuevo entorno* (este último objetivo es el que persigue el proyecto presentado en esta tesis de máster). Sea cual sea la definición que se considere, el concepto de evolución del software implica un cambio continuo desde un estado menor, más simple o peor a uno superior o mejor. Los objetivos que se persiguen al evolucionar un producto software son diversos: mantener operativo el sistema, incrementar la satisfacción de los usuarios, maximizar la inversión y reducir los costes, alargar la vida del software o adaptarlo a nuevos cambios o requisitos [17].

Las conocidas Leyes de Lehman para la evolución del software establecen:

***1ª Ley*:** Un programa grande que es utilizado se somete a un cambio persistente o se convierte en menos útil progresivamente. El proceso de cambio continúa hasta que se juzga como más rentable remplazar el sistema por una nueva versión.

***2ª Ley*:** Como un programa grande cambia de forma continua, su complejidad la cual refleja una estructura deteriorándose, se incrementa a menos que se realice un trabajo para mantenerla o reducirla.

Y: Las medidas de un proyecto global y atributos del sistema se auto regulan cíclicamente con tendencias e invariantes estadísticamente determinables.

Y: El ratio de actividad global en un gran proyecto de programación es invariante.

Ley: En una evolución fiable y planificada, un gran programa sometido a un cambio debe estar disponible para una ejecución regular del usuario en el máximo intervalo determinado por su crecimiento neto. Es decir, el sistema desarrolla un promedio característico de crecimiento seguro, que de ser excedido, causa problemas de calidad y utilización con tiempo y coste que excede del previsto.

Durante las 3 décadas pasadas el mantenimiento del software se caracterizó como un iceberg [18]. Se esperaba que lo inmediatamente visible fuese todo lo que había, pero se sabía que bajo la superficie se encontraba una enorme masa de problemas y costos potenciales. En ese sentido nada ha cambiado hasta hoy.

En el ciclo de vida del desarrollo del software *la primera versión es sólo una pequeña parte del trabajo* ya que el *mantenimiento* y la *evolución* cubren la *mayoría* del ciclo de vida. De hecho, se estima que la evolución del software consume un 80% del presupuesto mientras que el desarrollo es el 20% restante, tal como lo muestra la *figura 1.5-a.*

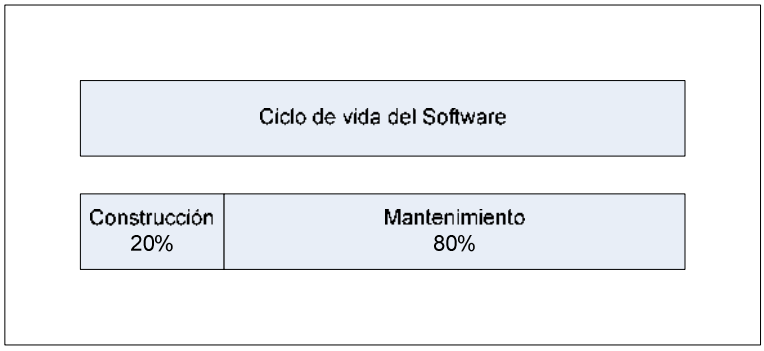

Figura 1.5-a – Ciclo de vida del Software (Fuente propia).

Software como servicio para empresas (SaaS)

Con la implantación masiva de Internet en las empresas ha llegado una nueva forma de adoptar las TICs, esta es la adopción de las TICs basada en internet [6]. Dicha solución proporciona herramientas de comunicación efectivas de bajo coste para los clientes. Esto es algo muy importante a tener en cuenta para la PYME, que adopta las TICs de una forma gradual y precisa de no tener que hacer grandes inversiones y repartir el esfuerzo durante años. Pero también aparece un nuevo problema, la seguridad, que puede llegar a ser una nueva barrera en esta forma de adopción TICs [19].

La adopción de las TICs basadas en Internet ha hecho posible el desarrollo del *paradigma de todo como servicio*, donde el usuario paga sólo por el uso realizado del servicio [20]. Estos pueden variar desde aplicaciones software (Aplication Service Provider (ASP) y Software as a Services (SaaS)) hasta infraestructuras de sistemas (Infraestructure as a Service o IaaS). Esta fórmula se intuye como una buena solución para la PYME, en la línea de saltar las barreras que le impedían aprovechar las TICs [1].

Algunos expertos definieron este concepto como “eRent” de Sistemas de Información (SI), realizado a través de internet y gracias a un ASP. Afirman que para la PYME, los SI a través de “eRental” podría ser una solución atractiva frente a las costosas y complejas adquisiciones e implementaciones de las TICs tradicionales [20].

También se afirma que la principal razón para que la PYME decida adoptar las TICs mediante la contratación de un ASP, es que estos permiten un control total del coste y un menor coste a la hora de adoptarlas y mantenerlas [4]. A pesar de esto, asegura que si se examina detenidamente esta decisión se observa que los clientes no enfatizan estas razones, e induce a pensar que la perspectiva del coste es secundaria para la PYME. De esta forma identifica las tres razones principales por las que los clientes de ASPs contratan sus servicios [20]. La principal es que la empresa busca *externalizar* todo lo que no sean competencias básicas de su negocio, cosa que en su mayoría suelen cumplir los Sistemas de Información, luego se encuentran *la falta de personal cualificado* y *la estrategia general de la organización* [21].

1.6. Metodología

1.6.1.Dimensión de la estrategia general

a) Tipo de estrategia y fundamentación

El marco metodológico es de carácter descriptivo, investigativo y exposición de resultados.

Está encaminado a especificar los conceptos, características, importancias, ventajas, desventajas, análisis comparativo y sugerencias para mejoras en la rapidez de entrega del producto por el equipo de desarrollo para la obtención de los resultados que se busca. Para ello se pretende estudiar los trabajos ya realizados sobre el mismo tema, realizando al final una conclusión sobre el impacto del proyecto.

Atendiendo a la metodología del trabajo de investigación, los siguientes procedimientos se llevaron a cabo para la realización del trabajo:

- Diseño, elaboración de planes y los programas de trabajos.
- Definición de Alcance y objetivos.

- Determinación de los recursos necesarios para realizar el proyecto.
- Confección y redacción del informe final.
- Redacción de la Metodología y publicación en formato de Wiki.
- Evaluación de la Metodología en base a un proyecto real.
- Determinación de las conclusiones y recomendaciones.
- Redacción de la carta e informe final.

a.1) Universo

Para este proceso de investigación se tuvo en cuenta el trabajo de construcción de sistemas realizado en pequeñas y medianas empresas desarrolladoras, los cuales producen software generalmente para clientes finales que resultan ser las empresas ya sea comerciales o Entidades.

a.2) Unidad de análisis

La unidad de análisis corresponde específicamente a una pequeña Empresa de desarrollo de Software cuyos proyectos son desarrollados por un equipo de 3 a 5 personas de forma colaborativa desde una misma oficina cada uno con computadores conectados a la red interna y a Internet. En la misma de producen software en la Nube para Clientes locales.

a.3) Selección de casos

Según la percepción del maestrando, por su experticia en el área y el conocimiento adquirido, el mismo ha detectado dos casos particularmente distintos en la región.

Un caso: una empresa pequeña de desarrollo de software, que por la propia situación del mercado no puede cobrar a su cliente un precio muy alto, real de lo que vale su producto, por lo tanto, debe fabricarlo y ponerlo en

producción en el menor tiempo posible y con una mínima cantidad de recursos (analistas-programadores), por lo tanto para obtener el beneficio económico que espera debe hacerlo rápido y ágilmente.

Otro contexto, corresponde a la Empresa que provee sus servicios a Clientes que pueden solventar el costo de la realización de un producto con el desarrollo de alguna metodología más pesada como la del RUC independientemente del precio que establezca el mercado, ya que generalmente poseen el recurso necesario para exigir a las Empresas de desarrollo toda la documentación relacionada al producto e incluso el código fuente de la aplicación. Entre los clientes de este tipo de empresas se pueden mencionar a Instituciones del gobierno, por ejemplo.

a.4) Énfasis en el proceso lineal

Ha habido un proceso lineal de relación teoría-empiria, con diferentes espacios y momentos para la definición de las hipótesis, la obtención de los datos, su análisis y su posterior interpretación.

a.5) Rol del investigador

El investigador intenta descubrir una realidad preexistente a la investigación, formulando hipótesis que luego habrá de confrontar con los datos obtenidos durante la investigación, como resultado de lo cual eventualmente podrían tener que reformularse las hipótesis originales.

1.6.2. Dimensión de las técnicas de obtención y análisis de información empírica

a) Técnicas de obtención y análisis

A los efectos de la realización de la investigación propuesta, se considera oportuno efectuar la recolección de información mediante cuestionarios y eventualmente entrevistas como así también de encuestas, para luego

realizar un análisis y comparación antes y después de la implementación de la metodología propuesta, a los efectos de lograr un modelo eficiente.

1.7. Estructura de esta tesis

Habiéndose indicado precedentemente la situación problemática motivadora de este trabajo de investigación, así como los principales antecedentes y los más destacados conceptos teóricos que constituyen su marco conceptual, y habiéndose mencionado los principales aspectos de la metodología a utilizar, se indica a continuación los restantes capítulos en que se ha estructurado esta tesis.

El **Capítulo II** tiene el objetivo de realizar un análisis sobre la problemática recurrente en las empresas de desarrollo, que ocurren con el intento de desarrollar aplicaciones, con el fin de brindar alternativas de soluciones oportunas, en el momento de realizar la propuesta.

También se realiza un análisis de las metodologías agiles más importantes en base a lo que dicta su manifiesto y que tienen relevancia en este trabajo, con el fin de profundizar en las diferentes áreas y poder hacer un buen provecho de los conocimientos previos en el momento del desarrollo de la nueva metodología.

Seguidamente, también en el Capítulo II, se engloban los aspectos fundamentales que forman al quehacer diario de una empresa que desarrolla software, sus retos en el campo del desarrollo, sus frustraciones a la hora de poner en marcha un proyecto de implantación y los nuevos desafíos que surgen en el diseño e implantación de los sistemas en la nube.

El **Capítulo III** representa al apartado principal de esta tesis, ya que en ella se describe el modelo propuesta de una forma representativa, en base a los 3 pilares fundamentales en la cual se halla sustentada. Se exponen de manera gráfica algunos conceptos y se utilizan diferentes tipos de diagramas para representar el flujo y los procesos, así como también los roles y actividades que deben ser tomados en cuenta. Para la definición y especificación de las

fases se asumen además de las fases de desarrollo, también unas fases tendientes a orientar las etapas de maduración de la empresa de desarrollo en cuanto al desarrollo de herramientas internas que irán mejorando con el tiempo y con el objetivo de hacerla más eficiente a largo plazo.

En el **Capítulo IV** se realiza un análisis del modelo propuesto. Se realiza la evaluación del modelo en términos de calidad y cantidad.

La evaluación realizada en este capítulo, se enfoca en brindar un resultado desde dos puntos de vista, uno más cualitativo y el otro, mas cuantitativo, el primero, se basa en verificar si el modelo cumple mínimamente con lo que se requiere para la elaboración de una metodología de desarrollo de software y la segunda se orienta en verificar cómo y en qué medida la propuesta se alinea con los valores postulados en el manifiesto ágil realizando una comparación con las dos metodologías ágiles más importantes en la actualidad.

Capítulo V – En este capítulo se expone de forma práctica la aplicación de la metodología, simulando un caso real para un sistema nuevo que es desarrollado dentro de una empresa.

Con la demostración de un caso práctico se ejemplifica de forma más clara los pasos que deben ser obtenidos en cada etapa y fase de la iteración en marcha.

Seguidamente, en el mismo capítulo, también se presentan los resultados obtenidos con la utilización de esta metodología en caso real de desarrollo de software dentro de una empresa también real.

Capítulo VI – En lo referente a conclusiones y trabajos futuros se analiza el resultado del trabajo verificándose el cumplimiento de los objetivos planteados y la verificación de la hipótesis indicada en base a los indicadores presentados. Se exponen las principales conclusiones y se dan las sugerencias para futuras líneas de investigación.

1.8. Discusiones y comentarios

En este capítulo fueron proporcionados diferentes conceptos e ideas sobre los criterios a tener en cuenta para la elaboración del trabajo, mencionando principalmente su objetivo y rescatando los fundamentos teóricos principales que serán utilizados como referencia para este trabajo.

De los conceptos teóricos se rescata que tecnologías como Java EE, fueron desarrolladas bajo un pensamiento para su utilización con metodologías tradicionales como el RUP, muy probablemente porque la aparición de ambas herramientas datan de la misma época, motivo por el cual se deben analizar diversos factores, principalmente aquellos relacionados a procedimientos técnicos, con el fin de orientar el uso de Java EE hacia un método más ágil.

Las metodologías ágiles de desarrollo requieren de una estrategia mínima para el logro de un entorno donde el mismo pueda ser viable. La mayoría de la bibliografía encontrada refiere en muchas partes sobre el cómo realizar las actividades o como se van a pasar de una fase a otra, roles, etc.; en síntesis, todo lo relacionado a procedimientos, pero para ser ágil en el desarrollo, no sólo es importante el procedimiento a ser llevado a cabo, sino también la selección de las herramientas a utilizarse, ya que ésta incide considerablemente en su aplicación práctica.

No se puede discutir que se dispone de una plataforma robusta y escalable, Java EE ha ido evolucionando hasta hoy, pero todavía presenta ciertas deficiencias en cuanto a agilidad que deben ser estudiadas para canalizarlas y sacarles el mayor provecho con este nuevo enfoque.

En el Capítulo II será analizado con mayor detalle todas esas situaciones limitantes ya sea que estén impuestas por la propia tecnología o por los tipos de procedimientos utilizados por el equipo.

Capítulo II

Descripción del problema

2.1. Resumen

Este capítulo describe el día a día de la práctica del desarrollo de sistemas en las empresas de software y los problemas comúnmente encontrados en las actividades rutinarias que retrasan la producción ágil de productos de software.

Mediante este capítulo también se dan a conocer las deficiencias en los procesos, personas y herramientas que podrían ser mejorados con la aplicación correcta de una metodología.

Como fue explicado en el Capítulo I, este proyecto está enfocado en el desarrollo de un modelo para el desarrollo rápido de Sistemas en la Nube,

cuyos procedimientos por lo general consisten en la producción de un nuevo estilo de software o de nueva generación, el cual se denomina software como servicio o SaaS (Software as a Service) por sus siglas en inglés.

Según [22] el Software como Servicio (SaaS) entrega software y datos como un servicio sobre Internet usualmente por medio de un browser que corre al lado del cliente sin tener que instalar nada en el dispositivo.

Este modelo de sistemas tiene una serie de ventajas sobre sus antecesores como el software Desktop o el Cliente-Servidor tanto para quien desarrolla el software, como así también para los clientes [1], pero al mismo tiempo acarrea una serie de procesos encadenados y bien organizados que se requerirán de parte de la Empresa de Desarrollo para lograr mantener un cierto control sobre varios puntos críticos que deben ser tomados en cuenta antes de su aplicación por la complejidad y tiempo que acarrean, como por ejemplo:

- El desarrollo compartido y las pruebas.
- El versionamiento del código fuente y aplicaciones.
- Los usuarios y las cuentas de los clientes.
- El uso de varias herramientas de gestión interna.
- La implantación y actualizaciones recurrentes en tiempo real.
- Infraestructura, equipos, servidores, configuración de balanceo de carga, clusterización, virtualización, etc.
- Entre varios otros elementos.

Puede parecer que todo este conjunto de procesos encadene una serie de otras actividades de gestión que impidan la rápida producción y disponibilidad del software, aunque es cierto que la supremacía de SaaS, por el hecho de poder actualizar sólo una copia del software en cada nueva versión, se alinea perfectamente con el ciclo de vida del software Ágil [22].

Las metodologías ágiles están revolucionando la manera de producir software, y a la vez generando un amplio debate entre sus seguidores y quienes por escepticismo o convencimiento no las ven como alternativa para las metodologías tradicionales [22].

2.2. Más allá del desarrollo

Para un proyecto de software en la nube, no sólo la construcción del software es importante, se requiere además de infraestructura acorde a la necesidad actual y previsión de crecimiento (como son Equipos, Servidores Centros de Datos distribuidos, Sistemas de Red, Seguridad, Backup, etc.) para lograr *comunicación*, que les permita a los clientes interactuar con los servicios.

Debe también tenerse en cuenta la *escalabiliad*, en que el servicio pueda agregar nuevos usuarios rápidamente, y esto hay que tenerlo en cuenta desde el punto de vista del Software y del Hardware, considerando por ejemplo, como uno de los principales puntos, la *elasticidad* para aumentar o disminuir la potencia de los equipos en situaciones de cambios drásticos de demanda.

La *dependebilidad* en que la comunicación y el servicio estén continuamente disponibles, es decir, todos los días las 24 horas es otro de los puntos importantes a considerar. Sin dudas, Internet provee la comunicación perfecta para el SaaS y el Cloud Computing [22].

2.3. La peculiar fase de implantación

Una de las nuevas tareas que incorpora la producción de software en la nube, es la de Implantación, el cual también pasa a ser una actividad de la Empresa de Desarrollo.

La mayoría de las metodologías ágiles existentes no tienen en cuenta esta fase, debido a que las fases que se pueden realizar dentro de ella pueden ser muy variables, o por motivos de que pueden existir una variedad de formas

de llevarlas a cabo, dependiendo del lenguaje o la tecnología que hasta entonces se podría realizar.

En [23] la metodología SCRUM propone solamente las siguientes tres fases:

1) *Fase de Planeamiento* – es subdividida en:

a) *Planeación* – se define el equipo del proyecto, herramientas, el sistema de desarrollo y se crea el product backlog con la lista de requerimientos conocidos hasta ese momento, se definen prioridades para los requerimientos y se estima el esfuerzo necesario para llevar a cabo la implementación de los mismos; y

b) *Diseño Arquitectónico* – se define la arquitectura del producto que permita implementar los requerimientos definidos.

2) *Fase de Desarrollo* – es la parte ágil, donde el sistema se desarrolla en sprints. Cada sprint incluye las fases tradicionales del desarrollo de software – relevamiento de requerimientos, análisis, diseño, implementación y entrega.

3) *Fase de Finalización* – incluye integración, testing y documentación. Indica la implementación de todos los requerimientos, quedando el product backlog vacío y el sistema listo para entrar en producción.

Nótese que la fase de la cual se está discutiendo en este apartado, se encuentra dentro del punto número 2, bajo el simple concepto de entrega, el cual deja percibir que una vez concluido el software éste ya simplemente se entrega al usuario. Para tecnologías como Java EE la fase de implantación es muy importante y debe realizarse considerándose una serie de sub procesos o tareas.

Tomando como base a la metodología XP, se puede hacer la siguiente comparación, nótese que la definición de XP, formulada por Kent Beck, se diferencia del resto de las metodologías por su énfasis en la adaptabilidad. La

metodología está diseñada para ofrecer el software que el usuario necesita, en el momento en que lo necesite. El éxito de la metodología se basa en potenciar las relaciones interpersonales, promoviendo el trabajo en equipo, el aprendizaje de los desarrolladores, y un ambiente de trabajo cordial. Los cinco principios básicos de XP incluyen: 1) Simplicidad – simplificar el diseño para agilizar el desarrollo y facilitar el mantenimiento mediante la refactorización del código; 2) Comunicación – fomenta la comunicación: escrita – como código autodocumentado y pruebas unitarias, recomendando documentar el objetivo de clases y la funcionalidad de métodos; y oral - entre programadores y con el cliente, recomendando que ambas sean constantes y fluidas; 3) Retro-alimentación – promueve la retro-alimentación constante del cliente a través de ciclos de entrega cortos y demostraciones de la funcionalidad entregada; 4) Coraje – para mantener la simplicidad permitiendo diferir decisiones de diseño, para comunicarse con los demás aún cuando esto permita exponer la propia falta de conocimiento y para recibir la retro-alimentación durante el desarrollo; y 5) Respeto – se infunde entre integrantes del equipo – los desarrolladores no pueden realizar cambios que hagan que las pruebas existentes fallen o que demore el trabajo de compañeros, y hacia el trabajo – los miembros del equipo tienen como principal objetivo lograr un producto de alta calidad con un diseño óptimo.

El proceso de desarrollo XP consiste de tres etapas:

1) Interacción con el cliente – el cliente interactúa permanentemente con el equipo de trabajo. Se elimina así la fase inicial de recolección de requerimientos, y éstos se van incorporando de manera ordenada a lo largo del desarrollo. La metodología propone utilizar la técnica de Historias de Usuario mediante la cual el usuario especifica los requerimientos funcionales y no funcionales del producto. Cada historia debe ser lo suficientemente atómica y comprensible, para que los desarrolladores puedan implementar los requerimientos durante una iteración.

2) Planificación del Proyecto – el equipo de trabajo estima el esfuerzo requerido para la implementación de las Historias de Usuario. Cada historia debe ser implementada en un período de una a tres semanas. Aquellas historias que requieran más tiempo se sub-dividen tratando de que resulten atómicas y puedan realizarse dentro del plazo máximo.

3) Diseño y Desarrollo de Pruebas – la implementación está conducida por las pruebas unitarias. Cada vez que se quiere implementar una función, primero se debe definir el test y luego el código para que lo satisfaga. Una vez que el código cumple el test exitosamente, se amplía y se continúa. A medida que se van implementando las Historias de Usuario, se van integrando los pequeños fragmentos de código. De este modo, se realiza una integración continua, evitando una integración más costosa al finalizar el proyecto. XP fomenta la programación por pares, donde el desarrollo es realizado por una pareja de programadores. Las parejas tienen que ir cambiando de manera periódica, para que el conocimiento sea adquirido por todo el grupo de desarrollo.

Se puede observar claramente que para la metodología XP, no existe una definición clara de la etapa de implantación del Software.

2.3.1. El crítico proceso de despliegue

Uno de los momentos más críticos a ser tomados en cuenta a la hora de elaborar el diseño de una aplicación SaaS, de hecho, es el momento de la fase de Implantación en el cual debe ser realizado el proceso de despliegue, ya que el realizarlo exige un muy minucioso estudio sobre la forma que será ejecutada en los servidores finales (de producción), considerando todos los riesgos que pueden darse para disponibilizar el Sistema en la nube, aún más si se trata de que la aplicación que ya se encuentra en funcionamiento.

Como ya fue mencionado, las metodologías ágiles actuales no brindan mucha recomendación al respecto o la definen muy superficialmente. Debe ser

porque el tipo de lenguaje y herramientas que utiliza cada desarrollador pueden variar y considerando también que para algunos casos las actualizaciones se realizan casi de manera instantánea con la sola tarea de copiar archivos, como es el caso de algunos lenguajes.

Para el caso particular de este modelo que es presentado, en el cual se conoce previamente la tecnología y en la cual se sabe que las actualizaciones no se dan con simples copias de archivos, es imperativo brindar una especificación más detallada de este proceso, considerando que debido a las características propias de la plataforma Java EE, a veces las actualizaciones requieren de recarga de la aplicación en el mejor de los casos o incluso de reinicio del Servidor de Aplicaciones en el peor, el cual obliga a parar el servicio por completo por algunos minutos, dejando al cliente sin sistema en dicho momento.

Así es que, entonces, en dicha fase surgen algunas preguntas que deben ser contestadas y que serán cruciales, por ejemplo:

- ¿Cómo realizar una actualización de una nueva versión cuando el sistema se encuentra actualmente en producción?.
- ¿Cómo actualizar la base de datos sin miedo a que estos generen errores en los sistemas que ya estaban funcionando?.
- ¿Cómo revertir una actualización, en caso de fallas?.

Son problemas que deben ser tratados y pensados de antemano.

2.3.2. Interacciones

Sea cual fuere la metodología ágil en particular, el ciclo de vida Ágil [22] involucra:

- Que todos los participantes (usuarios, clientes, grupo de mantenimiento, desarrolladores, operadores y administradores) trabajan en conjunto y de forma continua para especificar los *requerimientos* y los *test*.
- Mantener un prototipo de trabajo mientras se desarrollan nuevos aspectos generalmente cada dos semanas (iteración) y chequear con los participantes para decidir que se agregará la próxima vez y validar que el sistema actual es lo que ellos realmente quieren una iteración en el ciclo [22].

Esto requiere de mucha interacción entre los participantes, para lo cual debería de existir reglas claras y precisas sobre las actividades, de forma a fomentar la brevedad y ser conciso en las reuniones, para llegar con mayor rapidez al resultado, consiste en utilizar plantillas, scripts de ejecución, rutinas automatizadas, etc.

2.4. La solución antes de tiempo

Los programadores tienden a predecir las necesidades futuras e implementarlas antes de que estas ocurran [24].

Estas supuestas mejoras introducidas en el código, muy a menudo son realizadas por el desarrollador de manera individual, escapándose de esa manera de todo control de tiempo y plazos que se tiene previsto para el proyecto.

Para evitar que esto ocurra, es importante contar con reglas claras para el desarrollo, así como también de soluciones listas, ya probadas, e incluso de generadores de código o plantillas predefinidas para los programas y datos, lo cual aumenta su valor cuando el proyecto es realizado de manera compartida entre varios desarrolladores.

Todo esto debe ser realizado por un arquitecto especialista y los desarrolladores más experimentados en la fase de maduración de la Empresa.

De esta manera, se logra que el programador enfoque más su capacidad en la implementación de las funcionalidades del sistema y no en la mejora de los defectos técnicos o correcciones de errores que podrían ocasionar fallas posteriormente.

Según mediciones, esta acción de los programadores es una práctica ineficiente para el equipo, para el proyecto y para la empresa, concluyendo que tan sólo el 10 % de las soluciones hechas para el futuro serán utilizadas, desperdiciando tiempo de desarrollo y complicando el tiempo innecesariamente.

Según la experiencia del autor, el mismo destaca que esto por lo general ocurre mayormente con desarrolladores experimentados con sólidas habilidades y conocimiento sobre las predicciones de usabilidad y experiencia de los usuarios.

Es por eso que en la metodología se recomienda que sea el líder quien detecte las habilidades de los programadores con estas características, para mantenerlos al tanto de la Arquitectura del Sistema con el fin de hacerlos participar en las mejorías y sugerir sobre las innovaciones, así como también para que trabaje por la infraestructura o framework de desarrollo, así cuando esté desarrollando un proyecto en particular, sólo se preocupe de las funcionalidades.

2.5. Refactorización de código con Java EE

Una de las características de las metodologías ágiles en general es la de poder realizar refactorizaciones de código [25]. El diseño del sistema es una tarea permanente que se realiza durante todo el ciclo de vida del proyecto [24, 25]

y es con la refactorización de código que se consigue este objetivo. Cualquier metodología ágil, promueve el inicio del proceso de desarrollo con un diseño inicial, para luego ir adaptándolo de acuerdo a las necesidades de la Aplicación.

Estas prácticas son complicadas de realizar cuando se está empezando a utilizar alguna metodología ágil luego de trabajar varios años con metodologías tradicionales; y los motivos son diversos.

En primer lugar existe por temor que genera en los equipos de desarrollo cambiar algo que ya funciona bien sea a nivel de diseño, implementación, e incluso, en los datos o en su estructura.

Sin embargo si se cuenta con un esquema de pruebas completo y un sistema de automatización para las mismas, es más seguro tener éxito en el proceso. El problema radica justamente allí, ocurre que muchos desarrolladores se olvidan de realizar las pruebas unitarias de sus clases o el equipo de pruebas no automatiza correctamente las pruebas funcionales de tal manera que pueda probar justamente eso, las funcionalidades y no el código.

El otro motivo es la creencia de que es mucho más sacrificio y tiempo realizar la refactorización comparado con la sencillez del mantenimiento para el cual se está pensando. Según XP la ganancia obtenida en la refactorización es tan relevante que justifica suficientemente el esfuerzo extra en corrección de redundancias y funcionalidades innecesarias [24, 25].

En desarrollos Java EE esto se agrava aún más, ya que los mismos precursores de la tecnología indican de que antes de iniciar cualquier nuevo proyecto, se debe primero realizar un análisis profundo del Sistema, ya que cualquier modificación puede afectar muchos cambios posteriormente, mucho más aún sin contar que los cambios recién se verán reflejados una vez realizado el re-deploy de la aplicación, el cual toma su tiempo, es por eso el

surgimiento de nuevas tecnologías similares como Spring, pero no tan robustas cuando Java EE.

En tecnologías como Java EE e incluso en el desarrollo de software en general, existe un temor en tocar las clases, cambiar atributos, o nombres, ya que eso puede generar errores en los lugares menos esperados.

Eso significa que la herramienta en particular, o el entorno de desarrollo debe contar con una interfaz de refactor de acuerdo a la necesidad de la Empresa, por supuesto si no hay debe poder realizarse.

2.6. Experimentar en iteraciones pequeñas

Perfeccionada en el libro de Eric Ries, *The Lean Startup* [15], esta metodología parte de la premisa de que el elemento fundamental para el éxito de una empresa es encontrar un modelo de negocios viable y escalable mediante una serie de experimentos que sirven para aprender sobre el producto. El foco de dichos experimentos es siempre el cliente.

En términos prácticos, esta metodología consiste en el desarrollo rápido de prototipos que sirven para poner a prueba nuestras hipótesis y obtener feedback del mercado, para relacionar a los clientes con el producto, lo más rápido posible [16].

Esto reduce considerablemente los costos monetarios y emocionales de iniciar la construcción de un nuevo producto, ya que desde el primer minuto se itera el producto con el cliente, lo que provee a la organización de un conocimiento temprano del mercado.

Para esto, la metodología Lean Startup usa un ciclo de aprendizaje, que se basa en 3 etapas fundamentales:

Construir: Consiste en construir, a través de metodologías ágiles y lo más rápido posible, un software de calidad (usable y amigable con el usuario) que permita probar y modificar el producto, de acuerdo a la respuesta y a las necesidades del cliente. Aquí es recomendable usar herramientas de software libre y tecnologías en la nube [26], para evitar gastar tiempo en actividades que no estén directamente relacionados con el desarrollo de la empresa.

Medir: Consiste en "salir de la oficina" para probar la hipótesis en el mundo real, observando y analizando el comportamiento del producto en el mercado y la respuesta de los clientes, monitoreándolos y creando pruebas de usabilidad y de escalabilidad.

Aprender: Absorber algunas impresiones sobre el producto, como sus cualidades y defectos; y aspectos a mejorar. En esta etapa, es conveniente usar los medios como por ejemplo las redes sociales, para recibir feedback sobre el producto.

Este ciclo es iterativo, es decir, se genera uno al cabo de cada modificación que sufre el producto. Para ello, es importante que cada ciclo de desarrollo sea lo más corto posible. De esta manera nace el concepto de MVP o producto viable mínimo, el pilar fundamental de la metodología Lean Startup, que consiste en trabajar con una versión refinada del producto (con sus características esenciales ya definidas), con el objetivo de maximizar el aprendizaje del producto y del mercado.

Cada ciclo de construcción, medición y aprendizaje debe constituir un producto entregable.

Uno de los objetivos de los ciclos de desarrollo y de la experimentación es saber cuándo perseverar y cuándo pivotar las opciones del producto, cambiando su dirección. Esto se logra iterando las veces que sea necesario, escuchando a los clientes y mejorando el producto en base a su retroalimentación.

Con la elaboración de este proceso se cubre también el área de mantenimiento, ya que el ciclo iterativo nunca termina y se extiende mientras dure el producto durante toda su etapa de vida, mejorándolo infinitamente.

2.7. Discusiones y comentarios

En los párrafos que describe este capítulo, se puede percibir que, como en toda actividad, el ciclo de desarrollo de un sistema pasa por diferentes tipos de dificultades, desde el inicio de la construcción de un software hasta su culminación.

Algunos de los puntos abordados tales como "La solución antes de tiempo" pueden estar relacionados con la mentalidad del programador o el equipo de desarrollo como un todo, ya que éstos, al estar en el día a día, no están al tanto de la necesidad de urgencia que tiene la empresa en realizar de una manera ágil los proyectos y de entregarlo en el menor tiempo posible, cumpliendo así los plazos y las expectativas del cliente. Es cierto que se trabaja en base a cronograma y plazos planificados pero que comúnmente se ven superados por la falta de una organización de estructura interna.

Otro de los puntos, como es la falta de costumbre por no utilizar herramientas para automatizar, también se deben a este factor, ya que como los analistas-desarrolladores, son personas con conocimientos técnicos, gustan de resolver los inconvenientes que van surgiendo de una manera más artesanal haciendo uso de su propio esfuerzo, no considerando la elaboración de herramientas que mejoren su productividad para futuras situaciones del mismo tipo, aunque también hay que considerar que en una empresa de desarrollo lo que se desea es siempre acelerar el resultado de cada proceso, y por ese motivo casi nunca hay tiempo para elaborar dichas herramientas así como para realizar capacitaciones.

La mayoría de los puntos comentados podrían ser solucionados fácilmente con un mínimo de organización interna, así como la aplicación de una

metodología. Es importante también resaltar aquí la labor que el líder del proyecto debe llevar en frente con el objetivo de concienciar constantemente sobre un sistema basado en resultados.

En lo que respecta a los apartados sobre la refactorización de código, empaquetamiento, despliegue y reload de la aplicación, consisten en situaciones limitantes que se imponen por la propia tecnología, en este caso Java EE y que para sobrellevarlos, habrá que buscar una solución estratégica en sus diferentes etapas.

Como resumen de este capítulo, se observa la necesidad que tiene toda empresa de desarrollo de motivar a los equipos a trabajar con herramientas de apoyo que los auxilie a evitar tareas manuales repetitivas, minimizar errores y aumentar la productividad, contando para eso con software desarrollados por ellos mismos o utilizando los cientos de los ya existentes y en la mayoría open-source. Viendo esa necesidad es que en el Capítulo III se insiste en la fundamentación de 3 pilares de la metodología que nacen como la base para cubrir las expectativas del desarrollo ágil de proyectos con la Plataforma Java EE.

En cuanto a la necesidad de una organización clara de estructura interna, en el Capítulo III también se propone el desarrollo de las fases de maduración de una empresa de desarrollo, de tal forma que basado en ello, todo involucrado comprenda de manera fácil, el mundo en el cual se encuentra inserto, y pueda desempeñar mejor sus funciones, en busca de los logros comunes.

Capítulo III

Metodología propuesta

3.1. Resumen

Este capítulo describe la metodología propuesta de manera sencilla, de forma que cualquier empresa pueda comprenderla fácilmente y llevarla a la práctica.

La metodología está compuesta de varios aspectos que pueden ser tomados de manera independiente para ir implantándolas de a poco realizando un seguimiento continuo de los procesos e ir mejorando día tras día.

A modo de introducir al lector en el contexto general que representa este modelo se menciona que primeramente se hace énfasis en la utilización de 3 pilares para el desarrollo rápido, que se originan a partir del estudio de la problemática existente y que consideran fundamentales para la adecuada implantación de la metodología.

Posterior a la explicación de los pilares, se hace referencia al concepto de etapas y fases, dividiéndolo en dos aspectos claves que son la etapa de organización y visión de una empresa de desarrollo y la fase de desarrollo de un sistema.

Como fue mencionado anteriormente, mantener una estructura organizacional equilibrada es de suma importancia para que los equipos de desarrollo puedan desempeñarse de manera eficiente, por ese motivo en primer lugar se hace referencia a la evolución sistemática por el cual la empresa deberá pasar obligatoriamente, mirando sus resultados que van dejando y verificando puntos que se pueden mejorar.

En un segundo momento se mencionan sobre las fases que se deben considerar en un proyecto de desarrollo para obtener la agilidad que se espera.

Más adelante son mencionados los roles de recursos con los cuales se deberá disponer, mencionando algunas de sus actividades principales, las habilidades y destrezas requeridas por los profesionales que serán asignados, etc.

3.2. Los pilares del desarrollo rápido

Como ya fue mencionado, durante el estudio e investigación de este proyecto 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 son tomados en cuenta en lo largo de su presentación y que se presentan en la *figura 3.2-a*.

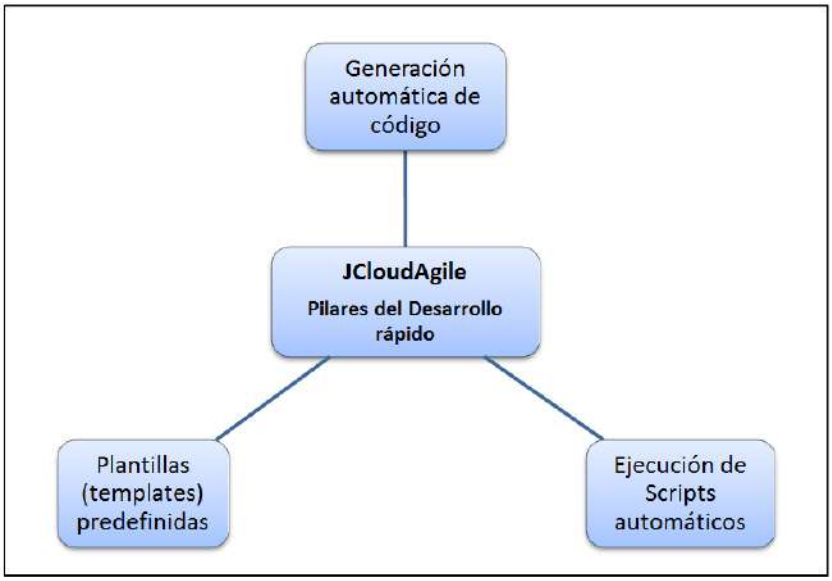

Figura 3.2-a – Pilares del Desarrollo rápido (Fuente propia).

A continuación un resumen estos 3 pilares.

- Generación automática de código inicial.
- La utilización de plantillas (templates) pre-definidas.
- Ejecución de Scripts automáticos.

Para que la aplicación de esta metodología sea exitosa, es necesario tener presente siempre estos pilares. Cada uno de ellos serán mejor detallados a continuación.

3.2.1. Pilar Nº 1 - Generación automática de código inicial

Existen programas que generan el código del software automáticamente sin la necesidad de escritura por parte de los programadores. A este tipo de programas se los denominan auto-generadores de código.

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* 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. El objetivo de un generador de código debe ser siempre optimizar el activo más valioso de la organización: la creatividad y el entusiasmo del equipo de desarrollo.

La auto-generación 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.

La generación automática de código fuente ha sido aplicada exitosamente a diversas actividades, entre las que cabe mencionar: el desarrollo de compiladores, la documentación de sistemas de software, el desarrollo de interfaces gráficas de usuario, el desarrollo de sistemas Web, entre otras [27].

Es impresionante la cantidad de herramientas existentes hoy en día que permiten la auto-generación de código que hacen tan sencilla y llevadera la vida de los programadores, principalmente en los lenguajes de alto nivel, como lo son Java y Java EE.

Ahora bien, para que la auto-generación realmente sea productiva, necesita que se cumplan ciertos factores de manera satisfactoria. Por ejemplo, podría ser bien aprovechado cuando existe una uniformidad de los formularios de la

aplicación, o cuando se estructura de una manera estandarizada sus flujos. Para el caso de la propuesta que se está elaborando esto no será muy complicado, ya que es la propia tecnología Java EE la que muchas veces establece una estandarización de todos sus componentes.

Está claro que habrá funcionalidades muy específicas de cada situación en particular que forzarán a realizar adaptaciones o modificaciones propias en el código de manera manual a lo creado con el generador. La idea no es construir todo el software con un generador y luego implantarlo en el cliente, pensando que todo va a funcionar a la perfección y de que ya no existe nada más manual que hacer, sino simplemente obtener una rápida estructura inicial de la aplicación, que permita a la empresa realizar la entrega inicial al cliente y presentarlo en pocos días o semanas para obtener su feedback y poder realizar adaptaciones en ciclos cortos disminuyendo así el largo periodo de tiempo que puede incurrirse en desarrollar desde cero.

El generador de código es una herramienta que debe crear cada nueva empresa de desarrollo como un producto interno. Debe estar acorde a las especificaciones y arquitectura del tipo de software que se desea crear y todos los actores, lideres, analistas, arquitectos y desarrolladores deben involucrarse en ello, y la mejor hora de desarrollarlo es cuando la empresa está en pleno proceso de desarrollo, es decir, en su fase de maduración.

Para el desarrollo del auto-generador hay que tener en cuenta qué objetos se pueden obtener de manera automática y qué objetos conviene implementarlo de manera manual. También debe pensarse en la posibilidad de ir realizando mejorías de manera gradual así como sería con cualquier otro sistema, hasta obtener finalmente una aplicación completa que aplique la generación automática de código en todos los sentidos.

De esta forma entonces, por ejemplo para un proyecto de auto-generación en Java EE versión 1.0, del relevamiento de datos sobre el Sistema podría obtenerse el siguiente contexto:

- **Creación de Proyectos:** montaje inicial de estructura de Archivos, genera conjunto completo de recursos para crear la aplicación rápidamente.
- **Creación del proyecto EJB**
    - Creación de las Clases de Entidad.
    - Creación de las Clases Facade.
- **Creación del Proyecto WEB**
    - Creación de las clases de Control de Flujo (Actions/Mappings/Navigations, etc.).
    - Creación de archivos de configuración, XML (dependiendo del framework).
- Vista de Formularios, pantallas, páginas, especificación de campos, etc.

Considerando que el arquitecto de la empresa ya se ha encargado de establecer un conjunto mínimo de superclases que se encargarán de la funcionalidad núcleo del sistema, esto no parecería ser una tarea muy complicada, en principio.

En cuanto a la estructura básica requerida para la ejecución del generador pueden utilizarse diferentes estrategias, que ya dependerían de los objetivos de la empresa, puede optarse por ejemplo por autogenerar código directamente de la base de datos, a esto se llama (down to up) de abajo hacia arriba, pero también puede darse la posibilidad de correr el generador de forma (up to down), por ejemplo, desde donde el usuario final realice el diseño de las funcionalidades a través de prototipos estableciendo sus flujos,

y luego el generador se encargaría de crear todos los componentes necesarios hasta llegar a la base de datos.

Para una versión 2.0 por ejemplo ya podría pensarse en la auto-generación de clases para realizar validaciones, o la lógica de negocios específica, de pruebas unitarias, ya sea para probar las clases de negocio o las clases de presentación. Para una versión 3.0 por ejemplo en la documentación automática de las clases o incluso y porqué no decirlo, la documentación completa del Sistema o el manual del usuario y así sucesivamente hasta obtener un buen conjunto de herramientas que funcione como un generador completo que trabaje de manera integrada.

Para llevar a cabo esta tarea, se pueden hacer uso de diversas herramientas disponibles, entre las cuales se encuentran Maven, Ant, Freemarker entre otras.

3.2.2. Pilar Nº 2 - La utilización de plantillas pre-definidas

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].

Como ya se mencionó existen plantillas que pueden ser utilizadas en todo el ciclo de vida del desarrollo, incluso desde el relevamiento, pasando por el análisis, la construcción y pruebas.

En [30, 28], 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.

Es crucial que la Empresa de Desarrollo trate de sacarle el mayor partido a la creación de plantillas para las necesidades que vayan surgiendo teniendo como premisa que su adopción acorta el tiempo de implementación de una funcionalidad en particular, utilizando para ello algunos modelos existentes que serán de gran provecho para el equipo de desarrollo.

En el sentido más general, lo que se quiere dejar por sentado es que cada empresa de desarrollo debe estructurar los elementos iniciales que conforman las actividades diarias en base a plantillas predefinidas a utilizarse para cada acto disciplinar, de la misma forma el de ir incorporando nuevas plantillas de acuerdo a las necesidades.

Para crear un proceso bien definido en cuanto a este pilar, se recomienda que la empresa destine un equipo de personas calificadas, que trabajen ya sea en formato de comisión o grupo y que se reúnan de forma periódica, en principio para establecer estas planillas y luego para analizar las mejorías o incorporar nuevas planillas que sean necesarias.

Esto 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.

En lo que respecta al uso de plantillas, no se habla de algo complicado ni muy científico, si no de seguir simples rutinas que pueden ayudar a la mejora continua de la empresa.

Al definir las plantillas, es muy importante clasificarlos de acuerdo a las diferentes áreas de aplicación o al caso en particular, así como también agruparlos de acuerdo a la fase o situación.

El uso de plantillas dentro de una empresa de desarrollo, facilita también la incorporación de cualquier nuevo miembro al equipo, quien se adapta rápidamente a la forma de trabajo. De esta forma se evita la necesidad de realizar una profunda capacitación sobre las actividades internas de la empresa, ya que simplemente se orienta al nuevo miembro a seguir ciertas plantillas que debe utilizar dependiendo de cada actividad que va a realizar.

Así, la empresa, por ejemplo podrá contar con planillas bien definidas organizadas de la siguiente manera, supongamos para la fase de implementación:

- Fase de Implementación:
    - Elementos de Bases de Datos:
        - Plantilla para creación de estructura inicial de Base de Datos.
        - Plantilla para la creación de índices, cuándo crear y cómo crear.
        - Plantilla para la creación de procedimientos almacenados, cómo y cuándo.

        - Elementos de Dominio.
    - Elementos de la Capa Web:
        - Plantillas de Clases de Navegación.

Estos elementos están citados de manera genérica, desde luego que las plantillas deben especificar un elemento o conjunto bien concreto y con un fin específico identificable a simple vista.

Ejemplo de otras Plantillas que pueden ser creadas

A continuación se citan a modo de ejemplos otras plantillas que podrían ser utilizadas. Estos ejemplos se encuentran agrupados y fueron extraídos tomando como base las actividades mínimas realizables en cualquier empresa de desarrollo de software.

- Planificación:
    - Plantilla de Concepción del Sistema.
    - Plantilla de Historia de Usuario de Negocio.
    - Plantilla Lista Reservada del Producto.
    - Plantilla de Historia de Usuario.
    - Plantilla Lista de Riesgos.
    - Plantilla de Modelo de Negocio.
- Desarrollo del Proyecto:
    - Plantilla de Cronograma de Producción.
    - Plantilla de Plan de Releases.
    - Plantilla de Tareas de Ingeniería.
- Código fuente y Pruebas:
    - Plantilla de Casos de Prueba de Aceptación.
- Base de Datos:
    - Plantillas para Creación de Tablas.
    - Plantillas para Creación de Procedimientos Almacenados.
- Mantenimiento:
    - Plantilla de Gestión de Cambios.
    - Plantillas de Documentación Mínima.

- Plantillas de Diseño.
- Plantillas para Informes.
- Plantillas de Base de Datos.

Además es necesario ordenar las plantillas definidas en un cierto orden, de manera que sirvan como un guión que debe ser seguido de manera secuencial durante todo el transcurso del desarrollo de software [31].

3.2.3. 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 Sistema. 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.

Para ser ágiles en el Desarrollo de Sistemas, es necesario automatizar, porque así se ahorra tiempo, se minimizan los errores y se afianzan las labores, por ejemplo de refactorización, cuando se tienen testes unitarios automatizados que se pueden correr directamente después de un cambio en el proyecto.

En la actualidad existen herramientas muy flexibles adaptables a cada situación, que permite prácticamente diseñar los scripts, de acuerdo a cada

necesidad específica, entre las que se pueden citar el *Ant* y *Maven* dentro del contexto de las aplicaciones Java EE.

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, como por ejemplo, tareas de mantenimiento de clientes, de verificación de estados y sondeos de actividades en el servidor.

En cuanto a este pilar, se recomienda a la empresa crear una comisión o grupo de integrantes, que se encargue de analizar las necesidades para la creación de scripts iniciales, como así también que esté pendiente de forma constante de los scripts que serán necesarios ir incorporando y que sirvan para ayudar en las labores cotidianas, de la misma forma como debe ser creada una comisión para tratar asuntos de plantillas o templates.

Esta comisión, debe estar conformada por un equipo tecnológico de dos o más personas, con experiencia en el campo de desarrollo, es decir, más orientadas hacia el enfoque técnico.

Se recomienda incluir como moderador de la comisión al arquitecto de sistemas, el cual puede ser apoyado con el conocimiento de los desarrolladores y el gerente de configuración y cambios.

Sobre los scripts es muy importante, además de llevar una documentación interna dentro del código del Script, también documentarlos como un proceso, en un documento aparte o en un manual del programador, como así también socializarlos con los demás miembros del equipo, de tal forma de que todos puedan beneficiarse con su uso. La documentación debe principalmente incluir el objetivo del Script, el resultado que arrojará o indicar los diferentes elementos que estarán modificándose, así como también un ejemplo de su uso, mostrando los parámetros y tipos de datos

que deben ser utilizados, la forma de invocación y sus diferentes dependencias, si las posee.

3.3. Enfoque hacia la automatización

Como fue mencionado anteriormente los demás enfoques metodológicos, principalmente 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 script sólamente se considera para el proceso de pruebas y el de las automatizaciones para el despliegue.

En la metodología aquí propuesta, teniendo como base los tres pilares descritos en el apartado anterior, se puede deducir fácilmente que esta metodología tiene un enfoque muy orientado hacia la automatización dejando abierta la idea de aplicar el proceso manual sólo en aquellos procesos donde realmente no pueda ser aplicado un enfoque automático.

3.4. 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.

De hecho, el primer postulado del manifiesto ágil es “Individuos e interacciones sobre procesos y herramientas”.

Cuando la empresa dispone de una buena base de plantillas para los diversos posibles casos, contando además con un generador de código que le permitirá desarrollar una primera versión rápida de sus aplicaciones y además cuenta con una serie de scripts preparados para las acciones más comunes que le ahorran tiempo y le brindan seguridad y tranquilidad, entonces es que los integrantes pueden darse el lujo de dedicarle el tiempo más valioso a los individuos y las interacciones.

No sería lo suficientemente ético de parte de la empresa de desarrollo robar el valioso tiempo al cliente, para traerlo frente al computador, donde los técnicos vayan a programar línea a línea su solución, para lo cual fue contratada. Es crítico para ello contar con soluciones listas tipo piezas de rompecabezas para ir jugando con las diversas opciones, y poder ser así realmente ágil en el desarrollo.

A menudo, cuando no se cuentan con dichos pilares como herramientas internas, lo que tiende a ocurrir, por más de que se esté desarrollando con alguna metodología ágil es que el equipo técnico siempre estará preocupado por si su arquitectura va o no cubrir las necesidades específicas del cliente, desviándose de esa forma del foco principal cuando se reúnen equipo y cliente para dialogar sobre el problema.

3.4.1. Entonces esta metodología es prescriptiva o adaptativa

Si casi todos los elementos están prediseñados para trabajar de forma automática, con la mínima intervención inicial del equipo técnico, entonces podríamos deducir que esta metodología se basa en un gran porcentaje en lo prescriptivo, ya que las reglas del juego, estarían todas especificadas de forma intrínseca ya sea en las plantillas, en el auto-generador o en los scripts automáticos.

Para ello el equipo técnico ha trabajado antes en la solución de manera a tener casi todo listo, cuando se sienten juntos técnicos – clientes para realizar la solución.

De ser así, entonces, surgen las siguientes preguntas:

- ¿No es esta metodología en extremo prescriptiva?.
- ¿Dónde queda la creatividad del programador?.

Para responder a la primera pregunta, se considera el objetivo que persigue esta metodología y que como bien es sabido se orienta al desarrollo de una propuesta para el desarrollo de Aplicaciones Corporativas.

Eso significa que la metodología propuesta debe poder aplicarse para el desarrollo de sistemas que abarcan todas las áreas funcionales de la Empresa con la tarea de ejecutar los procesos de negocio, entendiendo de esa manera que el mismo no puede ser construido a la ligera ya que exige mucha robustez y confiabilidad. Para ello entonces, no se tiene más que la alternativa de seguir las reglas o de apoyarse en herramientas que ya traen dichas reglas embutidas en su lógica interna, que es lo que aquí se recomienda.

En cuanto a dónde queda la creatividad del programador, se recuerda que lo que es automatizado son las especificaciones que se cumplen de manera general, por lo tanto siempre habrá la necesidad del uso creativo de propuestas que puedan realizar tanto arquitectos como desarrolladores para cubrir las excepciones que escapan de lo general.

3.5. 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, pero es imposible eludir que el desarrollo de un producto de software sucede o debe de suceder dentro del contexto de una organización, lo cual no se debe descuidar ya que finalmente se acabarán utilizando recursos para otras áreas de la empresa que también influirá en los resultados.

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].

Está claro que toda empresa de desarrollo necesita tener definido un mínimo de pautas sobre los procedimientos, actividades y recursos que se manejan de manera interna, por ejemplo, con el plantel de personal, así como también de manera externa, como por ejemplo, con la atención al cliente.

Los consejos dados sobre los procesos y actividades relacionados a la estructura organizacional de la empresa no pretenden ser definitorios, sino más bien servir de ejemplo referencial práctico con la finalidad de insistir que cada empresa debe definir a la par su modelo de negocio con el fin de que la implementación de una metodología de desarrollo ágil pueda darse de manera más sencilla.

3.5.1. 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], lo cual resultará en caos y repercutirá negativamente en los resultados de los productos que son desarrollados para los clientes, ganando mala reputación y la pérdida de confianza.

Todo esto puede ser mitigado con un plan de negocio de la empresa a ejecutarse en paralelo con la metodología propuesta, donde se establezca qué tiempo será necesario para lograr estabilizar la estructura organizacional, por ejemplo, un tiempo de doce meses luego de haberse iniciado la empresa, o de que la empresa haya iniciado de implementar esta metodología, con por lo menos 5 proyectos bien concluidos e implantados.

En la *figura 3.5-a* se presenta una propuesta interesante extraído del modelo Lean Canvas, que va a ayudar a lograr éste objetivo.

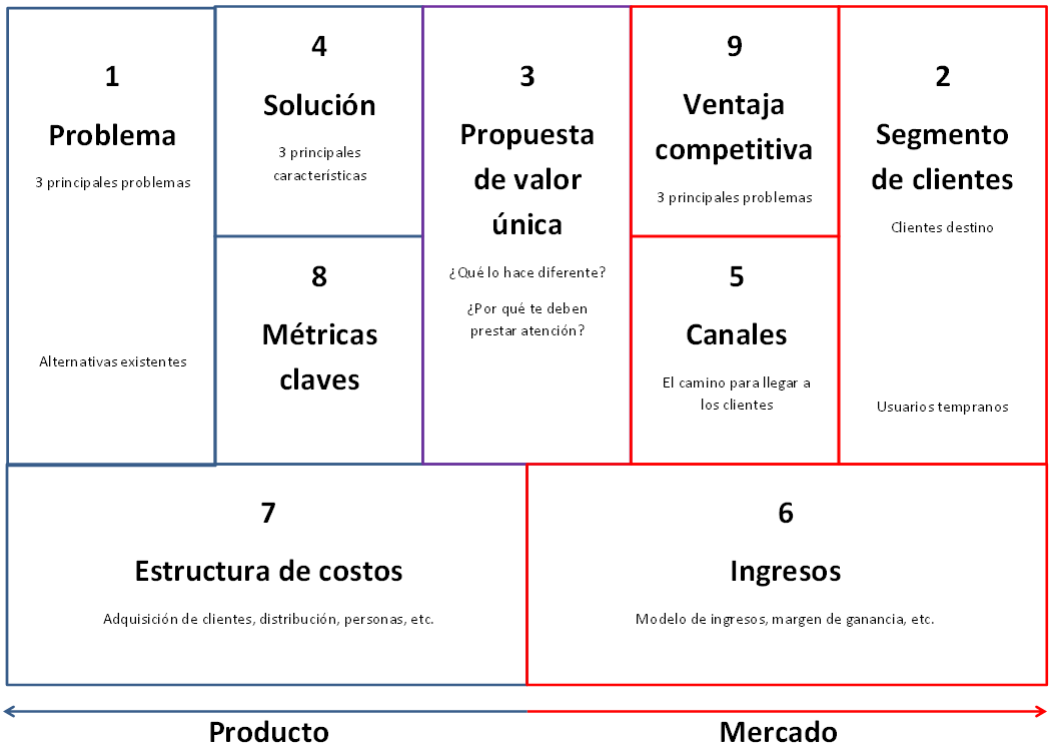

Figura 3.5-a – Bloques del modelo Lean Canvas [16].

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 3.5-b* siguiente:

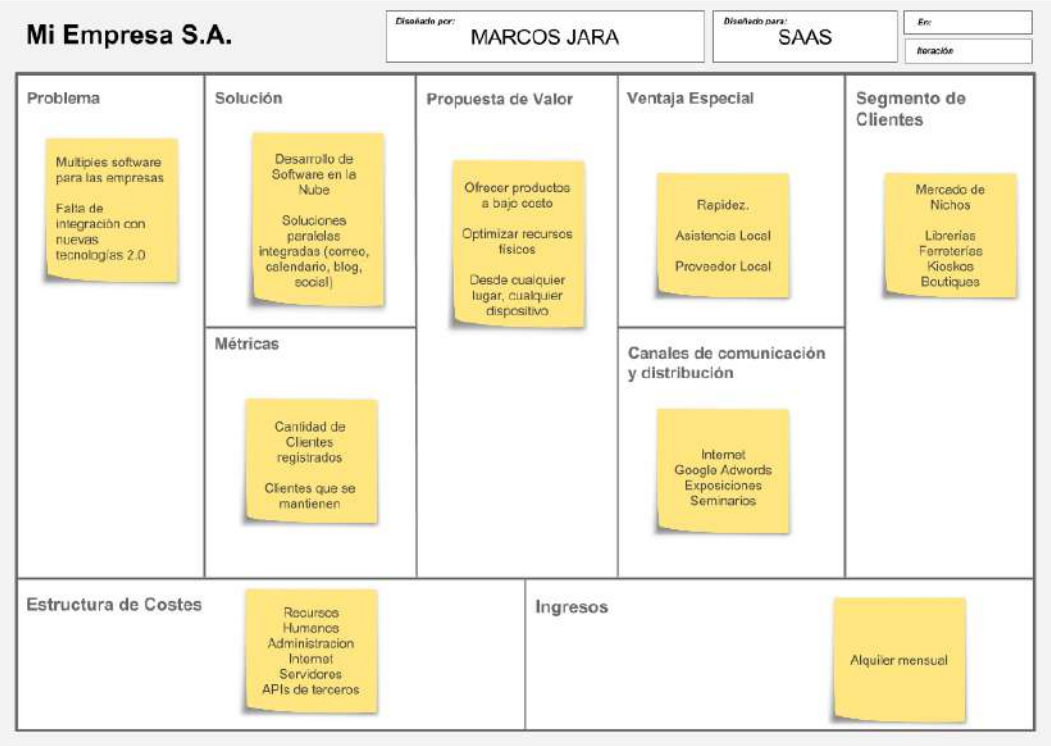

Figura 3.5-b – Modelo contextual de la idea del producto (Fuente propia).

A continuación se pasará a definir brevemente lo que la empresa debería contemplar, se debe tener en cuenta que para completar el lienzo sería conveniente realizarlo en el orden especificado:

1. **Problema:** Desde luego que la idea para crear el software nace para resolver un problema específico que fue identificado previamente. Como hoy en día en la región aún no existen muchas soluciones en la nube, a excepción de aquellas empresas y software internacionales, entonces, a modo de ejemplo uno de los problemas podrían ser que la mayoría de las aplicaciones no tienen integración con tecnologías 2.0 o Internet. Otros de los inconvenientes es que hoy en día una empresa puede estar contando con varios software que deben ser instalados en las computadoras y que sirven para diferentes áreas o tienen fines diferentes.
2. **Segmento de clientes:** Se especifica inicialmente un segmento de clientes bien definido con el cual se desea iniciar, en este caso sería un mercado de nichos, librerías, ferreterías, kioskos, boutiques.

3. **Propuesta de valor:** Su negocio seguro y controlado las 24hs desde cualquier parte.
4. **Solución:** Esto se traduce en las funcionalidades que tendrá el producto desarrollado, las funcionalidades deben cubrir los problemas mencionados en el punto 1.
5. **Canales de distribución y comunicación:** En este caso se especifican el medio por el cual se dará a conocer el producto, en este caso: Internet, Google Adwords, Exposiciones, Seminarios.
6. **Recursos clave:** Este bloque tiene mucha relación con los roles necesarios para el desarrollo de un software, citados más adelante. Debe especificarse en esta sección los recursos de los cuales la empresa no puede prescindir, como por ejemplo un analista especializado, arquitecto especializado, además por supuesto de los recursos no orientados específicamente al desarrollo del producto.
7. **Ingresos:** Alquiler mensual.
8. **Estructura de costes:** Toda empresa cuenta con costos fijos y variables necesarios para subsistir que deben ser considerados [33], y una empresa de desarrollo por tratarse del rubro de servicios no está exento de ello. El empresario debe en este apartado considerar todos los costes que demandará una inversión como lo son los recursos humanos, administración, Internet, servidores, APIs de terceros.
9. **Ventaja especial:** Debe tenerse en cuenta que al proveer un producto al mercado no será el único que estará proveyendo dicho servicio, es por eso que en este bloque debe especificar lo que lo hace diferente de la competencia.

Como se puede observar en el modelo de negocios presentado, en el mismo se hace referencia al producto desde un punto de vista más empresarial, de marketing y enfocado hacia el cliente, motivo por el cual se constituye en un complemento ideal para la metodología que está siendo propuesta.

Más adelante se brindarán detalles específicos sobre las fases del desarrollo ágil de un sistema o producto, pero la idea de esta propuesta de organización interna de la empresa consiste en agilizar aún más el desarrollo y disminuir los tiempos de finalización, ya que trata de automatizar todas las tareas que sean posibles y que en un momento dado la empresa alcance su maduración donde los costos serán más reducidos, ya que se irá aprendiendo procesos repetitivos de los sistemas en particular, e implementando mejoras en los siguientes procesos de forma reiterativa.

3.5.2. Ciclo iterativo para el desarrollo del producto

En la *figura 3.5-c* 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.

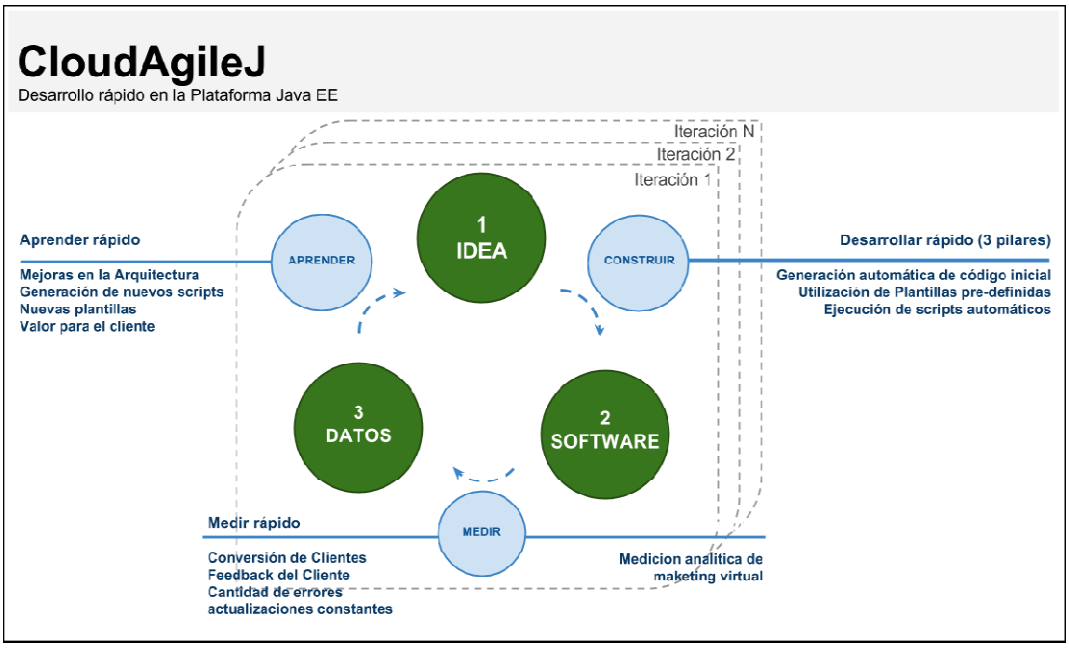

Figura 3.5-c - Ciclo iterativo para el desarrollo del producto (Fuente propia).

En la *figura 3.5-d* 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).

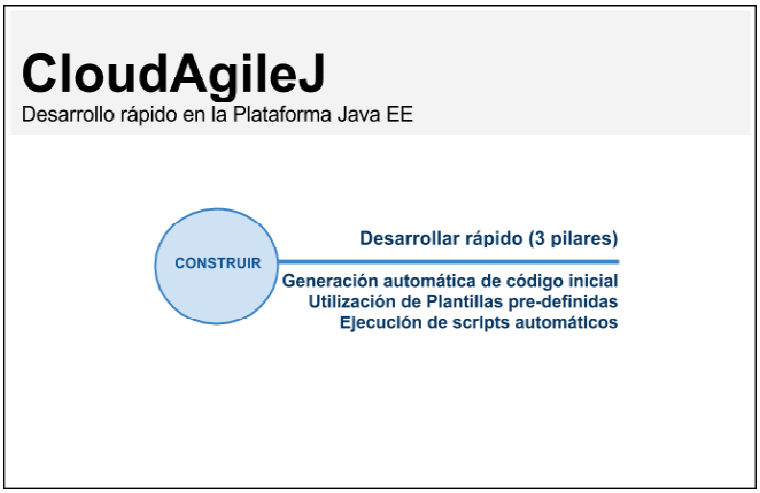

Figura 3.5-d – Nodo *construir* basado en 3 pilares (Fuente propia).

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.

Como se indica en la *figura 3.5-c*, luego de la *idea*, viene la *construcción*, donde debe obtenerse un producto terminado o una versión funcional, seguidamente habrá que ponerlo a prueba con los usuarios, donde habrá que realizar las validaciones sobre qué puntos en el sistema se deben mejorar, e incorporarlo en una nueva versión, con lo cual todo el ciclo empieza nuevamente.

3.5.3. 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 *2 – Generación automática de código inicial*, pero no se debe contemplar a la auto-generación de código como la única herramienta que ayudará a desarrollar rápido las aplicaciones, ya que también existen otros dos pilares muy importantes.

Este sería el segundo paso de la fase de construcción, el de desarrollar dichas 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.

En cuanto a plantillas se refiera, estamos hablando de prácticamente todo lo que pueda ser automatizado, sea código fuente de programas, formularios de presentación, páginas web, archivos de configuración, flujos de navegación, e incluso proyectos de base de datos.

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.

Apache Ant es una herramienta usada en programación para la realización de tareas mecánicas y repetitivas, normalmente durante la fase de compilación y construcción (build).

Cabe destacar que el *Apache Ant* es una herramienta de bastante tiempo atrás que ha ido evolucionando y acompañando las diferentes versiones del Java que fueron apareciendo, y que hoy en día sirve para muchas tareas más, pues como es un producto que permite extender sus funcionalidades, muchos colaboradores de la comunidad han ido incorporando ideas y funciones que posibilitan una infinidad de actividades.

Por lo tanto, a la definición anterior, faltaría agregarle uno de los puntos más importante que es su enfoque hacia la fase de despliegue, ya que como se verá más adelante una de las fases principales, en la cual menos intervención humana debería existir y que en lo posible debe ser completamente automatizada es la de despliegue del proyecto, el cual incluyen actualizaciones de versiones, releases, etc.

Pero el Ant, no es la única herramienta de la que dispondrá e 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.

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.

Para ello debe haber un equipo que constantemente esté acompañando la implementación de las actividades, de tal forma a planificar la automatización de este tipo de tareas, generando programas de generación automática de código, talvéz, a travéz de metadatos, o con ingeniería reversa o alguna otra opción. En ese caso la empresa en la fase de maduración pasará de realizar los software acostumbrados a desarrollar programas que generan programas, mejorando también así la productividad.

Los patrones de diseño son un claro ejemplo de este tipo de situaciones, aunque no consistan precisamente en generadores de códigos, se clasifican como trozos de código pre-establecidos que los desarrolladores lo almacenan en sus bases de datos de buenas soluciones para re-utilizarlos cuando se encuentran con otro problema similar ya sea de programación o de diseño [36].

Una mayor aproximación a lo comentado se puede notar en los generadores de código de mapeo objeto-relacional, que genera código fuente a partir de la base de datos.

En cuanto a herramientas disponibles, en lo relacionado al software libre también se disponen de diversos software y tecnologías que permiten estas acciones, se pueden citar entre ellas al Freemarker que a travéz de un modelo puede generar código fuente, entre muchas otras.

Debe tenerse claro los objetivos y las metas de la empresa, para que esta actividad no solape 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, es por eso que tanto los scripts de ejecución como las herramientas de generación de código producidas, deben producirse en escalas pequeñas (pequeñas funciones que obedezcan a un objetivo), y ser probadas de forma granular conforme avanzan los proyectos, por qué no decirlo a partir del segundo o tercer proyecto.

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.

En esta parte, no se debe perder el punto de vista de las metodologías Lean: *construir*, *medir*, *validar*.

3.5.4. Fase de desarrollo de un sistema

Este apartado está dedicado a definir las diferentes fases por la que atraviesa la construcción de un sistema informático teniendo en cuenta las diferentes actividades principales, considerando su ciclo de vida, desde su inicio hasta la obtención de una versión funcional desplegada en la nube y disponible para su uso por parte del usuario.

En cuando a las fases propias del desarrollo de un sistema, la propuesta es simple, pero 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 3.5-e* se puede observar el modelo de fases propuesto:

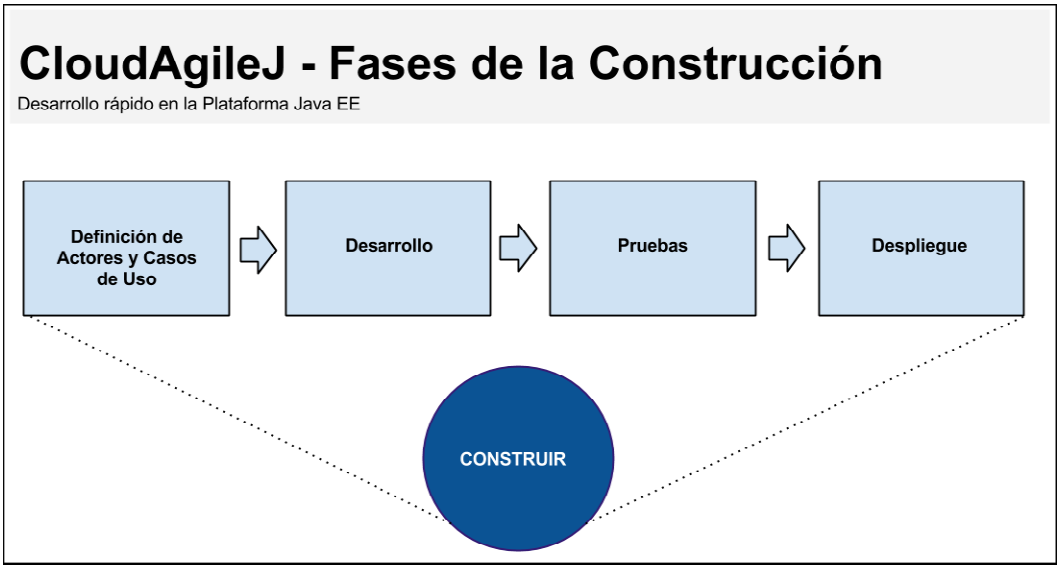

Figura 3.5-e – Fases del desarrollo de un sistema (Fuente propia).

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: La mayoría de los estudios realizados sobre las fases previas al desarrollo hacen constatar que las actuales metodologías dejan una fuerte dependencia sobre las planificaciones, lo cual dificulta posteriormente su adaptación a las necesidades cambiantes de los clientes [37]. 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.

La definición inicial clara de los actores y su clasificación es importante, ya que dichos datos se utilizan para obtener la magnitud del sistema, el cual se verá más adelante.

Para listar los actores del sistema, puede utilizarse como ejemplo el contenido de la *tabla 3.5-a*:

Lista de Actores	**Iteración: N**	**Fecha: dd/mm/aaaa**
Nro.	**Nombre del Actor**	**Tipo**
1		Otro sistema mediante una API
2		Otro sistema mediante protocolo
3		Usuario por medio de una interfaz
N		--

Tabla 3.5-a – Definición de los actores (Fuente propia).

Es importante completar el nombre del actor, por ejemplo: secretaria, gerente, administrador, sistema, etc., y el tipo al menos en ésta etapa, más informaciones sobre cada actor será complementado en la etapa de realización de la estimación de la magnitud del software.

Seguidamente se tiene la segunda parte, la cual consiste en especificar los casos de uso del sistema, cabe recordar que toda la fase de construcción se va realizando en diferentes iteraciones, lo cual significa que 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. Estas funcionalidades una vez implantadas producirán lo que podría ser denominado el producto mínimo, requerido para servir de base para las demás solicitudes que puedan surgir.

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 3.5-b* se muestra una plantilla sencilla que servirá para este fin:

Lista de Casos de 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

Tabla 3.5-b – Definición de la lista de Casos de uso (Fuente propia).

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, un modelo de pantalla donde se mostrará los campos y acciones, servirá para entender el flujo y las iteraciones que tendrán.

Se recomienda que este prototipo debe ser elaborado al estilo lápiz y papel, pues por el medio del camino surgirán modificaciones resultantes del trabajo realizado o de las reuniones con los usuarios y clientes, de esa forma podrá borronear, copiar, añadir o quitar campos fácilmente y acompañar el desarrollo de las fases en todas sus interacciones.

Se puede decir que ésta es la fase que requerirá más dedicación del equipo de trabajo y del cliente, pues esta fase no puede ser automatizada, ya que requiere del conocimiento, la experiencia y las decisiones que sean tomadas en base a conversaciones, reuniones y observaciones que deben ser realizadas.

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.

En la primera iteración se debe lograr disminuir a máximo el tiempo de entrega, con el objetivo de disponibilizar al cliente un producto mínimo, inicial y funcional, de manera a obtener el feedback.

Para lograr una rápida producción del producto mínimo funcional, se debe hacer uso del auto-generador 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.

Como ya se mencionarán en los roles, cada uno de los procesos de automatización que serán necesarios deberán ser realizados por los profesionales competentes y probados y validados por los demás miembros del equipo.

3.6. Roles

Como en cualquier proceso de desarrollo de algún producto o proyecto, existen actividades que deben ser realizadas, es conveniente también tener bien definidos los roles de las personas que estarán colaborando con la ejecución del mismo. En este caso 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.

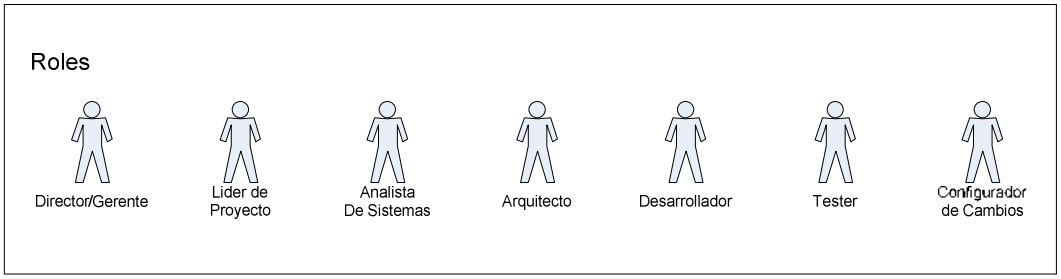

Figura 3.6-a – Roles sugeridos por el modelo (Fuente propia).

En la *figura 3.6-a* 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.

3.6.1. 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.

Actividades

- Analiza presupuestos, costos y valores asignados a las métricas del sistema.
- Conoce y gerencia el o los proyectos de la empresa a un nivel macro.
- Organiza, y coordina las actividades de las demás áreas estratégicas como el marketing y publicidad, ventas, captación de nuevos clientes, etc.
- Promueve soluciones innovadores relacionadas al producto.
- Realizar llamadas a los clientes para ventas y pos ventas.
- Estudia las necesidades de los clientes y factibilidad de nuevos proyectos.
- Estimación y elaboración de presupuestos.

Habilidades y destrezas

Como fue mencionado, este rol no precisamente debe ser cubierto por un capacitado técnico que domine los códigos fuentes, programas o bases de datos aunque sí deba conocerlo ya que debe manejar perfectamente las

ventajas y beneficios de los productos desarrollados para poder dialogar de un manera más profunda y en un mismo lenguaje con el cliente.

Muchas veces estas actividades las realiza el propietario mismo o uno o más socios de la empresa, quienes son los que poseen la visión de la organización, es decir, a dónde se quiere llegar y sabe con qué recursos se cuenta en todo momento.

3.6.2. 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.

Actividades

- Encargado de la planificación del proyecto específico o los proyectos asignados bajo su responsabilidad.
- Gerencia los recursos y tiempos de cada proyecto.
- Organiza las tareas no planificadas y las delega.
- Media los conflictos que surgen entre compañeros de trabajo.
- Fomenta la motivación en el equipo.
- Organiza las reuniones de seguimiento y avance.
- Analiza riesgos y busca estrategia para mitigarlos en caso de que ocurra.
- Sirve de nexo de comunicación entre los equipos de trabajo y la alta gerencia o directivos de la organización.
- Verificar el cumplimiento de los procesos.

Habilidades y destrezas

La persona que desea cubrir este puesto debe estar dotada de todos los conocimientos técnicos necesarios para encaminar a su equipo, así como también, sin lugar a dudas a dominar el arte de la Administración, en relación al planeamiento, dirección y control ya que ese será su trabajo. Además conviene que sea una persona excelente con cualidades de comunicación, ya que su trato es con la alta gerencia y con los equipos, además de saber realizar presentaciones magistrales.

3.6.3. 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.

Actividades

- Realiza el relevamiento de datos sobre la empresa del cliente ya sea para el desarrollo de un nuevo Sistema o para realizar modificaciones en Sistemas existentes.
- Análisis de las necesidades y conversión en requisitos para el sistema.
- Especificación de los requisitos funcionales y no funcionales.
- Prepara los documentos especificados en la metodología.
- Conoce el dominio de la aplicación.
- Realiza los prototipos de pantalla.

Habilidades y destrezas

- Habilidades de comunicación.
- Poder de convencimiento.

3.6.4. 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 embuida 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.

Actividades

- Definir, diseñar, implementar y mantener la arquitectura (software, scripts, programas automáticos, plantillas, etc.) a ser utilizada durante el proceso de desarrollo.
- Construye prototipos previos para realizar pruebas que aseguren la correcta implementación de una determinada funcionalidad o aspecto.
- Define los lineamientos de diseño e implementación en base a documentos que debe elaborar para el mismo.

El arquitecto es considerado muchas veces como un experto de las tecnologías, y es correcto que sea así, ya que genera confianza en el equipo y los desarrolladores, su presencia es importantísima en los proyectos de desarrollo para asegurar el éxito.

Habilidades

- Muy buena formación técnica.

- Contar con varios años de experiencia en los lenguajes de programación y sistemas computacionales, al menos de 10 años.
- Autodidacta.

3.6.5. 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.

Actividades

- Implementa los casos de uso.
- Codificar los componentes de la aplicación.
- Crear y ejecutar los testes unitarios.
- Documentar las clases que ha desarrollado.
- Actualizar la codificación necesaria de acuerdo al control de versiones.

Habilidades

- Poseer amplio conocimiento de las herramientas de desarrollo y lenguajes de programación.
- Sólida base de conocimiento sobre administración y configuración de base de datos.

3.6.6. 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.

Actividades

- Realiza los casos de pruebas funcionales del sistema, de acuerdo a la especificación de los casos de uso.
- Automatiza, en lo posible dichas pruebas, para que puedan ser ejecutadas una y otra vez.
- Analiza criteriosamente y en detalles los datos que son ingresados y que emite el sistema, con el fin de detectar validaciones no realizadas o incoherencia en la información.

Habilidades

- Manejar las técnicas de pruebas de sistemas, métricas de software y aseguramiento de la calidad.
- Conocer profundamente el sistema desarrollado así como los objetivos de cada requisito considerado.
- Conocer los lenguajes de programación ya que deberá programar los testes automatizados.

3.6.7. 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.

Actividades

- Preparar el ambiente necesario para el servidor del sistema disponibilizando las herramientas y el software para el mismo.
- Preparar un ambiente para las pruebas del sistema, con las mismas características del Servidor real, aunque tal vez con menos capacidad

- Preparar el ambiente para la mejora continua.
- Preparar scripts de actualización del sistema y de la base de datos, considerando que la mayor parte del trabajo se realice sin la intervención humana.
- Preparar scripts para crear nuevos proyectos.
- Deshacer cambios en el servidor, y preparar planes de contingencia, en el caso de que se haya actualizado indebidamente el servidor o se encontraron fallas luego de una actualización.

3.7. Integración de los diferentes roles

Como se pudo notar en el apartado anterior, 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. Por ello es importante elaborar previamente al inicio de actividades de la empresa, una lista detallada de tareas generales que debe ejecutar cada rol, indicando el compromiso, la obligación y la responsabilidad que tienen cada uno de ellos, de manera que todo quede más organizado.

3.7.1. Implementación de roles de forma gradual

Es importante para el o los propietarios de la empresa de desarrollo saber ubicarse en el contexto actual donde se encuentra, para poder de acuerdo a eso seleccionar una de las 3 propuestas de implementación de roles presentadas a continuación:

Propuesta 1 – Inicio de Operaciones

Si se trata de una empresa que está recién iniciando o abriendo sus puertas al público, y digamos, ya cuenta con al menos 1 o 2 proyectos en ejecución, lo que recomendamos es contar con al menos 3 personas que se involucren de manera permanente en el proceso cubriendo todas las necesidades previstas, de forma que ninguna tarea o actividad quede huérfana.

Así se tendría una organización de acuerdo a la *figura 3.7-a* presentada a continuación:

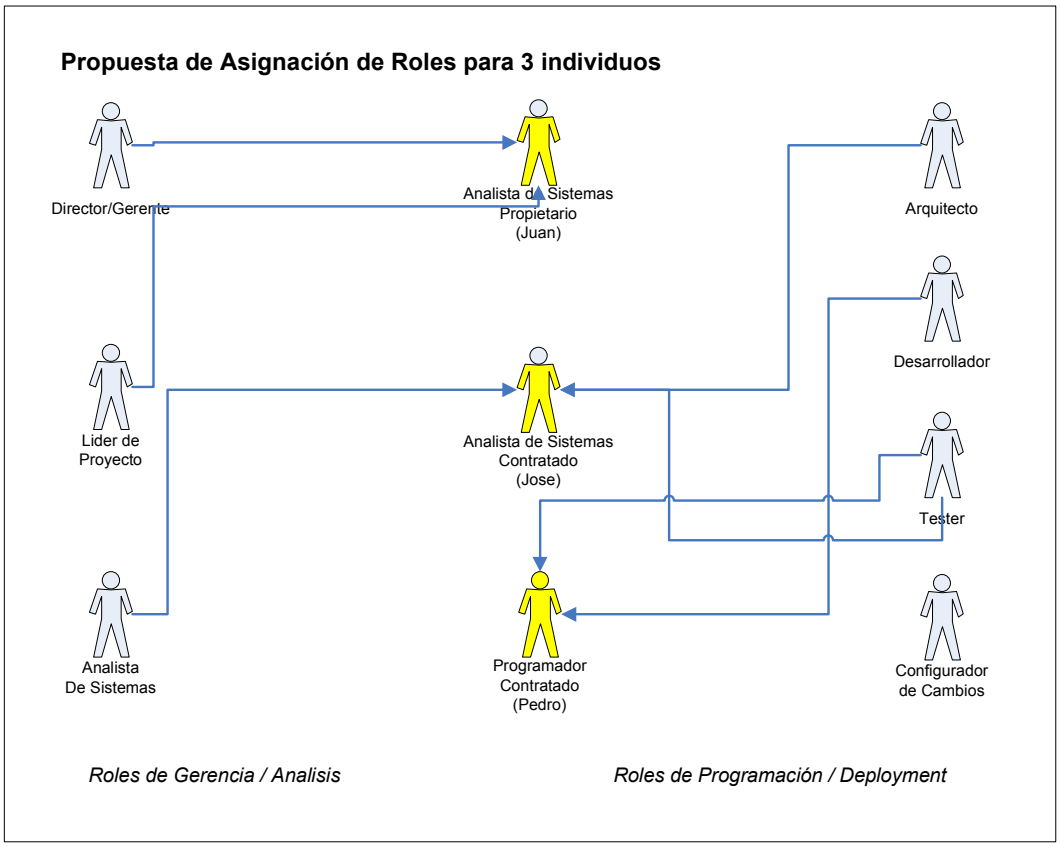

Figura 3.7-a – Propuesta de asignación de roles de 3 individuos (Fuente propia).

De acuerdo al esquema propuesto de 3 integrantes, en la *figura 3.7-a* se puede observar que es el propietario mismo quien cumple el papel del director de la empresa y de líder de proyecto, según la definición de roles, se puede notar que existen varias afinidades entre las actividades y habilidades de éstos roles.

La recomendación es que en este punto y al iniciar las actividades se contrate a un analista de sistemas con experiencia en el desarrollo de sistemas y a un programador, en lo posible también con experiencia.

Está claro que no será lo mismo la ejecución de las actividades de manera óptima, ya que principalmente en las tareas de arquitectura y gerencia de Cambios existe un gran trabajo inicial que debe ser elaborado, el cual será un poco descuidado si se adopta esta propuesta.

Propuesta 2 – Situación normal

En el caso de que la empresa ya se encuentre más consolidada, y en el momento se encuentra desarrollando uno o más proyectos de gran envergadura o contratos más grandes, puede optar por contratar una persona por cada rol, tal como lo muestra la *figura 3.7-b*.

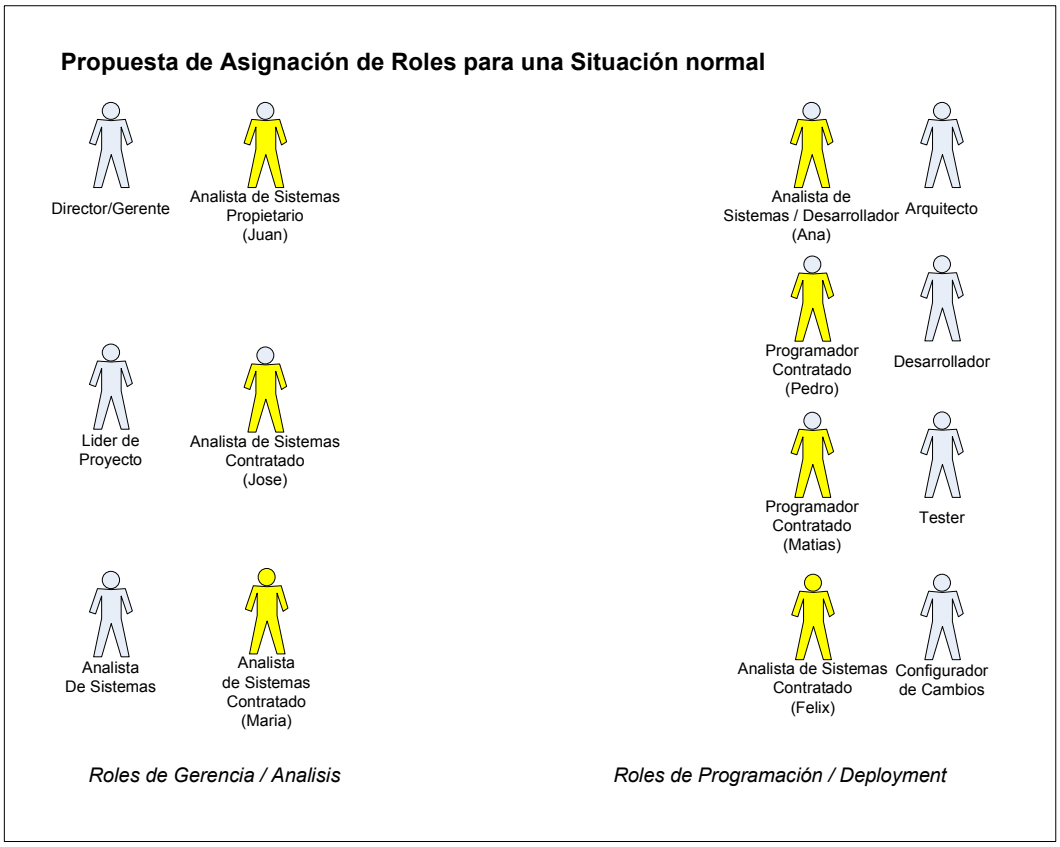

Figura 3.7-b – Propuesta de asignación de roles normal (Fuente propia).

Propuesta 3 – Estructura en crecimiento

En el caso de que la empresa ya se encuentre consolidada en el mercado y ha afianzado en su relación con sus clientes, se puede iniciar a pensar en una expansión en relación a la cantidad de proyectos asumidos.

En este momento se encuentra desarrollando uno o más proyectos de gran envergadura y con contratos grandes en marcha.

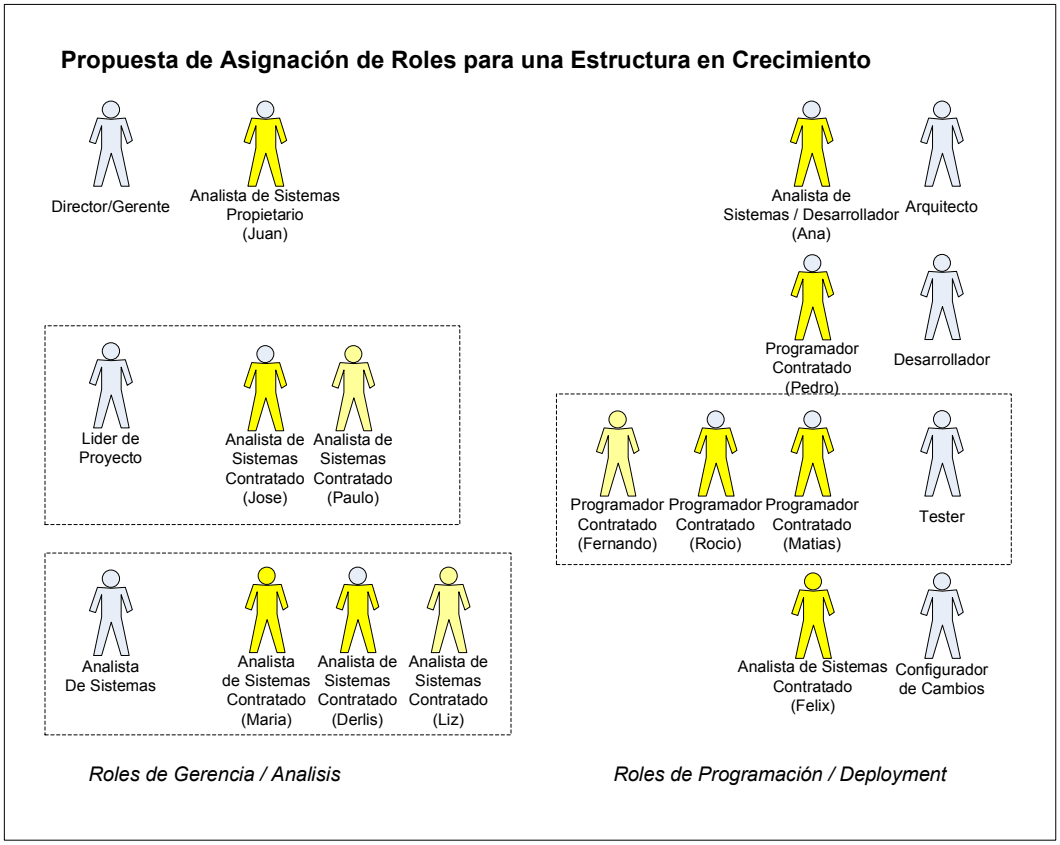

Figura 3.7-c –Roles para una estructura en crecimiento (Fuente propia).

Nótese que en la *figura 3.7-c* se puede observar que los roles que se expanden son el del líder del proyecto, el de analista de sistemas y el de programador, debido a que son los recursos que más demandan actividades por ser realizadas.

Hay que tener en cuenta también de que la proporción para incrementar el individuo para el rol de analista con el del programador es de 1 a 1, ya para incrementar el individuo para el rol del líder aumenta 2 a 3.

Para un líder, como su labor es más gerencial, y si utiliza las herramientas adecuadas, puede llevar hasta un total de 5 proyectos en paralelo, cada uno en sus diferentes fases o en fases iguales, inclusive.

Ya para un analista esa labor se hace más complicada en cuanto a la cantidad, ya que este por lo general estará atrapado en la rutina de realizar reuniones y de actualizar documentos, que le consumirán mucho tiempo.

3.8. Actividades por roles

Está claro que las actividades, a partir de este punto está bien delimitada por los roles asignados a cada persona, es importante tener en cuenta que no significa que una persona que hoy está realizando la actividad A, mañana no pueda realizar una actividad B, ya que una de las propuestas es que los individuos vayan rotando sus roles, de forma que el analista o líder descubra las cualidades específicas de cada uno.

Se presenta a continuación las actividades de cada rol, iniciando en la *figura 3.8-a* con el rol de director/gerente.

Actividades del rol de Director/Gerente

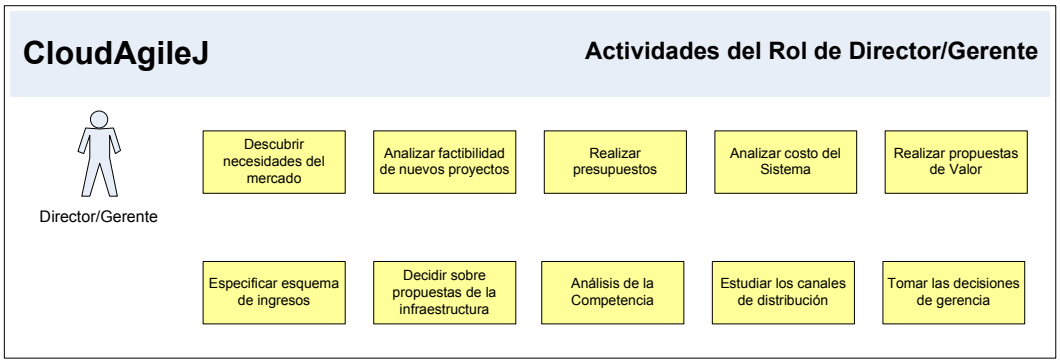

Figura 3.8-a – Actividades del rol de Director/Gerente (Fuente propia).

- Descubrir necesidades del mercado.
- Analizar factibilidad de nuevos proyectos.
- Realizar presupuesto.
- Analizar costo de sistema.
- Decidir sobre propuestas de infraestructura.
- Especificar esquema de ingresos por uso de software.

- Decidir sobre propuestas de infraestructura.
- Analizar a la competencia y descubrir la ventaja especial de su producto.
- Estudiar los diferentes canales posibles para distribuir el producto.
- Tomar decisiones de gerencia relacionados al proyecto.

Actividades del rol de Líder de Proyecto

En la *figura 3.8-b* se presentan las actividades del rol de líder de proyecto.

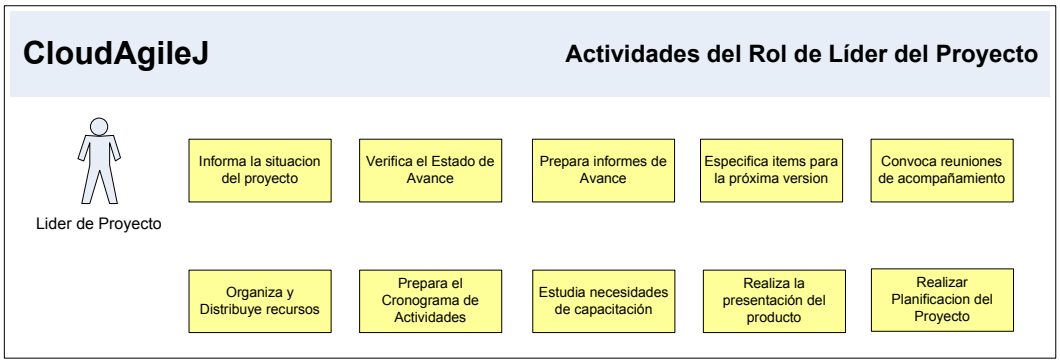

Figura 3.8-b – Actividades del rol de Líder de proyecto (Fuente propia).

- Informe la situación del proyecto.
- Verifica el estado de avance.
- Prepara informe de avance.
- Especifica funcionalidades para la próxima iteración/versión.
- Convoca reuniones de acompañamiento.
- Organiza y distribuye los recursos.
- Prepara el cronograma de actividades.
- Estudia necesidades de capacitación.
- Realiza la presentación del producto.
- Realiza la planificación del proyecto.

Actividades del rol de Analista

En la *figura 3.8-c* se presentan las actividades del rol de analista.

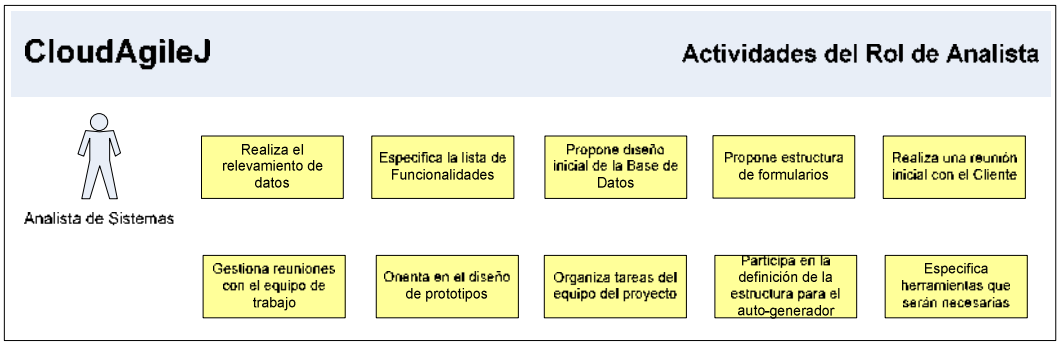

Figura 3.8-c – Actividades del rol de Analista (Fuente propia).

- Realiza el relevamiento de datos.
- Especifica la lista de funcionalidades.
- Propone el diseño inicial de la base de datos.
- Propone la estructura de formularios.
- Realiza la reunión inicial con el cliente.
- Gestiona reuniones con equipo de trabajo.
- Orienta en el diseño de prototipos.
- Organiza tareas del equipo del proyecto.
- Participa en la definición de la estructura para el generador de código.
- Especifica herramientas que serán necesarias (de los pilares).

Actividades del rol de Arquitecto

En la *figura 3.8-d* se presentan las actividades del rol de arquitecto.

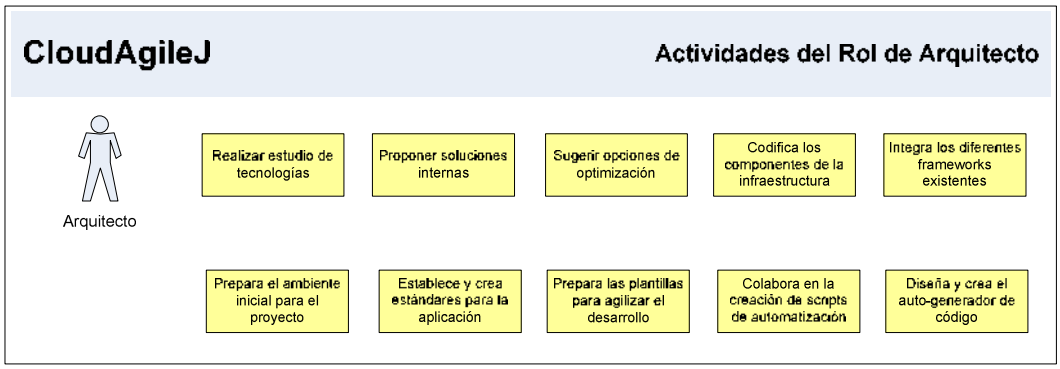

Figura 3.8-d – Actividades del rol de Arquitecto (Fuente propia).

- Realizar estudio de tecnologías.
- Proponer herramientas internas.
- Sugerir opciones de optimización.
- Codifica los componentes de la infraestructura.
- Integra los diferentes frameworks existentes.
- Prepara el ambiente inicial para el proyecto.
- Establece y crea estándares para las aplicaciones.
- Prepara las diferentes plantillas para agilizar el desarrollo.
- Colabora en la creación scripts de automatización de tareas.
- Diseña y crea el auto-generador de código.

Actividades del rol de Desarrollador

En la *figura 3.8-e* se presentan las actividades del rol de desarrollador.

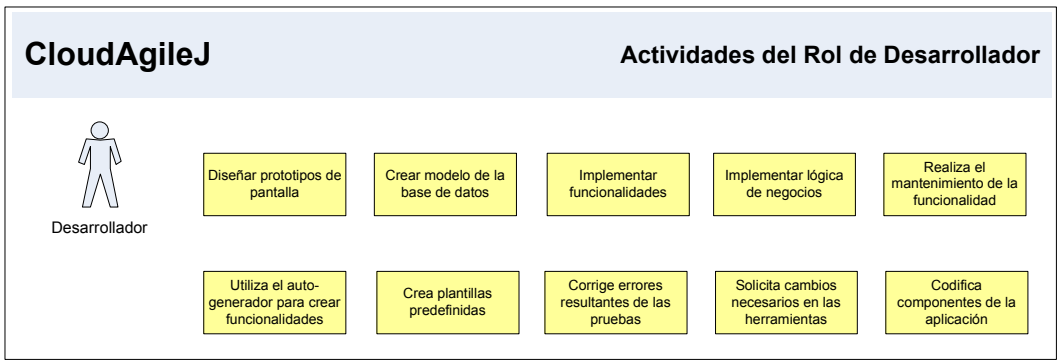

Figura 3.8-e – Actividades del rol de Desarrollador (Fuente propia).

- Diseñar prototipos de pantalla.
- Implementar, crear el modelo de la base de datos.
- Implementar funcionalidades.
- Implementar lógica de negocios.
- Realiza el mantenimiento de la funcionalidad – refactoring.
- Utiliza el auto-generador para crear la funcionalidad inicial.
- Crean plantillas predefinidas para su uso posterior.
- Corrige errores resultantes de las pruebas.
- Solicita recomienda cambios en las herramientas, scripts y auto-generador
- Codifica los Componentes de la Aplicación.

Actividades del rol de Testers

En la *figura 3.8-f* se presentan las actividades del rol de tester.

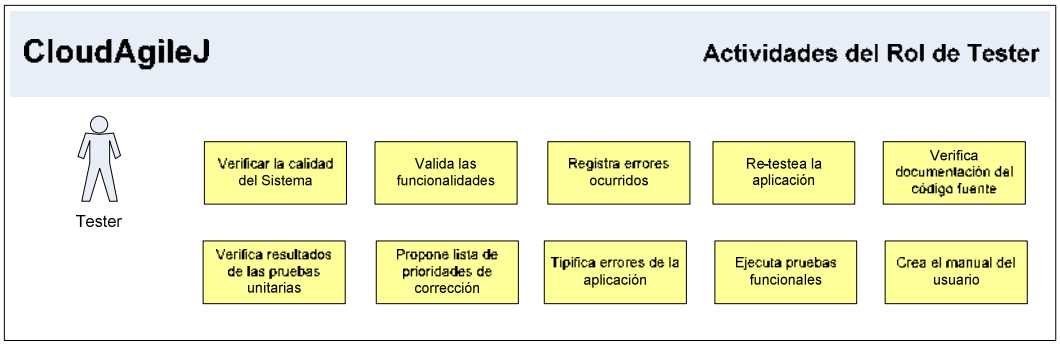

Figura 3.8-f – Actividades del rol de Tester (Fuente propia).

- Verificar la calidad del sistema.
- Valida las funcionalidades.
- Registra los errores ocurridos.
- Re-testea la aplicación.
- Verifica documentación del código fuente del producto.

- Verifica resultados de las pruebas unitarias.
- Propone lista de prioridades de corrección.
- Tipifica errores de la aplicación.
- Ejecuta las pruebas funcionales automatizadas.
- Crea manuales del usuario.

Actividades del Rol de Configurador de Cambios

En la *figura 3.8-g* se presentan las actividades del rol de configurador de cambios.

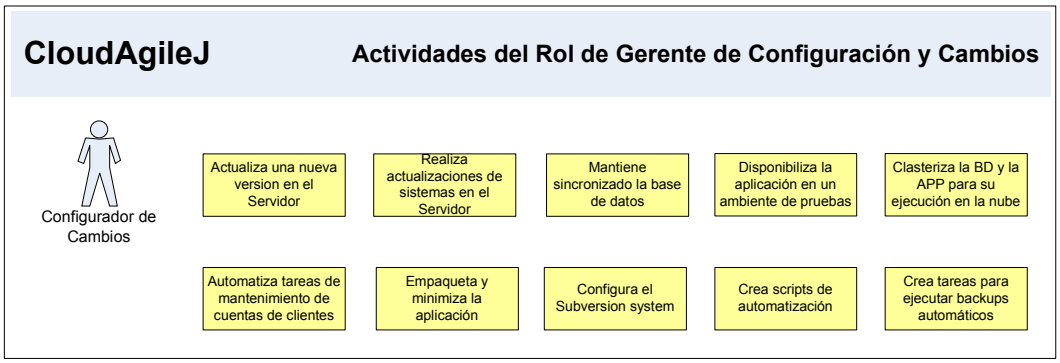

Figura 3.8-g – Actividades del rol de configurador de cambios (Fuente propia).

- Configura la infraestructura del servidor.
- Realiza las actualizaciones en el servidor.
- Mantiene sincronizado la base de datos.
- Disponibiliza la aplicación para la versión de pruebas.
- Realiza el cluster de la base de datos y el sistema para su ejecución en la nube.
- Automatiza tareas de mantenimiento de cuentas de clientes.
- Empaqueta, comprime y minimiza la aplicación para su ejecución.
- Configura el subversion system.
- Realiza scripts de automatización con el servidor.

- Crea tareas para ejecutar backups automáticos.

3.9. 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.

Este apartado intenta cubrir los conceptos relacionados con la estimación del software a desarrollarse con este modelo considerando que se utilizan herramientas y técnicas concretas, con el objetivo de obtener una unidad de medida cuantitativa del software, cuyo valor resultante ayude a estimar el tiempo y el costo que demandará la construcción o calcular la magnitud del sistema en una unidad de medida estándar una vez que el proyecto esté todo concluido.

3.9.1. 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.

Como bien es sabido para obtener una estimación global en cuanto al costo existe una serie de otros factores que deben ser tomados en cuenta como son por ejemplo los costos administrativos y de gestión, gastos de oficina, alquileres, desplazamiento, energía eléctrica y honorarios profesionales. Así como también para estimar el tiempo es importante considerar además del tiempo de implementación, los feriados y vacaciones, eventos que se dan dentro de la empresa como cursos y jornadas, y los tiempos que maneja el cliente para atender las dudas y evacuar la consultas del equipo de desarrollo.

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.

3.9.2. 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.

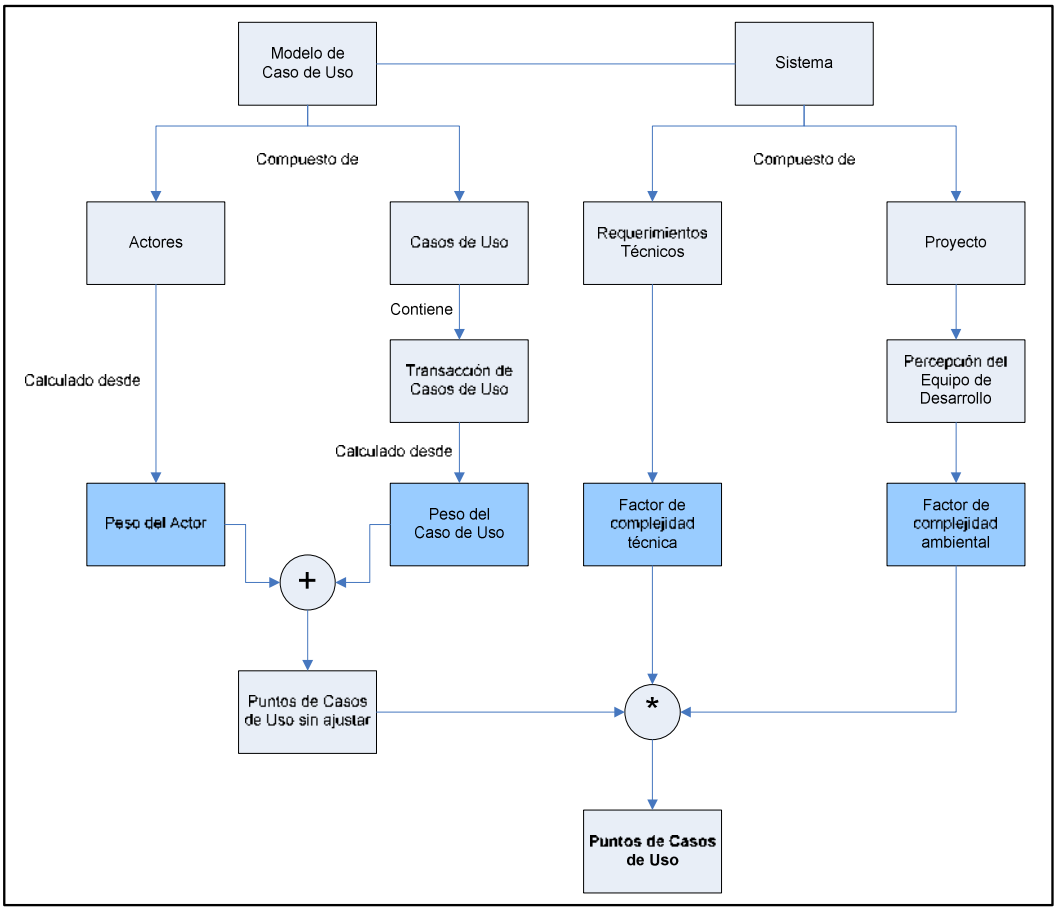

Figura 3.9-a – Puntos de casos de uso (Fuente propia en base a [38]).

La *figura 3.9-a* 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.

3.9.3. 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].

3.9.4. 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:

UCP = UUCP * TCF * ECF

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 3.9-a* y la suma de los pesos de la complejidad de los actores en la *tabla 3.9-b*:

1. Suma de los pesos de la complejidad de los casos de uso sin ajustar				
Tipo de Caso de Uso	**Descripción**	**Peso**	**Cantidad de Casos de Uso**	**Subtotal**
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**

Tabla 3.9-a – Factor de complejidad de los casos de uso [38].

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.

2. Suma de los pesos de la complejidad de los actores				
Tipo de Actor	**Descripción**	**Peso**	**Número de actores**	**Subtotal**
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**

Tabla 3.9-b – Factor de complejidad de los actores [38].

Como resultado para el cálculo para el UUCP tenemos entonces:

UUCP = 125 *(Tabla 3.9-a)* + 9 *(Tabla 3.9-b)*.

UUCP = 134.

El proceso continúa de la misma manera para calcular el factor de complejidad técnica (TCF), el cual en el modelo original está compuesto por 13 puntos clave de los cuales finalmente para este modelo se dejan sólo 5 ya que los demás se consideran irrelevantes si se sigue esta metodología.

Para calcular el TCF se utiliza un peso fijo establecido por la metodología y un factor que representa la complejidad percibida de manera subjetiva por el equipo de desarrollo cuyos valores se encuentran en el rango de 1 al 5.

A continuación en la *tabla 3.9-c* se puede observar un caso práctico:

Factor de complejidad técnica				
Factor técnico	**Descripción**	**Peso Fijo**	**Impacto Percibido (0 al 5)**	**Subtotal**
T1	Rendimiento o tiempo de respuesta	1	2	2
T2	Procesamiento interno complejo	1	1	1
T3	Facilidad de uso	0.5	2	1
T4	Características especiales de seguridad	1	0	1
T5	Se requiere facilidades especiales de entrenamiento al usuario	1	1	1
			Total	**6**

Tabla 3.9-c – Factor de complejidad técnica (Fuente propia en base a [38]).

Finalmente para concluir con el cálculo existe una constante aplicado a una formula se obtiene el resultado:

TCF = 0.6 + (0.01 * Subtotal Factor Técnico).

Así, reemplazando se tiene:

TCF = 0.6 + (0.01 * 6).

TCF = 0.66.

El siguiente paso consiste en calcular el factor de complejidad ambiental (ECF), donde se tiene en cuenta también el impacto percibido por el equipo de desarrollo, según se observa en la *tabla 3.9-d*.

Factor de complejidad ambiental				
Factor ambiental	**Descripción**	**Peso Fijo**	**Impacto Percibido (0 al 5)**	**Subtotal**
E1	Personal tiempo parcial	-1	2	-2
E2	Capacidad del analista líder	0.5	0	0
E3	Motivación	1	4	4
			Total	**2**

Tabla 3.9-d – Factor de complejidad ambiental (Fuente propia en base a [38]).

Finalmente para concluir con el cálculo existe una constante aplicado a una formula se obtiene el resultado

ECF = 1.4 + (-0.03 * Subtotal Factor Ambiental).

Así, reemplazando se tiene:

ECF = 1.4 + (-0.03 * 2).

ECF = 1.34.

Finalmente reemplazamos el factor de cada formula obtenida en la ecuación original, que era:

UCP = UUCP * TCF * ECF.

Reemplazamos, se tiene:

UCP = 134 * 0.66 * 1.34.

UCP = 118.51 (PUNTOS DE CASOS DE USO).

3.9.5. 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.

Haciendo un cálculo estimado por el valor referencial de 20, se tendría que entones multiplicar los puntos de casos de uso por 20.

Horas estimadas = 118.51 (UCP) * 20.

Resultado = 2371 Hs.

Posteriormente se podría utilizar el resultado de las horas obtenidas, para calcular la cantidad de recursos disponibles y posteriormente el tiempo total que llevará el desarrollo.

3.10. Discusiones y comentarios

Este capítulo presentó una serie de informaciones relacionadas a la práctica del desarrollo de software que en su conjunto forman parte de la metodología propuesta.

Para lograr el objetivo real de lograr agilidad en la construcción de sistemas, se vio necesario trabajar en base a tres elementos fundamentales que según

varias fuentes bibliográficas, son los pilares para el desarrollo rápido y por qué no decirlo también, los pilares de la metodología y que con el correr de los tiempos se convertirán en las herramientas indispensables en las empresa de desarrollo.

Si bien los tres pilares están para ser aprovechados en la etapa de la construcción, sólo los dos primeros pilares (la generación automática de código inicial y la utilización de plantillas pre-definidas) se proponen como directamente destinados a la construcción de funcionalidades, siendo el tercer pilar (ejecución de scripts automáticos) destinado a las pruebas y al despliegue del sistema, éste último siendo de vital importancia para el desarrollo en la nube.

Es interesante rescatar que las demás metodologías ágiles no hacen referencia al uso de estas herramientas como parte de sus procesos, debe ser por que abordan temas más orientados a procedimientos y cuestiones de organización, en fin, como aquí se abordó el tema de una metodología ágil basado en una plataforma específica (Java EE) se abrió paso a esta recomendación.

En el siguiente capítulo se hará una evaluación basado en dos criterios diferentes, la que valida si la propuesta cuenta con los requisitos mínimos que deben de contemplar una metodología de desarrollo y la que hace una comparación de la metodología propuesta con sus demás pares ágiles SCRUM y XP.

Capítulo IV

Evaluación del modelo

En este capítulo se realiza la evaluación del modelo propuesto en base a una serie de indicadores.

4.1. Evaluación del modelo propuesto

Con el objetivo de cumplir cabalmente con la tarea de auto-evaluar la propuesta surgió la necesidad de realizar el estudio de varias metodologías de evaluación existentes en la bibliografía estudiada, que conjugadas ayudan a cuantificar y obtener un valor cualitativo del análisis.

En un primer momento se realiza la evaluación de la metodología teniendo en cuenta los requerimientos mínimos que son necesarios para la elaboración de una metodología de desarrollo de software [40], indicando los puntos que ésta debe contemplar a modo general y que fueron extraídos de un estudio realizado por el profesor Rafael Barzanallana de la Universidad de Murcia [41].

Posteriormente se especifican los requerimientos en una matriz de información que se cruzan con el cumplimiento o no de cada ítem de la propuesta, obteniendo así un valor más cualitativo que cuantitativo, pero que sirve para indicar en qué nivel se cumplen las necesidades metodológicas,

técnicas y de resultados, obteniendo de cierto modo un valor para la calidad del modelo.

En un análisis posterior, se realiza otra evaluación de la propuesta para determinar en qué nivel se cumplen los requerimientos de las metodologías ágiles haciendo hincapié en los postulados de su manifiesto [23], obteniendo así un valor más cuantitativo que cualitativo y también comparando con los valores obtenidos a través del mismo análisis con sus metodologías pares SCRUM y XP.

4.2. Tipos de evaluación

Para la realización de la evaluación del modelo propuesto fue realizado una adaptación de dos formatos bibliográficos presentados en [23, 40, 41] donde se definen algunos indicadores especiales que se tienen en cuenta en la hora de la evaluación y se utilizan para comparar las dos metodologías antes comentada, SCRUM y XP, para presentarlas en forma de comparación.

Es importante tener en cuenta que para el caso específico del desarrollo de este modelo se tuvo en cuenta una serie de otros requerimientos necesarios para su implantación en la nube, como son por ejemplo el despliegue, motivo por el cual surgen nuevos indicadores que son añadidos a la evaluación realizada.

Existen sólo algunos trabajos realizados sobre evaluación de metodologías de desarrollo, basadas en indicadores mayormente cuantitativos. En cuanto a la bibliografía encontrada, resultó muy interesante el framework propuesto en [23] el cual se dedica netamente a evaluar las metodologías ágiles más tradicionales, SCRUM y XP utilizando netamente para ello indicadores extraídos del manifiesto ágil.

Complementando el framework de desarrollo ágil descrito en el párrafo anterior, en [40] también se hace referencia a un modelo de evaluación de

metodologías de desarrollo bastante interesante, ya que además de las metodologías ágiles, también hace un fuerte análisis comparativo sobre las metodologías tradicionales. Como bien es sabido, la tecnología Java EE está más enfocada para su desarrollo en tecnologías tradicionales, donde se exige un fuerte énfasis en la documentación de análisis, para luego realizar la implementación.

Cabe recalcar entonces de que para la evaluación se utilizan dos metodologías que se combinan para la obtención de un resultado más significativo.

4.3. Calidad del modelo

La primera evaluación será realizada en base a si el cumplimiento del modelo propuesto cumple mínimamente con los requerimientos básicos que debe contener una propuesta metodológica.

En tal sentido, en [40] se exponen 13 requisitos mínimos e indispensables que toda metodología debe poseer, los cuales se transcriben en la *tabla 4.3-a*:

Nro	Requerimiento	Descripción
1	Debe ajustarse a los objetivos.	Cada metodología tiene un fin que lo motiva a su creación y cuya metodología debe cubrir una vez concluida y presentada.
2	Debe cubrir todas las fases del desarrollo del software.	Se deben de proponer acciones, actividades y procedimientos para todas las etapas, desde el inicio hasta su entrega y puesta en funcionamiento del Software.
3	Debe integrar las distintas fases del desarrollo.	Debe tenerse en cuenta asuntos como la rastreabilidad, posibilidad de moverse también hacia una fase previa, y las limitaciones y responsabilidades que cada fase

		debe completar antes de iniciar una nueva.
4	Debe incluir la realización de validaciones.	Detectar un error de manera temprana e incluso antes de que ocurran es crucial para todo proceso de desarrollo, por eso la metodología debe proveer las herramientas para su identificación
5	Brindar el soporte para determinar el estado actual del sistema en cualquier momento.	Esto implica muchos asuntos que van desde las especificación de las necesidades hasta su comprobación de su cumplimiento.
6	Servir como base para una comunicación efectiva.	Además de gestionar el trabajo del equipo del proyecto, se deben de incluir procedimientos de comunicación efectiva entre todos los involucrados (stakeholders) del proyecto.
7	Adaptarse a un entorno dinámico orientado al usuario.	Durante todo el ciclo de vida, debe permitir la transferencia de conocimientos hacia el usuario. Involucrar al usuario es de importancia vital ya que sus necesidades evolucionan constantemente.
8	Especificar claramente los responsables de los resultados.	Debe especificar claramente quienes son los participantes de cada tarea a desarrollar.
9	Poder utilizarse en un amplio entorno de proyectos de software.	En cuanto a la variedad de sistemas y clientes, así como también considerar el tamaño y la vida del sistema así como también su complejidad y entorno tecnológico.
10	Se debe poder enseñar.	Por más práctica que resulte, si su complejidad para aprender es muy dificultosa y demore mucho tiempo

		no sería de mucha utilidad.
11	Debe poder ser soportada por herramientas existentes o a desarrollar.	Las herramientas automatizadas mejoran la productividad del equipo, así como del desarrollo en general.
12	Soportar la evolución del sistema.	Para ello debe permitir estructurar (versionar) sus elementos de manera que resulte sencillo su mantenimiento a lo largo mucho tiempo.
13	Requerir de actividades que conduzcan a la mejora continua del desarrollo.	

Tabla 4.3-a – Requisitos indispensables de una metodología [40].

Teniendo en cuenta los puntos especificados en la *tabla 4.3-a*, lo primero a realizar es la asignación de valores a para cada uno de los requerimientos, con el objetivo de visualizar el nivel de cumplimiento de los mismos, en el contexto del modelo propuesto, de esa manera se tiene la siguiente escala de valores:

1 = Totalmente en desacuerdo.

2 = Desacuerdo parcialmente.

3 = Indeciso neutral.

4 = De acuerdo parcialmente.

5 = Totalmente de acuerdo.

Utilizando esta escala de valores, posteriormente se procederá a establecer equivalencias de porcentajes para una mejor interpretación de resultados, quedando de la siguiente manera 1 = 10%, 2 = 40%, 3 = 60%, 4 = 80% y 5 = 100% [40].

Posteriormente se elabora una *tabla 4.3-b* donde pueden ser visualizados los valores completados.

Nro	Requerimiento	Punto	%
1	Debe ajustarse a los objetivos.	5	100%
2	Debe cubrir todas las fases del desarrollo del software.	5	100%
3	Debe integrar las distintas fases del desarrollo.	3	60%
4	Debe incluir la realización de validaciones.	2	40%
5	Brindar el soporte para determinar el estado actual del sistema en cualquier momento.	3	60%
6	Servir como base para una comunicación efectiva.	4	80%
7	Adaptarse a un entorno dinámico orientado al usuario.	4	80%
8	Especificar claramente los responsables de los resultados.	3	60%
9	Poder utilizarse en un amplio entorno de proyectos de software.	4	80%
10	Se debe poder enseñar.	3	60%
11	Debe poder ser soportada por herramientas existentes o a desarrollar.	4	80%
12	Soportar la evolución del Sistema.	4	80%
13	Requerir de actividades que conduzcan a la mejora continua del desarrollo.	4	80%

Tabla 4.3-b – Resultados de la evaluación de la metodología (Fuente propia).

A través de la tabla anterior, se puede visualizar en qué porcentaje se cumplen cada uno de los requerimientos metodológicos de la propuesta presentada.

4.4. Evaluación cuantitativa

A continuación se describe el proceso de evaluación de las metodologías ágiles [23] el cual mide de qué manera las metodologías ágiles cumplen con los postulados del Manifiesto Ágil.

A modo de recordatorio, se vuelven a exponer aquí los postulados:

1. Valorar al individuo y a las interacciones del equipo de desarrollo por encima del proceso y las herramientas.
2. Valorar el desarrollo de software que funcione por sobre una documentación exhaustiva.
3. Valorar la colaboración con el cliente por sobre la negociación contractual.
4. Valorar la respuesta al cambio por sobre el seguimiento de un plan.

A modo de tener un valor cuantitativo como resultado de la evaluación, el framework define medidas donde son utilizadas una escala de intervalos.

Se proveen de esta manera las mediciones para los cuatro postulados del manifiesto ágil basados en la siguiente expresión (Pi, i=[1..4]).

En consecuencia de que cada postulado del Manifiesto ágil, posee una sugerencia para valorar más a algunos elementos que por sobre otros, es que a la expresión anterior (Pi, i=[1..4]) serán expresados como la valoración de dos atributos (Pi.A y Pi.B).

Entonces, para el postulado 1 (P1) se tendrían las siguientes dos medidas:

P1 A

Valorar al individuo y a las interacciones del equipo de desarrollo.

P1 B

Por encima del proceso y las herramientas.

La medida de cada postulado se define como la suma de las medidas de los atributos relacionados, como se formula a continuación:

$$m(Pi) = m(Pi.A) + m(Pi.B), i=[1..4].$$

Para la asignación de los respectivos valores a las medidas, se utiliza una escala de valores máximos que van desde [-5..5] distribuidos de la siguiente manera:

Para los valores que el postulado intenta reforzar, ej: "Valorar al individuo y a las interacciones del equipo de desarrollo", se asignarán valores positivos que van en una escala del [0..5]. Para los valores que el postulado intenta minimizar, ej: "Por encima del proceso y las herramientas", se asignarán valores negativos que van en una escala del [-5..0].

Así, cada principio podría obtener una medida entre -5, en el peor de los casos, ej: (-5 el atributo negativo y 0 el atributo positivo), y 5, en el mejor de los casos (0 el atributo negativo y 5 el atributo positivo).

Para el principio que se obtenga un valor cercano a cero, esto indicaría que la metodología analizada no se enfoca a satisfacer significativamente los principios positivos por sobre el negativo, con el cual se puede concluir que el Manifiesto Ágil no se satisface plenamente.

4.4.1. Presentación del framework de evaluación

A continuación se transcribe la planilla utilizada en el framework [23] que será utilizado como base para la evaluación.

El framework presenta 4 (cuatro) tablas en total, donde dichos indicadores son desglosados y valorados según los tópicos comentados anteriormente.

La *tabla 4.4-a* presenta entonces, la valoración del primer postulado del manifiesto ágil:

P1	**Valorar al individuo y a las interacciones del equipo de desarrollo por encima del proceso y las herramientas.**		
P1 A	Valorar al individuo y a las interacciones	P1 B	Valorar procesos y herramientas
Valor	Descripción	Valor	Descripción
0	No define roles para individuos.	-5	Define actividades, entregables, herramientas de desarrollo y de gestión.
1	Clara definición de roles para individuos.	-3	Define actividades, entregables, y herramientas de desarrollo.
2	Clara definición de roles y responsabilidades.	-2	Define actividades y entregables.
3	Clara definición de roles, responsabilidades y conocimientos técnicos.	-1	Define actividades para cada iteración.
5	Clara definición de roles, responsabilidades, conocimientos técnicos e interacciones entre el equipo de trabajo.	0	Define actividades para el proyecto pero no a nivel de cada iteración.

Tabla 4.4-a – Valoración del primer postulado del manifiesto ágil [23].

Como la tabla 4.4-a es la primera de las que serán utilizadas, cabe destacar que por el lado izquierdo se especifican los indicadores que llevan valores positivos y por el lado izquierdo los indicadores negativos.

En la *tabla 4.4-b* se presenta la valoración del segundo postulado:

P2	**Valorar el desarrollo de software que funcione por sobre una documentación exhaustiva**		
P2 A	Valorar el desarrollo de software que funcione	P2 B	Valorar documentación exhaustiva
Valor	Descripción	Valor	Descripción
0	Generar entregable al finalizar el proyecto.	-5	Requiere documentación detallada al inicio del proyecto.
3	Generar entregable con testing satisfactorio al finalizar cada iteración.	-3	Requiere sólo documentación necesaria al inicio de cada iteración.
5	Generar entregable con testing satisfactorio integrado con el resto de las funcionalidades al finalizar cada iteración.	0	No requiere documentación para comenzar a implementar la funcionalidad incluida en una iteración.

Tabla 4.4-b – Valoración del segundo postulado del manifiesto ágil [23].

En la *tabla 4.4-b* se puede observar que los valores no son valores consecutivos, lo cual indica que al utilizar el framework de evaluación dichos indicadores pueden ser evaluados en valores intermedios que no se indiquen precisamente en la tabla pero que corresponden igualmente ya que según el evaluador son los valores que más se acercan al resultado.

En la *tabla 4.4-c* se valora el tercer postulado del manifiesto ágil:

P3	**Valorar la colaboración con el cliente por sobre la negociación contractual**		
P3 A	Valorar la colaboración con el Cliente	P3 B	Valorar la negociación contractual
Valor	Descripción	Valor	Descripción
0	Cliente colabora a demanda del Equipo.	-5	Existe contratación detallada al inicio y no se aceptan cambios.
3	Cliente es parte del equipo, responde las consultas y planifica las iteraciones.	-3	La contratación exige contemplar cambios durante el proyecto.
5	Cliente es parte del equipo, responde las consultas, planifica las iteraciones y colabora con la escritura de requerimientos y pruebas.	0	El contrato por la construcción del producto no aporta valor al producto.

Tabla 4.4-c – Valoración del tercer postulado del manifiesto ágil [23].

Los valores otorgados como peso a cada postulado de la *tabla 4.4-c* son similares a la de la tabla anterior *(tabla 4.4-b)* motivo por el cual la evaluación está sujeta a la misma condición.

Y en la *tabla 4.4-d* se desglosa el cuarto postulado del manifiesto:

P4	**Valorar la respuesta al cambio por sobre el seguimiento de un plan**		
P4 A	Valorar la respuesta al cambio	P4 B	Valorar el seguimiento de un plan
Valor	Descripción	Valor	Descripción
0	No prevé incorporar cambios durante la ejecución del proyecto.	-5	Define un plan detallado al inicio del proyecto.
1	Prevé introducir cambios sólo de alta prioridad.	-3	Define un plan detallado de iteraciones, no aceptando cambios durante la iteración.
4	Permite la evolución y el cambio, pero no es recomendable en la iteración en curso.	-1	Define un plan detallado para cada iteración, que puede ser modificado.
5	Permite introducir cambios en la iteración en curso.	0	No define planificación alguna.

Tabla 4.4-d – Valoración del cuarto postulado del manifiesto ágil [23].

4.4.2. Aplicación del framework de evaluación

A continuación, en la siguiente *tabla 4.4-e*, se puede observar los resultados obtenidos de la aplicación de la metodología, en base al modelo planteado y a las dos metodologías ágiles en estudio de este trabajo.

Resultados	**P1**			**P2**			**P3**			**P4**		
Metodología	P1A	P1B	**TOT**	P2A	P2B	**TOT**	P3A	P3B	**TOT**	P4A	P4B	**TOT**
SCRUM	5	-3	**2**	4	-2	**2**	5	0	**5**	4	-3	**1**
XP	5	-2	**3**	5	-2	**3**	5	0	**5**	5	-2	**3**
CloudAgileJ	3	0	**3**	3	-3	**0**	2	0	**2**	4	-1	**3**

Tabla 4.4-e – Resultados de la evaluación cuantitativa (Fuente propia).

Atendiendo a los resultados obtenidos, se puede realizar el siguiente análisis:

Para el *postulado 1*, tanto el XP como la metodología propuesta, llevan la delantera debido a que ambos obtuvieron un valor de 3, mientras que SCRUM, obtuvo solamente 2.

El promedio de la puntuación obtenida en el *postulado 1*, el cual se considera medio para bueno, se debe principalmente a que el modelo define las actividades que deben ser ejecutadas (P1 B), pero no a un nivel muy profundo por lo que se asigna el valor 0 (cero) para éste indicador, dejando siempre abierta la imaginación, lo cual lo hace adaptativo dependiendo de cada caso en particular [32].

Para el caso del *postulado 2*, nuevamente XP se pone a la delantera frente a SCRUM y el modelo CloudAgileJ aparentemente tendría un valor neutro, ya que el resultado final es cero, pero esto se analiza en el siguiente párrafo.

El resultado obtenido por CloudAgileJ para el *postulado 2* aunque parezca neutro no lo es tanto así, ya que el mismo se debe a que es muy importante ir realizando entregas de formas parciales (P2 A = 3), es decir, luego de cada iteración, para que el usuario ponga en marcha su funcionamiento, al mismo tiempo de que aunque no se requiera de una documentación exhaustiva, sí es necesario algo básico al inicio de cada iteración (P2 b = -3). Se debe recordar que el modelo promueve experimentar en iteraciones pequeñas [15].

En el caso del *postulado 3*, al observar los resultados de la *tabla 4.4-e*, el modelo CloudAgileJ aparentemente obtuvo un valor inferior a la media en relación a los demás, pero analizando el contexto propuesto el resultado es satisfactorio y justificable, debido a que el resultado de 2 se obtiene por razón de que para el modelo propuesto no es una exigencia que el cliente forme parte del equipo de desarrollo, tal como lo promueven las 2 metodologías ágiles en cuestión, lo cual no significa que no exista

colaboración sino más bien que el cliente colabora desde donde está probablemente desde la oficina donde habitualmente trabaja.

Y por último y para finalizar, se puede notar que para el caso del *postulado 4*, sí la valoración es significativa, con un valor de 3, ya que se encuentra en la misma escala que la metodología SCRUM, siendo que XP ha obtenido solamente un valor de 1.

El resultado habla por sí mismo. Al observar el valor de (P4 A = 4), se identifica claramente que el modelo propuesto permite la modificación o el cambio de requisitos, lo cual es la esencia de las metodologías ágiles, sino nada tendría mucho sentido; a su vez como ya se mencionó anteriormente se requiere de una documentación inicial mínima, pero modificable en el tiempo (P4 B = -1) motivo por el cual el resultado es de 3.

4.5. Discusiones y comentarios

En este capítulo se presentó la evaluación realizada sobre el modelo propuesto con el objetivo de brindarle un mayor sustento académico-investigativo.

Cabe destacar también que el proceso de evaluación sirvió para que se tomen en cuenta una serie de otros parámetros no tenidos en cuenta inicialmente, pero que fueron revisados y agregados para mejorar los indicadores y volviéndose a evaluar posteriormente en varias oportunidades hasta llegar al resultado obtenido.

Como análisis general se puede mencionar que la metodología propuesta presenta los resultados esperados de una buena evaluación conforme lo planteado, teniendo en cuenta los parámetros de la región y de la tecnología Java EE que se desea aplicar, arrojando como resultado de la evaluación algunos valores inferiores numéricamente, pero debidamente justificados,

por lo que se presta a interpretación, teniendo en cuenta que no todo valor inferior sea malo, sino que probablemente exista alguna razón.

En el siguiente capítulo, se ejemplificará el uso y la aplicación del modelo en base al desarrollo de un sistema real para una empresa de desarrollo ya constituida.

Capítulo V

Aplicación práctica del modelo

5.1. Resumen

Este capítulo está destinado a servir de referencia rápida al lector, de forma a comprender la utilización de la metodología a través de un ejemplo práctico.

Se presentarán una serie de escenarios tratando de simular las actividades que ocurren en una empresa de desarrollo a la hora de construir un producto donde se visualizará cómo se van utilizando los conceptos de este modelo en las diferentes etapas.

Durante la aplicación práctica se brinda la mayor cantidad información gráfica posible completando una iteración entera, de forma a guiar al lector a una fácil comprensión del modelo.

Se inicia la utilización del modelo empezando la primera iteración.

5.2. Pre requisitos para la compresión del ejemplo práctico

Se debe tener en cuenta que este modelo presenta una estructura de desarrollo iterativo, en donde para una iteración inicial del primer sistema que será creado, será necesario que ciertas condiciones estén presentes.

Dichas condiciones se exponen a continuación y se presentan como información previa necesaria para la comprensión de la aplicación del modelo.

5.2.1. Nombre de la empresa

SaaS S.A.

5.2.2. Tres pilares para el desarrollo rápido

La Empresa "SaaS S.A." se encuentra en el mercado hace poco más de 6 meses, motivo por el cual ya tiene una infra-estructura básica de desarrollo fundamentada en los 3 pilares, en cuanto a:

- Generación automática de código inicial.
- Utilización de plantillas pre definidas.
- Scripts automáticos.

Es decir, al iniciarse la empresa, los primeros productos en los que estuvieron trabajando fueron en las herramientas propias que servirían como base para los siguientes desarrollos que vendrían.

5.2.3. Roles y recursos

La empresa ya cuenta con 4 recursos destinados, y los mismos se encuentran organizados de la siguiente manera (ver *tabla 5.2-a*):

Nombre de Profesional	Rol
Juan Martínez	Director Gerente Líder de Proyecto
Ana González	Analista de Sistemas
José Pérez	Arquitecto Desarrollador
Pedro Benítez	Gerente de Configuración

Tabla 5.2-a – Roles y recursos disponibles para la aplicación (Fuente propia).

5.3. Definición de la idea

Se inicia la actividad con la primera etapa, la etapa que corresponde al nodo de la *idea*, pasando a definir todos los datos relacionados al contexto. En la *figura 5.3-a* se puede observar que todo inicia con el nodo de la *idea*, denotando un fondo un poco más claro y con letras de color amarillo.

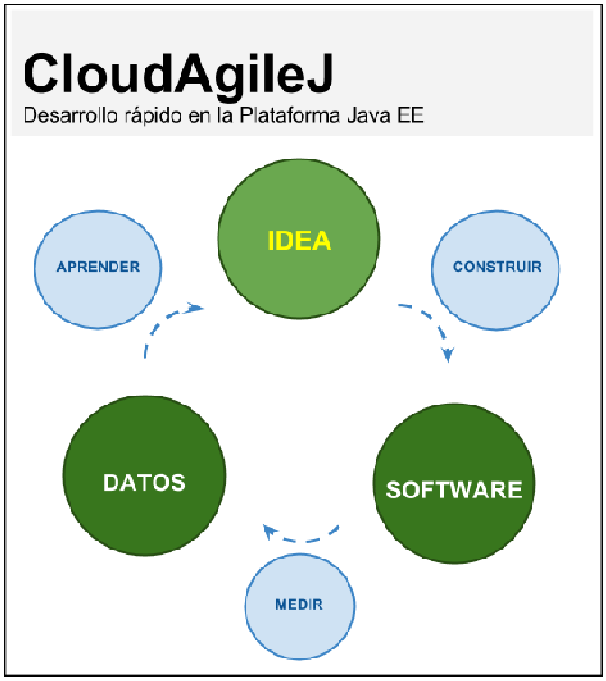

Figura 5.3-a – Nodo *idea* de la primera iteración (Fuente propia).

Como la empresa encuentra situada en este nodo, estará realizando todo lo necesario para definir claramente su idea para el software, definir el producto y el mercado, a qué tipo de clientes irá dirigido, etc.

5.3.1. Escenario de la aplicación

La empresa "SaaS S.A." decide emprender un nuevo proyecto en la nube: Crear una Aplicación SaaS para los Estudios Contables de Ciudad del Este que funcione de manera centralizada y permita a los estudios realizar las operaciones a través de un usuario y contraseña.

De la descripción del escenario anterior, se desprenden los siguientes datos técnicos.

5.3.2. Aplicación

Sistema de Contabilidad - ContaSoft

Con dicha información en mano, se procede a iniciar la elaboración del Proyecto, utilizando las técnicas mencionadas en el modelo CloudAgileJ siguiendo un procedimiento paso a paso.

5.3.3. Paso 1 - Definición del producto y mercado

¿Cuánto tiempo?: 1 día.

¿Dónde?: Oficina de la empresa de software.

¿Quiénes?: Director/Gerente, Líder de Proyecto.

La primera actividad se realiza dentro de la Empresa de desarrollo, involucrando a todos los participantes del proyecto, incluyendo propietario, analista, arquitecto y desarrolladores.

El primer paso consiste en definir de manera contextual al producto dentro del mercado, utilizando la planilla proveída en anexo completando detalladamente cada uno de los bloques. A continuación en la *figura 5.3-b* se visualiza el resultado del modelo definido para la versión 1.1.

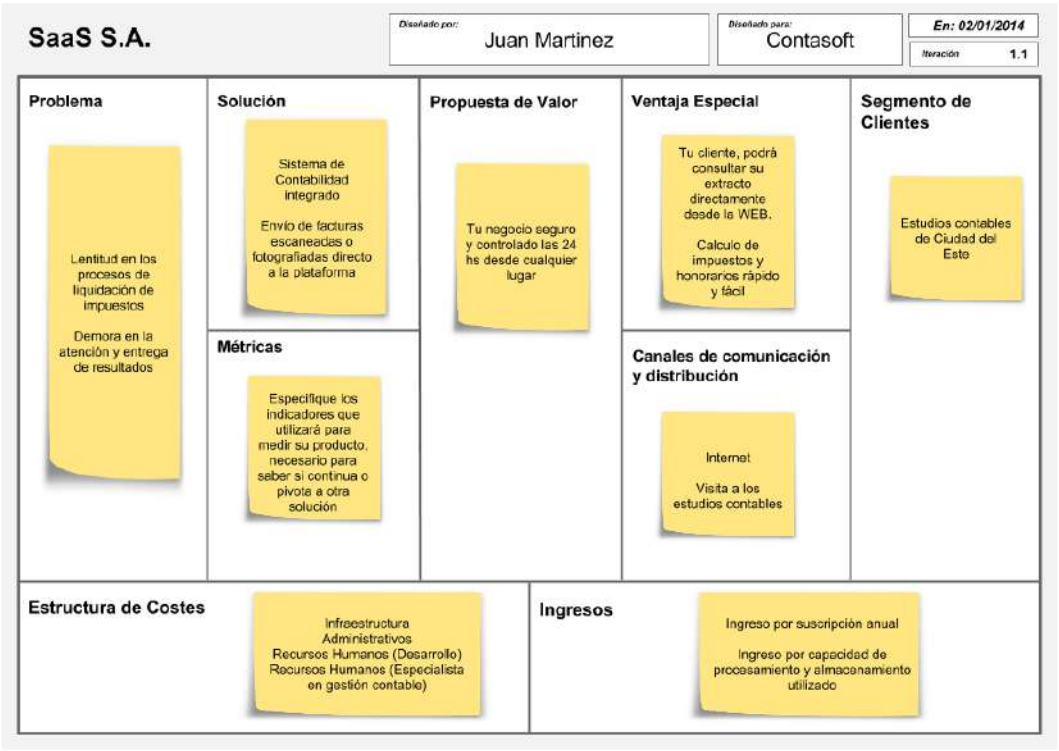

Figura 5.3-b – Definición del contexto del producto (Fuente propia).

Este lienzo dará una visión general a todos los recursos relacionados al desarrollo del nuevo Sistema, los harán sentirse comprometidos, con el plazo y con las funcionalidades que sean requeridas.

5.3.4. Paso 2 – Realizar reunión inicial

¿Cuánto tiempo?: Medio día.

¿Dónde?: Oficinas del cliente.

¿Quiénes?: Líder de Proyecto, Analista de Sistemas.

¿Para qué?: Para obtener datos para validar los datos de contexto realizado inicialmente, obtener funcionalidades recibidas por el cliente, detectar actividades prioritarias para los estudios contables.

Una vez contextualizado la aplicación, y teniendo bien claro desde el punto de vista de la empresa la solución que se desea crear, se procede a realizar reuniones con los clientes. En este caso se visita a 3 clientes en especial que fueron seleccionados previamente como clientes tempranos, que formarán parte de las empresas que probarán el sistema, y darán su feedback sobre las funcionalidades implementadas.

5.3.5. Paso 3 – Revisión del modelo contextual

¿Cuánto tiempo?: Medio día.

¿Dónde?: Empresa de software.

¿Quiénes?: Director/Gerente, Líder del Proyecto, Analista.

De la reunión inicial realizada con los clientes, se detecta que el modelo de contexto debe ser actualizado a la versión 1.2, el cual se presenta a continuación en la *figura 5.3-a*:

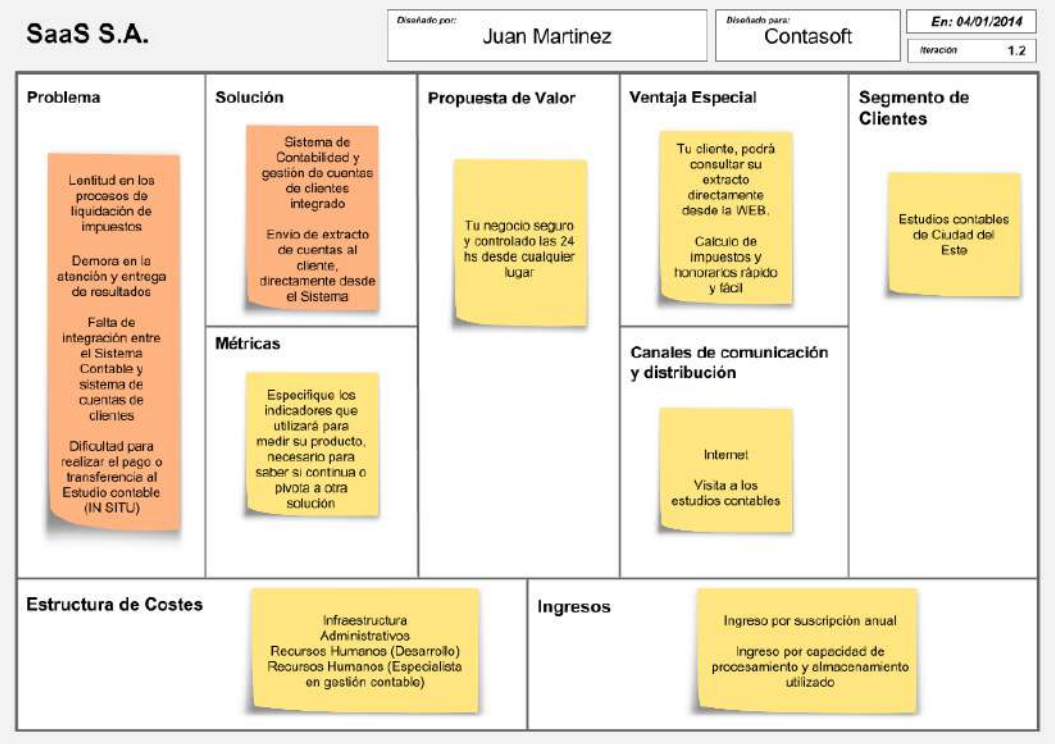

Figura 5.3-c – Definición del contexto del producto modificado (Fuente propia).

En dicha reunión resultó interesante la discusión con el cliente, que mencionó que a parte de los problemas mencionados por la empresa de desarrollo, uno de los problemas que genera más lentitud en todo el proceso es el que no se cuenta con un sistema de cuentas de clientes, integrado al sistema de contabilidad, de forma a poder calcular rápidamente el extracto al cliente.

Otro de los problemas es que los sistemas de pagos utilizados actualmente son todos off-line, pudiendo ya contar un sistema sincronizado con los medios automáticos disponibles, Aquí pago, banco, etc.

5.3.6. Paso 4 – Estimar la magnitud del sistema

¿Cuánto tiempo?: Medio día.

¿Dónde?: Empresa de software.

¿Quiénes?: Director/Gerente, Líder del Proyecto, Analista.

Como bien se ha dicho en la propuesta, la estimación es uno de los requerimientos más solicitados por parte del cliente y necesario también para que la empresa de desarrollo pueda estimar un costo y tener una base del tiempo que llevará su desarrollo.

Para estimar la magnitud, primeramente se deben establecer entonces los actores y casos de uso, el cual para el sistema en cuestión se visualizan en la *tabla 5.3-a* y *tabla 5.3-b*:

Lista de Actores	**Iteración: 1**	**Fecha: 10/05/2014**	
Nro.	**Nombre del Actor**	**Descripción**	**Complejidad**
1	Sistema Externo.	Otro sistema mediante una API	1 – Simple
2	Gerente.	Usuario por medio de una interfaz	3 – Complejo
3	Auxiliar contable.	Usuario por medio de una interfaz	3 – Complejo

Tabla 5.3-a – Lista de actores para estimación de magnitud (Fuente propia).

Lista de Casos de Uso	**Iteración: 1**	**Fecha: 10/05/2014**	
Nro.	**Casos de Uso**	**Complejidad**	**Transacción**
1	Plan de Cuentas.	3 – Complejo	8
2	Asiento Diario.	2 – Medio	5
3	Centro de Costos.	1 – Simple	3
4	Libro de Compras.	2 – Medio	4
5	Libros de Ventas.	2 – Medio	4
6	Informe Mayor.	1 – Simple	1
7	Informe Balance.	2 – Medio	2
8	Registro de Clientes.	1 – Simple	3
9	Cuentas de Clientes.	2 – Medio	4
10	Cobros de Clientes.	2 – Medio	4

Tabla 5.3-b – Casos de uso para estimación de magnitud (Fuente propia).

Una vez establecido la complejidad de los actores y los casos de uso, se procede a completar las primeras planillas con las informaciones de los casos de uso, con el objetivo de poder calcular la ecuación:

UCP = UUCP * TCF * ECF

Se procede a calcular entonces el primer factor de la ecuación (UUCP), completando los valores en la *tabla 5.3-c*.

1. Suma de los pesos de la complejidad de los casos de uso sin ajustar				
Tipo de Caso de Uso	**Descripción**	**Peso**	**Cantidad de Casos de Uso**	**Subtotal**
Simple	De 1 a 3 transacciones. Hasta 5 clases.	5	3	15
Medio	4 a 7 transacciones. De 6 a 10 clases.	10	6	60
Complejo	Más de 7 transacciones. Más de 10 Clases.	15	1	15
			Total	**90**

Tabla 5.3-c – Factor de complejidad de los casos de uso (Fuente propia).

Ahora se procede a completar la complejidad de actores, el cual se visualiza en la *tabla 5.3-d*:

2. Suma de los pesos de la complejidad de los actores				
Tipo de Actor	**Descripción**	**Peso**	**Número de actores**	**Subtotal**
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	0	0
Complejo	Una persona que interactúa con el sistema mediante una interfaz gráfica (GUI).	3	2	6
			Total	**7**

Tabla 5.3-d – Factor de complejidad de los actores (Fuente propia)

Realizando la sumatoria de las dos tablas (*tabla 5.3-c* y *tabla 5.3-d*) se tiene como resultado:

UUCP = 90 + 7.

UUCP = 97.

En la *tabla 5.3-e* se completan los valores de la complejidad técnica:

Factor de complejidad técnica				
Factor técnico	**Descripción**	**Peso Fijo**	**Impacto Percibido (0 al 5)**	**Subtotal**
T1	Rendimiento o tiempo de respuesta	1	2	2
T2	Procesamiento interno complejo	1	1	1
T3	Facilidad de uso	0.5	2	1
T4	Características especiales de seguridad	1	0	1
T5	Se requiere facilidades especiales de entrenamiento al usuario	1	1	1
			Total	**6**

Tabla 5.3-e – Factor de complejidad técnica (Fuente propia).

TCF = 0.6 + (0.1 * 6).

TCF = 0.66.

El siguiente paso consiste en calcular el factor de complejidad ambiental (ECF), donde se tiene en cuenta también el impacto percibido por el equipo de desarrollo. El mismo se observa en la *tabla 5.3-f*.

Factor de complejidad ambiental				
Factor ambiental	**Descripción**	**Peso Fijo**	**Impacto Percibido (0 al 5)**	**Subtotal**
E1	Personal tiempo parcial	-1	2	-2
E2	Capacidad del analista líder	0.5	0	0
E3	Motivación	1	4	4
			Total	**2**

Tabla 5.3-f – Factor de complejidad ambiental (Fuente propia).

Finalmente para concluir con el cálculo existe una constante aplicado a una formula se obtiene el resultado:

ECF = 1.4 + (-0.03 * 2).

ECF = 1.34.

Finalmente se reemplaza el factor de cada formula obtenida en la ecuación original, que era:

UCP = 90 * 0.66 * 1.34.

UCP = 79.60 (PUNTOS DE CASOS DE USO).

A partir del resultado de la estimación, tanto el cliente como el equipo de desarrollo tendrán una idea del esfuerzo y tiempo que le llevará su desarrollo.

5.4. Desarrollo o implementación

Con la actualización del modelo y la estimación realizada se concluye con la primera fase constituida por el nodo de la *idea*, para pasar ahora al nodo de *construir*, donde inicialmente se estará trabajando específicamente sobre los requisitos del Sistema.

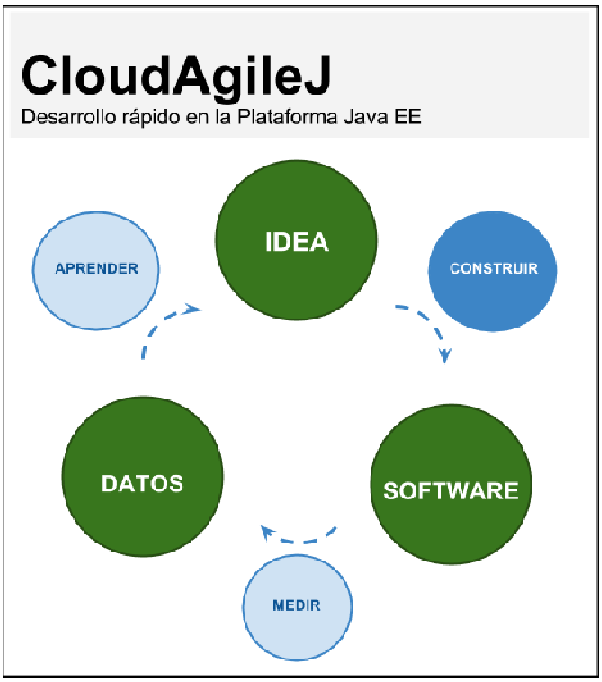

Figura 5.4-a – Nodo construir de la primera iteración (Fuente propia).

La ubicación del nodo construir dentro del esquema se puede identificar, gracias a la *figura 5.4-a.*

Se debe tener en cuenta que la Etapa de Construcción debe atravesar por 4 fases, los cuales se visualizan en la *figura 5.4-b*:

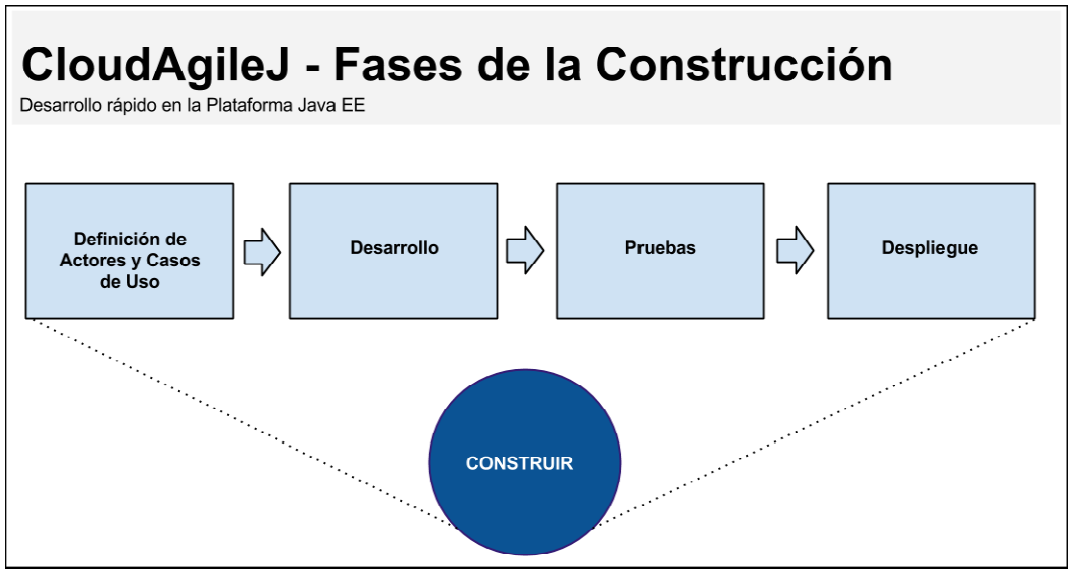

Figura 5.4-b – Fase de definición de funcionalidades, iteración 1 (Fuente propia).

Como ésta es la primera iteración, y no existe nada analizado aún, entonces se iniciará con la fase de definición de actores y casos de uso, el cual también exige un ordenamiento de los mismos de acuerdo a la lista de prioridades, el cual sería el siguiente paso.

5.4.1. Paso 5 – Lista de funcionalidades básicas

¿Cuánto tiempo?: Medio día.

¿Dónde?: En la empresa de software.

¿Quiénes?: Director de proyecto, Analistas.

La siguiente actividad consiste en definir la lista de actividades iniciales. Las informaciones sobre el relevamiento de datos sobre las necesidades que se deben contemplar, tuvieron que ser levantadas en la reunión superficial del Paso 2.

Se debe tener en cuenta que no necesitan ser funcionalidades terminales, es decir, basta con que esté compuesta de las principales características, con el fin de brindar a los involucrados los conceptos principales.

En la *tabla 5.4-a* se puede observar una lista inicial básica, compuesta de las funcionalidades levantadas en dicha reunión. Todos los involucrados deben estar abiertos a que tal vez no sean todas las funcionalidades que se requerirán finalmente y que habrá necesidades de realizar cambios futuros.

Lista de Casos de Uso	**Iteración: 1**	**Fecha: 10/05/2014**	
Nro.	**Casos de Uso**	**Complejidad**	**Prioridad**
1	Plan de Cuentas	3 – Complejo	3 – Alto
2	Asiento Diario	2 – Mediano	2 – Medio
3	Centro de Costos	1 – Simple	3 – Alto
4	Libro de Compras	2 – Medio	3 – Alto
5	Libros de Ventas	2 – Medio	2 – Medio
6	Informe Mayor	1 – Simple	2 – Medio
7	Informe Balance	2 – Mediano	2 – Medio
8	Registro de Clientes	1 – Simple	1 – Bajo
9	Cuentas de Clientes	2 – Mediano	2 – Medio
10	Cobros de Clientes	2 – Mediano	2 – Medio

Tabla 5.4-a – Listado de casos de uso para estimación (Fuente propia).

5.4.2. Paso 6 – Diseño de prototipos

¿Cuánto tiempo?: Medio día en promedio para aprox. 10 funcionalidades.

¿Dónde?: En la empresa de software.

¿Quiénes?: Analista de Sistemas, Arquitecto, Líder de Proyecto.

La siguiente actividad consiste en elaborar un bosquejo básico y superficial de la pantalla de la funcionalidad, si puede ser a mano y en lápiz mejor, considerando dejarlo bien definido para que las hojas puedan ser fotocopiadas posteriormente.

Como ya se tiene un esquema montado para la navegación del sistema y de los demás elementos (pantalla principal, menús, formato básico, etc.), se considera aquí solamente lo que hace relación específica a los campos de las funcionalidades y a las acciones (botones). Se evita desperdiciar el tiempo en cuestiones de formato y disposición, ya que eso fue definido en otro contexto de la empresa, no como parte del proyecto. El objetivo del prototipo es meramente el de servir de ayuda para el entendimiento de los demás miembros del equipo.

En la *figura 5.4-c* se puede visualizar un ejemplo para la funcionalidad de clientes, note que sólo contempla información específica de dicha funcionalidad:

Figura 5.4-c – Prototipo de pantalla de clientes (Fuente propia).

De esta misma forma, se elabora los prototipos para cada uno de las demás funcionalidades.

5.4.3. Paso 7 – Reunión de socialización - inicial

¿Cuánto tiempo?: Media mañana / media tarde.

¿Dónde?: En la empresa de software.

¿Quiénes?: Director de proyecto, Analista, Arquitecto, Desarrolladores.

Una vez concluida la tarea de prototipos de las funcionalidades iniciales, es hora de socializarlo con los demás involucrados, éste es el último paso antes de iniciar la siguiente fase de desarrollo y consiste abrir una discusión con el fin de aclarar todas las dudas posibles y enmarcar delineamientos básicos sobre el mismo, donde incluso los prototipos están abiertos a modificación.

Aquí también se explica el esquema de trabajo y la forma de asignación de actividades.

5.4.4. Paso 8 – Inicio del desarrollo

¿Cuánto tiempo?: Aprox. 4 días para 10 funcionalidades.

¿Dónde?: En la empresa de software.

¿Quiénes?: Desarrolladores y Arquitectos de Software.

Con este paso se inicia el modelado de la base de datos y el desarrollo en sí. Es aquí donde deberán hacerse uso de los 3 pilares. En un primer momento se hará uso del pilar de auto-generación de código, para generar el código inicial de todas las funcionalidades.

Esta fase se representa en la *figura 5.4-d* siguiente.

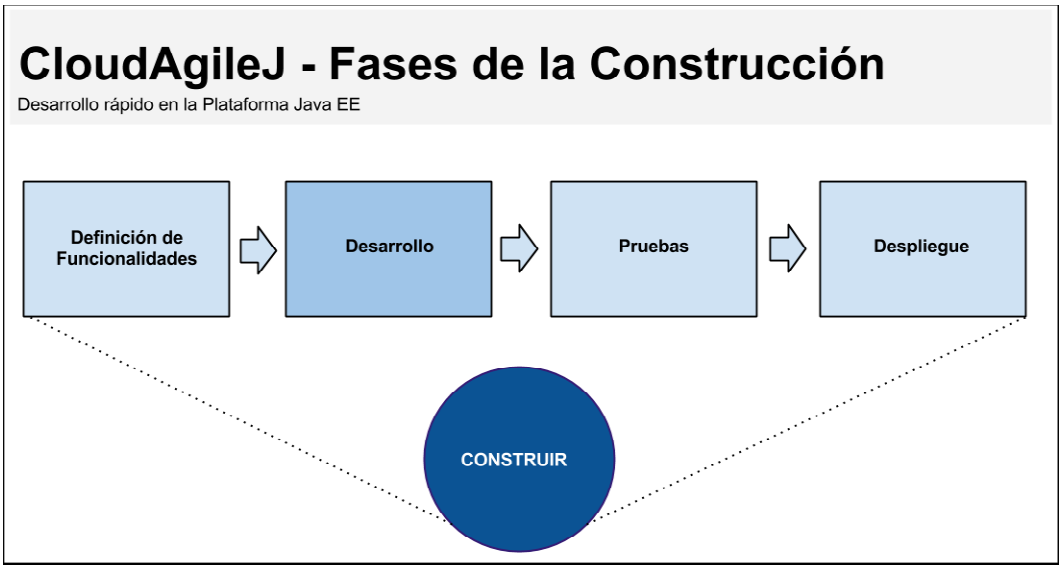

Figura 5.4-d – Fase de desarrollo, iteración 1 (Fuente propia)

Es decir, se parte de la *idea* de que la empresa ya se encuentra con los 3 pilares mínimamente levantados y cuenta con un auto-generador de código como el que se recomienda en el modelo.

Concluido esta etapa, la empresa de desarrollo ya cuenta con las funcionalidades básicas implementadas, ese sería su producto inicial, listo para disponibilizarlo si es que pasa las pruebas correspondientes.

Es necesario recordar que este producto inicial no cuenta con toda la lógica de negocio que se espera de una aplicación compleja.

5.4.5. Paso 9 – Pruebas

¿Cuánto tiempo?: Aprox. 2 días para 10 funcionalidades.

¿Dónde?: En la empresa de software.

¿Quiénes?: Testers y Desarrolladores.

Es hora de realizar las pruebas de funcionamiento del sistema auto-generado, esto se realiza probando el sistema primero con herramientas automatizadas, es decir con la ejecución automática de scripts, el cual es otro de los pilares de este modelo.

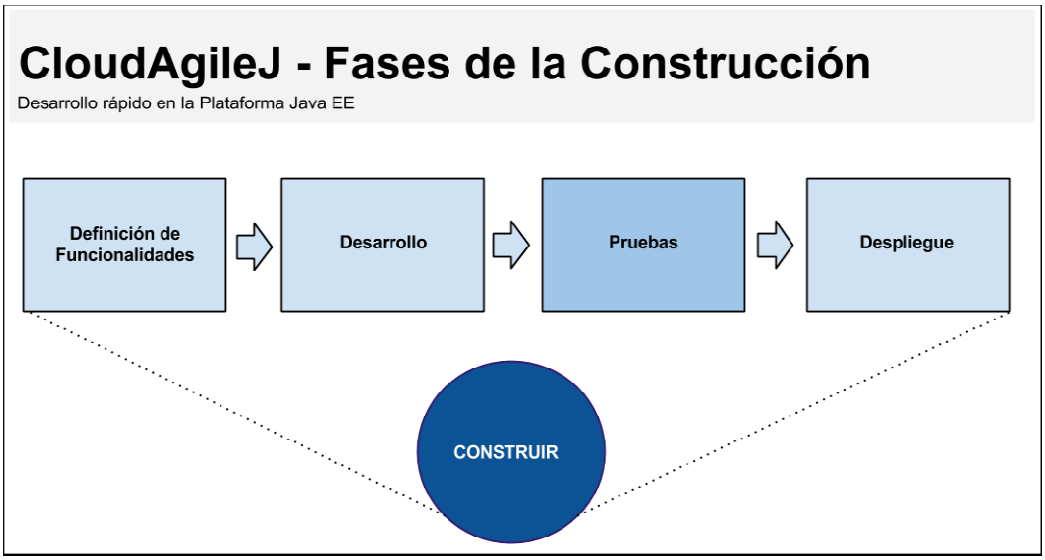

Figura 5.4-e – Fase de pruebas, iteración 1 (Fuente propia).

Se recuerda que en la fase de pruebas, visualizada en la *figura 5.4-e* no deben corresponder a pruebas exhaustivas, sino simplemente verificar el funcionamiento general y navegabilidad por las opciones del sistema con la finalidad de que no se bloquee en ninguna parte. De todas formas para esta versión del software todavía no está previsto que los usuarios carguen datos reales en el ella, sino que solamente datos de prueba.

Si se encuentran errores que corregir en esta parte, los desarrolladores deberían de darle prioridad cero, y corregirlos inmediatamente para volver a disponibilizarlos.

Se debe tener en cuenta que aquí también pueden surgir errores de infra-estructura (herramientas, frameworks, APIs, etc.), debido a las fallas del auto-generador, motivo por el cual el arquitecto también debe estar atento para realizar las correcciones necesarias en los mismos.

5.4.6. Paso 10 – Despliegue

¿Cuánto tiempo?: Medio día.

¿Dónde?: En la empresa de software.

¿Quiénes?: Gerente de Configuración, Arquitectos de Software.

Luego de concluir con la etapa de la construcción se debe disponibilizar el producto inicial en el servidor de pruebas para realizar la demostración al cliente. Este es el último paso y la regla es que sea realizado con la mínima intervención humana posible, valiéndose para ello de los scripts automáticos creados por los arquitectos en una fase anterior.

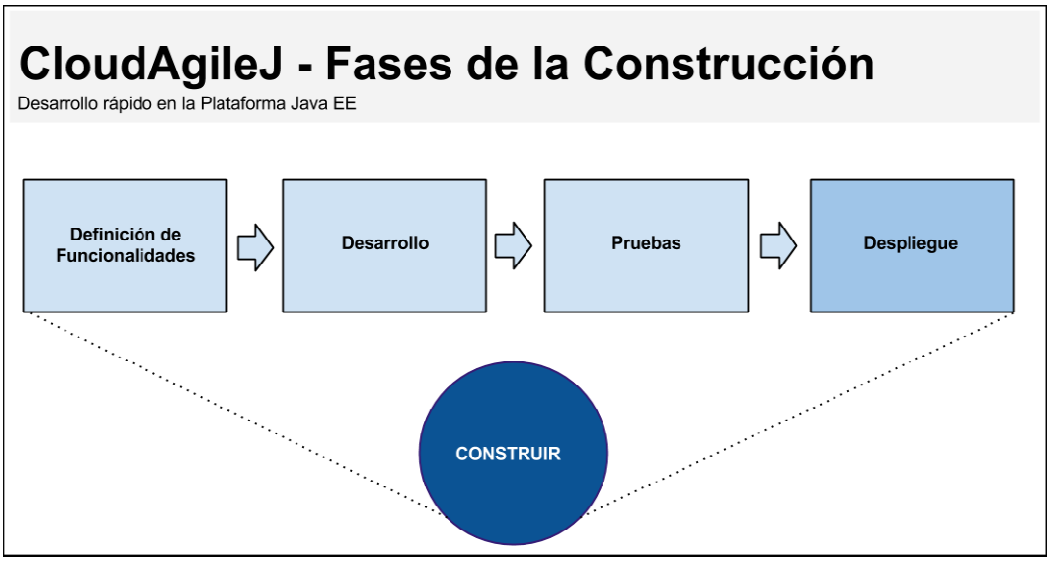

Figura 5.4-f – Fase de despliegue, iteración 1 (Fuente propia).

Esta fase *(ver figura 5.4-f)* debe concluir y darse por cerrado con una última evaluación por parte de los testers, para verificar simplemente la navegabilidad del sistema. Cabe recordar que el despliegue es realizado en un servidor de pruebas, como se debe a la versión de la primera iteración, el cual tiene como objetivo disponibilizarlo para obtener sugerencias por parte del cliente.

Se puede observar en la *figura 5.4-g* se concluye la fase de construcción, llegando al nodo de software, donde se posee un producto funcional para el usuario.

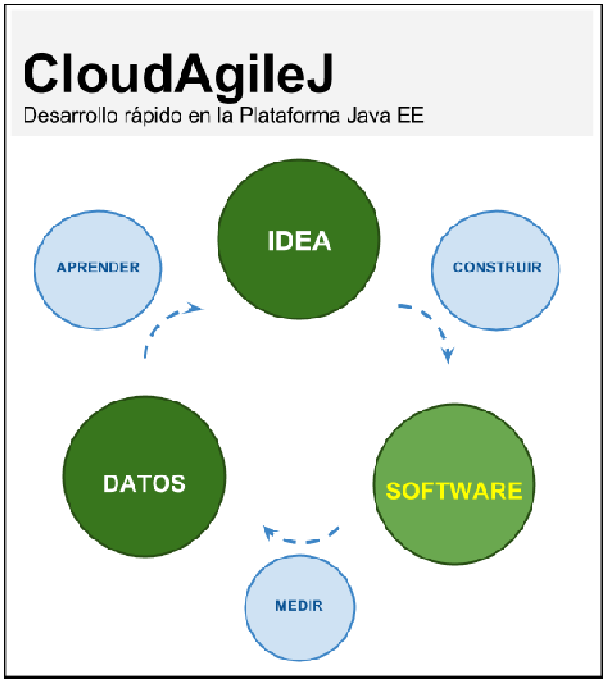

Figura 5.4-g – Nodo *software* de la iteración 1 (Fuente propia).

5.4.7. Paso 11 – Demostración del producto

¿Cuánto tiempo?: Medio día / un día. **Acumulado:** 12 días.

¿Dónde?: En la empresa de software u oficina del cliente.

¿Quiénes?: Líder de Proyecto, Analista de Sistemas.

Ha pasado un tiempo, corto, si se piensa que ya se tiene la versión del sistema para mostrar a los usuarios. El cliente debería sorprenderse por el corto tiempo transcurrido desde la reunión inicial.

Esta demostración debe servir al cliente, para que este pueda realizar la carga de los primeros datos en el sistema.

Mientras el analista va acompañando la utilización del sistema, el líder puede ir realizando las notas relacionadas a los cuestionamientos que surjan.

5.4.8. Paso 12 – Reunión para analizar situación

¿Cuánto tiempo?: Media mañana / tarde.

¿Dónde?: En la empresa de software.

¿Quiénes?: Líder de Proyecto, Analista de Sistemas, Desarrolladores.

Luego de la primera demostración del sistema al cliente, el equipo del proyecto, ya posee una visión más ampliada de la situación general, Ahora, tanto los analistas como desarrolladores tienen una visión más clara sobre el funcionamiento de los casos que fueron implementados, si se soluciona o no el problema presentado, qué puntos necesitan ser modificados o adaptados, una nueva lista de prioridades, y consiguen ver un poco más allá de las funcionalidades ya implementadas, porque tienen una nueva lista de funcionalidades no levantadas al inicio.

Esta reunión es de suma importancia, ya que a partir de ahora el líder del proyecto en conjunto con el analista, son capaces de trazar nuevas metas, organizarlas en conjunto con los desarrolladores, para establecer un siguiente punto de evaluación – hito, para el cual toda la iteración deberá repetirse.

Se debe recordar que se trabaja con metodologías ágiles, y se poseen las herramientas para realizar las refactorizaciones de códigos, necesarias para una rápida adaptación, por lo tanto los cambios que surgen a partir de aquí no deben representar muchos problemas.

A modo de guiar al lector por el proceso, se indica que en este momento se atraviesa por el momento de la *medición* de software, el cual se puede visualizar en la figura 5.4-h.

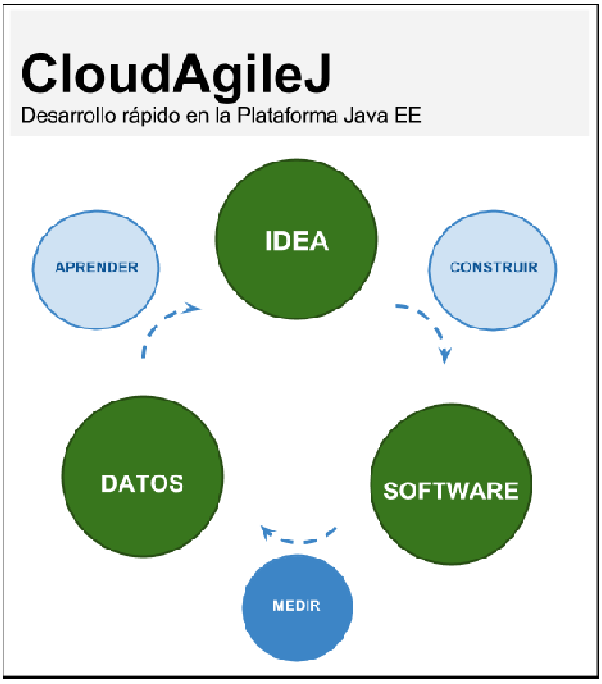

Figura 5.4-h – Nodo *medir* de la iteración 1 (Fuente propia).

5.4.9. Criterios para organización de datos

De la reunión realizada para *medir*, deben ser organizados los datos recolectados en la presentación del sistema. Las preguntas lanzadas aquí dependerán en gran medida de los indicadores especificados en el modelo inicial, pero para el caso específico del sistema realizado surgen algunas preguntas concretas. Es importante obtener básicamente los siguientes datos.

- ¿Qué funcionalidades solucionan los problemas presentados?.
- ¿Qué funcionalidades deben ser desechadas?.
- ¿En qué porcentaje las funcionalidades que solucionan los problemas cubren la necesidad?.
- De las funcionalidades que solucionan el problema, ¿hay algunos que necesitan dividirse en dos o más funcionalidades?

- Para la arquitectura también será importante saber en qué medida el auto-generador de código y las herramientas de automatización cubrieron las necesidades específicas así como también que cambios manuales tuvieron que ser realizados encima para adaptarlo a la necesidad específica de la empresa.

5.5. Etapa de aprendizaje

Se debe recordar que en este punto estamos en la etapa de los datos recabados de la medición *(ver figura 5.5-a)*.

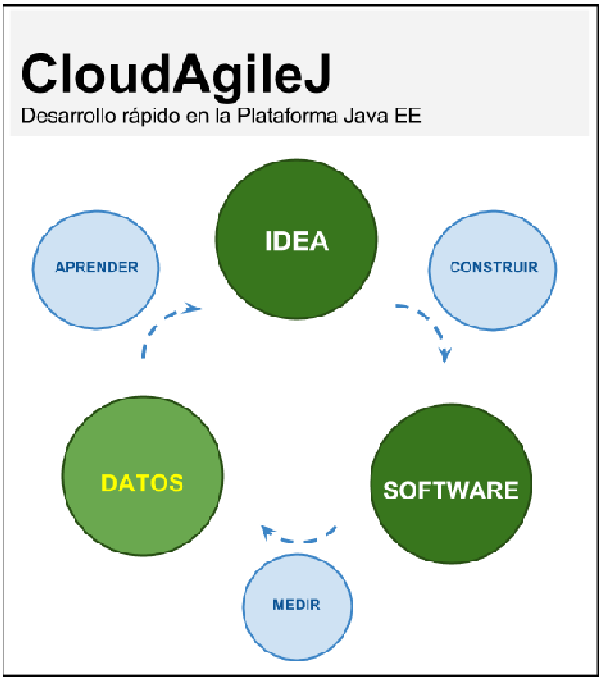

Figura 5.5-a – Nodo *datos* de la Iteración 1 (Fuente propia).

5.5.1. Paso 11 – Reunión de aprendizaje

¿Cuánto tiempo?: Medio día.

¿Dónde?: En la empresa de software.

¿Quiénes?: Líder de Proyecto, Analista de Sistemas, Desarrolladores, Arquitecto de Sistema.

Es hora de decidir realizar los ajustes necesarios sobre los puntos que fueron detectados en la medición.

Se trata de una breve reunión, donde se organizan y designan tareas para realizar los cambios necesarios, tanto en el sistema de contabilidad como en la infraestructura del sistema. Se debe recordar que se trabaja en base a 3 pilares fundamentales que consisten en las herramientas de la empresa, por lo tanto, surge la pregunta: ¿qué modificaciones requerirán las herramientas para mejorar la siguiente iteración?.

Estamos en la etapa de aprendizaje, donde se deben asumir ciertas acciones y aceptar los errores que ocurrieron para evitar que vuelvan a ocurrir futuramente.

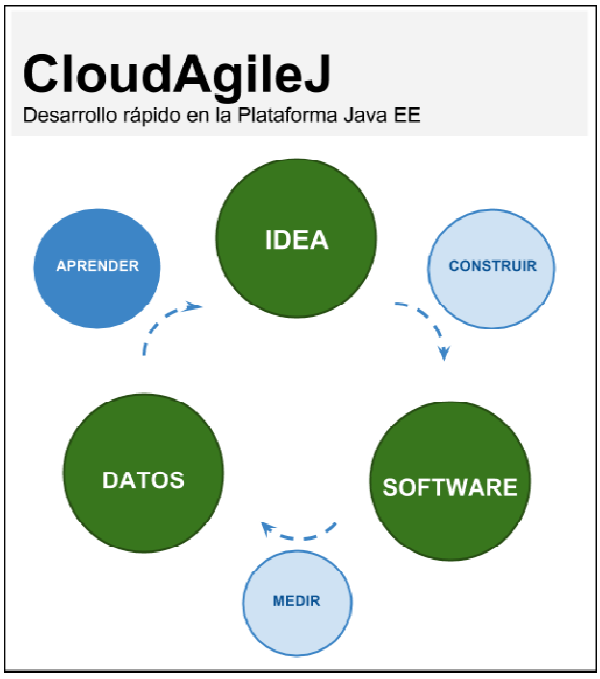

Figura 5.5-b – Nodo *aprender* de la iteración 1 (Fuente propia).

Muy probablemente, esta etapa *(ver figura 5.5-b)* requerirá su tiempo, más aún si se trata de la primera iteración del primer sistema.

Teniendo como base ahora, lo aprendido en la iteración anterior, se inicia de vuelta todo el proceso, y así sucesivamente hasta finalizarlo.

5.6. Discusiones y comentarios

En este capítulo se ejecutó el flujo de tareas comunes a todo desarrollo de software, donde se utilizaron de manera global todos los elementos que en su conjunto forman parte del modelo propuesto, y donde se pudo observar el orden cronológico de las acciones que deben ser encaradas por los diferentes miembros que están relacionados con la construcción del sistema.

Mediante el ejemplo se pudo notar la importancia de contar con los 3 pilares para el desarrollo rápido, con el objetivo de alcanzar los mismos tiempos utilizados en el ejemplo, los cuales resultaron bastante alentadores de acuerdo al tipo de sistema que se fue desarrollado.

Finalmente resulta importante destacar que la empresa de desarrollo debe involucrar a todos los integrantes con el objetivo de obtener el compromiso necesario para trabajar en la mejora continua del proceso, pues es ese el objetivo luego de cada iteración concluida y posterior entrega del producto ya que cada iteración tiene que resultar mejor que la anterior y así sucesivamente.

Capítulo VI

Conclusiones y trabajos futuros

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

6.1. Conclusiones

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.

6.2. 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.

Bibliografía utilizada

[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 Publico 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-seduce-tanto-a-las-empresas-tecnolgicas 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 susscceful 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., Las 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, Pagina del Profesor, Universidad de Murcia Disponible en http://www.um.es/docencia/barzana/IAGP/Iagp3.html, Universidad de Murcia, España, 2007.