

Metodología de Desarrollo para la Mejora de Procesos de Software a Través del Desarrollo Dirigido por Pruebas

Memoria de Tesis Doctoral

Documento en desarrollo.

10 de Noviembre 2013

Presentado por
Adolfo Centeno Téllez

Centro Interdisciplinario de Posgrados
Posgrado en Ingeniería de Software
Universidad Popular Autónoma del Estado de Puebla

Directora de tesis
Dra. María del Pilar Gómez Gil
Instituto Nacional de Astrofísica, Óptica Y Electrónica

Puebla, Puebla. México a 10 de Noviembre de 2013



UPAEP – Secretaría General

Dirección General de Apoyos Académicos

Dirección del Centro de Recursos para el Aprendizaje y la Investigación.

Biblioteca Central - **Karol Wojtyla**

Tesis Digitales Restricciones de uso:

DERECHOS RESERVADOS ©

PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis está protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de textos, imágenes, gráficas, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente de donde la obtuvo mencionando el autor o autores involucrados en el documento.

Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Resumen

Los problemas presentes en las empresas mexicanas de la construcción de software están íntimamente relacionados con la falta de procesos maduros y repetibles usados en la elaboración de sus productos. Esta falta de procesos institucionalizados se refleja en la mala calidad de los productos, entregas fuera de calendario y presupuestos rebasados.

En la actualidad se han desarrollado varias estrategias para solventar estas deficiencias. Por ejemplo, Moprosoft es el modelo de procesos para la industria de software mexicana que fue creado para apoyar en la solución de la problemática de la crisis de software; su objetivo es facilitar a las organizaciones dedicadas al desarrollo y mantenimiento de software la adopción de las mejores prácticas reconocidas internacionalmente a través de modelos como: SW-CMM, CMMi, PSP, TSP, ISO/IEC 15504, PMBOK y SWEBOK. [Ruvalcaba, 2005]. Moprosoft dio origen a la norma mexicana de mejora de procesos NMX-I-059/02-NYCE-2005, la cual ha estado vigente en México desde el 2005. Por otro lado, se han creado modelos que tienden a mejorar la calidad del desarrollo. Entre éstos se encuentra el desarrollo dirigido por pruebas (*Test Driven Development* o TDD), el cual es una de las 12 prácticas claves de la Programación Extrema. TDD tiene por objetivo crear código de alta calidad que funcione correctamente, a través de la siguiente secuencia de pasos: escribir pruebas unitarias antes que el código, generar el código más simple que pase las pruebas y finalmente refactorizar de manera incremental.

En este trabajo se analiza a fondo el desarrollo dirigido por pruebas y la mejora de procesos de software, se investiga su aplicación y se propone una metodología formal de desarrollo de software llamado DEPYM (por las siglas en español de **M**odelo de **D**esarrollo Dirigido por **P**uebas y **M**ejora de **P**rocesos) que se inspira en la norma de mejora de procesos NMX-I-059/02-NYCE-2005 y se adapta en el diseño ágil y la metodología de desarrollo por Pruebas. Esta investigación se enfoca en la Gestión del proyecto y el proceso de Desarrollo de Software con enfoque Ágil, a fin de enfrentar y proponer soluciones a la problemática de implementación de estos procesos.

Para la validación de la investigación se seleccionaron 3 proyectos de empresas PyMEs del estado de Veracruz, donde se aplicó el modelo en 3 proyectos con presupuesto de tiempo menor a 18 meses y equipos de desarrollo menores de 9 integrantes. Para evaluar el

modelo se definieron 4 métricas: valor entregado al cliente, velocidad de desarrollo, horas pendientes en la iteración y número de defectos por KLOC. Los resultados obtenidos en los proyectos 1 y 2 completamente terminados, fueron favorables de acuerdo a las métricas establecidas. El proyecto 3 se contaba con un 25% de avance a la fecha de preparación de este manuscrito, con resultados también satisfactorios

Índice General

Contenido

Resumen	2
Índice de Figuras.	6
Índice de Tablas.	6
1. Introducción	7
1.1. Definición del Problema	8
1.2. Planteamiento y justificación del trabajo	9
1.3. Hipótesis y objetivos	10
1.4. Metodología de investigación	11
1.5. Validación de la investigación	11
2. Conceptos básicos y Estado del Arte	14
2.1. Modelos Agiles.	15
2.1.1. Definición.	15
2.1.2. Tipos de modelos existentes.	16
2.1.2.1. Programación Extrema.	21
2.1.2.2. Scrum.	22
2.2. Moprosoft.	25
2.2.1. Definición.	25
2.2.2. Descripción.	26
2.3. Desarrollo dirigido por pruebas	29
2.3.1. Definición.	29
2.3.2. Descripción.	29
2.3.3. Ventajas actuales.	30
2.4. Modelo MVC (Model View Controller)	31
2.4.1 Definición.	31
2.4.2 Descripción	32
2.4.3 Herramientas actuales que usan MVC	33
3. Modelo DEPYM	38
3.1 Introducción	38
3.2 Descripción General	38
3.3 Patrón de proceso.	43

3.4	Script general del proceso DEPYM.....	45
3.5	Etapa de Configuración inicial.....	48
3.6	Etapa de planeación.	51
3.7	Etapa de Requerimientos.....	54
3.8	Etapa de Generación de Pruebas Unitarias de DEPYM	56
3.9	Etapa de Codificación de DEPYM.....	58
3.10	Etapa de Cierre del Modelo DEPYM	59
4.	Resultados y discusión.....	65
4.1.	Descripción de los experimentos de prueba.....	65
4.2.	Resultados de la aplicación del modelo.....	66
4.2.1.	Caso 1.....	66
4.2.2.	Caso 2.....	68
4.2.3.	Caso 3.....	70
5.	Conclusiones.....	75
5.1.	Aportaciones.....	76
5.2.	Consideraciones finales	76
5.3.	Trabajo futuro	77
	Publicaciones.....	78
	Bibliografía.....	79
6.	Apéndice A. Formas del modelo DEPYM.....	83
7.	Apéndice B. Estándares del modelo DEPYM	88
8.	Apéndice C. Ejemplo de aplicación del modelo DEPYM.....	Error! Bookmark not defined.

Índice de Figuras.

Figura 1.1.- Metodología de investigación	12
Figura 2.1. Proceso general de la metodología XP.	23
Figura 2.2. Proceso de Scrum.	25
Figura 2.3. Diagrama de categorías de procesos.	27
Figura 2.4. Modelo de procesos del Desarrollo Dirigido por pruebas.	31
Figura 2.5. Diagrama de Modelo MVC (Model View Controller).	32
Figura 3.1. Diagrama general del modelo DEPYM.....	40
Figura 3.2. Diagrama en bloques del proceso de desarrollo del modelo DEPYM.	41

Índice de Tablas.

Tabla 1.1. Empresas Certificadas en Modelos de Calidad a Diciembre 2010. [SNIITI, 2012]	10
Tabla 2.1. Herramientas actuales que usan el patrón MVC.....	33
Tabla 3.1. Formas propuestas por el modelo DEPYM.....	42
Tabla 3.2.- Estándares propuestos por el modelo DEPYM.	43
Tabla 3.3. Estructura del patrón de procesos del Modelo DEPYM.	44
Tabla 3.4. Script General de procesos del modelo DEPYM.	46
Tabla 3.5. Script de configuración inicial del modelo DEPYM.....	49
Tabla 3.6. Script de configuración inicial del modelo DEPYM.....	51
Tabla 3.7. Script de Requerimientos del modelo DEPYM.....	55
Tabla 3.8. Script de Generación de Pruebas unitarias del modelo DEPYM.....	57
Tabla 3.9. Script de Generación de Pruebas unitarias del modelo DEPYM.....	58
Tabla 3.10. Script de Cierre del modelo DEPYM.	60
Tabla 4.1. Resultados obtenidos del proyecto 1 del experimento de evaluación de DEPYM. ..	67
Tabla 4.2. Resultados obtenidos del proyecto 2 del experimento de evaluación de DEPYM	69
Tabla 4.3. Resultados obtenidos del proyecto 3 del experimento de evaluación de DEPYM. ...	72

1. Introducción

En un mundo de cambios constantes y competencia global, las organizaciones de desarrollo de software son presionadas a alcanzar mayor eficiencia con menores costos. Para poder lograr este objetivo, es necesario adoptar una forma de trabajo que permita entender, controlar, comunicar, mejorar, predecir y certificar el trabajo realizado. Desafortunadamente la industria del software se encuentra en crisis. Janzen comenta que, a pesar de medio siglo de avances, la industria de construcción de software continúa mostrando signos de inmadurez [Janzen, 2005]. Las organizaciones de desarrollo de software continúan esforzándose por producir software confiable de forma predecible y repetitiva. Mientras una variedad de prácticas de desarrollo son defendidas y podrían mejorar la situación, los desarrolladores se rehúsan a adoptar nuevas y mejores prácticas, usando solo prácticas basadas en evidencias anecdóticas. Las evidencias empíricas de prácticas eficaces no están disponibles o completamente definidas, y el adoptar nuevas prácticas consume tiempo, esfuerzo y es riesgoso [Janzen, 2005].

La mejora de procesos de software sugiere cambios que puede realizar una organización para obtener mejor desempeño en sus procesos de producción de software en cuando a tiempos, costos y calidad. Actualmente existe una gran variedad de modelos para la mejora de procesos de software. Estos modelos pueden clasificarse como genéricos o específicos [Ruvalcaba, 2005], y se describen a continuación:

Modelos Genéricos: Abarcan todos los procesos relacionados con el desarrollo de software, se usan como referencia para definir procesos en una organización y autoevaluación. Ejemplos de éstos son [Ruvalcaba, 2005]:

- **CMM (*Capability Maturity Model*)**. Modelo de Madurez y Capacidad de Procesos, describe elementos clave de procesos efectivos de software; fue creado por el *Software Engineering Institute (SEI)* en conjunto con *Carnegie Mellon University*
- **CMMi**. Es una iniciativa que reúne los diferentes modelos CMM's; incluye: CMM software, CMM Ingeniería de Sistemas y CMM Desarrollo integrado de Producto.

- ISO 9001-2000. Es un conjunto de estándares creados por ISO (*International Standards Organization*) que establecen los requerimientos para la gestión de Sistemas de Calidad
- ISO/IEC 15504. Es un estándar internacional que ofrece un marco para la evaluación de procesos. Este modelo se usa como guía de referencia para la evaluación de múltiples modelos (de software o no)
- Moprosoft. Significa “modelo de procesos para la industria de software mexicana.” Fue creado para facilitar la adopción en México de las mejores prácticas reconocidas internacionalmente a través de modelos como: SW-CMM, CMMi, PSP, TSP, ISO/IEC 15504, PMBOK y SWEBOK

Modelos Específicos: Están enfocados a la ingeniería de productos de software y se usan como guía para ejecutar proyectos:

- RUP. (*Rational Unified Process*) modelo de procesos desarrollado por Rational Software (IBM) que incluye 6 prácticas: desarrollo iterativo, requerimientos, arquitectura basada en componentes, modelación visual, verificación continua de calidad y control de cambios.
- PSP (*Personal Software Process*) (PSP) modelo basado en las mejores prácticas de CMM y diseñado para ayudar de forma personal a los ingenieros de software a controlar, manejar y mejorar su trabajo, propuesto por Watts Humphrey
- TSP (*Team Software Process*) basado en PSP, enseña a los ingenieros a construir equipos auto-dirigidos y desempeñarse como un miembro efectivo dentro del equipo. También muestra a los administradores como guiar y soportar estos equipos, propuesto por Watts Humphrey.

1.1. Definición del Problema

Los problemas presentes en las empresas mexicanas de la construcción de software están íntimamente relacionados con la falta de procesos maduros y repetibles usados en la elaboración de sus productos. Esta falta de procesos institucionalizados se refleja en la mala calidad de los productos, entregas fuera de calendario y presupuestos rebasados. El problema presente en el uso no efectivo de métodos de desarrollo de software genera pérdidas económicas importantes en las organizaciones. Para evitar estas pérdidas, se han desarrollado proyectos por parte del gobierno federal como la Iniciativa Nacional *TSP/PSP*

que consiste en una estrategia de la Secretaría de Economía para crear los recursos humanos que permitan la expansión acelerada de *TSP/PSP*. Esto se realiza por medio de un convenio directo con el SEI, el Dr. Watts Humphrey [SNIITI, 2009] y el proyecto *MexicoFIRST* (*Federal Institute for Remote Services and Technologies*). *MexicoFIRST* es una iniciativa de carácter nacional que tiene por objetivo promover y facilitar la capacitación y certificación de recursos humanos para llevar a cabo actividades de TI y *BPO* (*Business Process Outsourcing*). [Bujaidar, 2008]. Sin embargo, estas iniciativas no han sido suficientes a la fecha, debido a que la mayoría de las empresas continúan trabajando con procesos inmaduros y no institucionalizados. Muchas empresas que han recibido capacitación en PSP/TSP, salvo algunas excepciones, no aplican formalmente estas metodologías, debido a que no fueron pensadas para empresas pymes mexicanas. Además, algunas empresas han recibido la verificación en Moprosoft nivel 1 o 2 y no ocupan al 100% los procesos de este modelo, sino solo una versión limitada a sus necesidades. De igual forma, el número de empresas con certificación CMMi en alguno de sus niveles es muy reducido y las empresas que usan métodos ágiles como XP o SCRUM no lo hacen de manera formal.

Para ayudar a resolver esta problemática, este trabajo propone un modelo de desarrollo innovador dirigido por pruebas, adaptable a las empresas mexicanas, el cual utilice las ventajas del conocimiento de las empresas en los métodos ágiles.

1.2. Planteamiento y justificación del trabajo

Hasta Septiembre del 2013 se habían registrado 371 centros de desarrollo de software evaluados en algún proceso de calidad en 21 Estados de la República Mexicana: Aguascalientes, Baja California, Chihuahua, Coahuila, Colima, Distrito Federal, Hidalgo, Jalisco, México, Michoacán, Nuevo León, México, Oaxaca, Puebla, Querétaro, Sinaloa, Sonora, Tlaxcala, Veracruz, Yucatán y Zacatecas [SNIITI, 2012]. La tabla 1.1 muestra la distribución de las empresas mexicanas evaluadas en 3 normas de calidad. Allí puede notarse que existen 218 empresas de las 302 evaluadas que han sido aprobadas en la norma mexicana NMX-I-059/02-NYCE-2005, lo que constituye el 65.22% del total nacional evaluado. De las 371 empresas Moprosoft, 255 están solo en el nivel 1, lo que representa el

68.73% de las empresas evaluadas. Entonces, el 68.73% de empresas mexicanas interesadas en certificación cuenta solamente con los procesos mínimos requeridos para terminar un proyecto de software, sin tomar en cuenta aún aspectos más formales de calidad, costos y tiempos de entrega.

Consideramos que esta problemática se debe, entre otras razones, al hecho de que las empresas mexicanas no han identificado una metodología que les permita manejar un desarrollo ágil al mismo tiempo que les permita usar procesos de desarrollo de software maduros y eficaces. Por esta razón, este trabajo propone una nueva metodología con un enfoque de desarrollo dirigido por pruebas, aplicable al proceso de desarrollo de Moprosoft conocido como “Gestión de Proyecto”. La metodología presenta un enfoque ágil como parte integral del proceso de desarrollo de software de las empresas mexicanas, cumpliendo con sus lineamientos de la ingeniería de productos de software

Es importante resaltar que esta investigación impacta en el área económica, ya que el modelo propuesto ayuda a las empresas a crear productos de mejor calidad, lo que se verá reflejado en el éxito de sus proyectos y mejora en su competitividad.

Aunque las técnicas del desarrollo dirigido por pruebas no son nuevas, lo innovador del modelo de desarrollo usando este enfoque es su adaptabilidad a las necesidades y tamaño de las empresas PYMES mexicanas desarrolladoras de software y que abarca tanto la gestión del proyecto como el proceso de desarrollo del producto.

Tabla 1.1. Empresas Certificadas en Modelos de Calidad a Septiembre 2013. [SNIITI, 2013]

Modelo/Nivel	1	2	3	4	5	Total
CMM	0	2	6	2	4	8
CMMi	0	14	13	2	7	84
Moprosoft	255	112	4	0	0	371

1.3. Hipótesis y objetivos

Nuestra hipótesis básica de investigación es:

“Es posible mejorar la calidad de los productos de software de las empresas mexicanas con un modelo ágil de desarrollo dirigido por pruebas enfocado en los procesos de 'Gestión de Proyectos y desarrollo de software'.”

El objetivo principal de este trabajo es crear un modelo de desarrollo de software, llamado DEPYM (Metodología de **D**esarrollo Dirigido por **P**uebas y Mejora de procesos para **PyMes**), basado en la técnica de desarrollo dirigido por pruebas que permita a empresas mexicanas mejorar su desempeño.

Los objetivos específicos del trabajo son los siguientes:

1. Caracterizar las fortalezas y debilidades de las empresas PYMES mexicanas, con respecto a su metodología de desarrollo de software,
2. Crear el modelo DEPYM y definir su estrategia de implementación a fin de que potencialice las fortalezas de las empresas.

1.4. Metodología de investigación

En el presente trabajo se analiza a fondo el desarrollo dirigido por pruebas y la mejora de procesos de software, se investiga su aplicación y se propone una modelo formal de desarrollo de software que se adapta en el diseño ágil y la metodología de desarrollo por Pruebas. Para alcanzar los objetivos definidos, se realizarán los pasos descritos en la figura 1.1.

1.5. Validación de la investigación

Para definir el método de evaluación del modelo DEPYM se utilizara la medición de resultado del experimento, donde se tomaran en cuenta métricas de acuerdo a la naturaleza de los métodos ágiles como:

- Valor entregado al cliente.
- Velocidad de desarrollo.
- Horas pendientes en la iteración.

- Numero de defectos por KLOC (porcentaje de defectos por cada 1000 líneas de código).

Para su validación se realizara una serie de experimentos dentro de un ambiente real, a través de las siguientes actividades que se muestran en la figura 1.1.

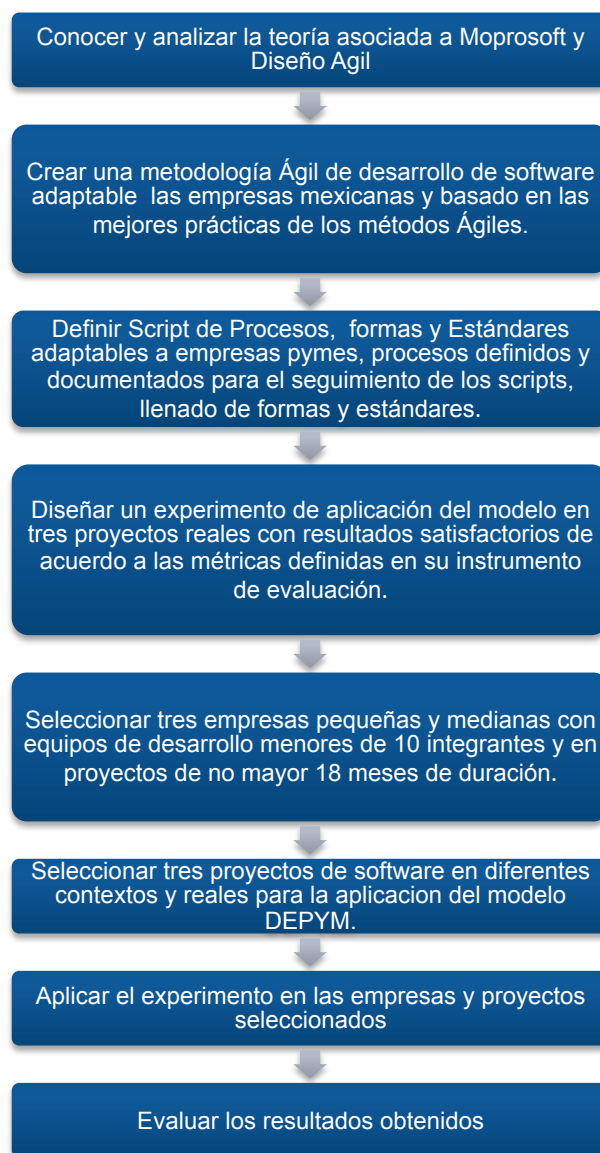


Figura 1.1.- Metodología de investigación

Para medición de los resultados del experimento se recolectaron las métricas que se describen en la sección 3.11, las cuales se analizarán estadísticamente.

Este documento está organizado de la siguiente manera: El capítulo 2 describe conceptos básicos que se usarán en el resto del documento, así como el estado del arte de los modelos Ágiles y tradicionales en los que se inspira esta investigación. El capítulo 3 describe de manera formal el modelo DEPYM, su patrón de procesos, *scripts*, formas y estándares, El capítulo 4 describe el experimento de evaluación y reporta los resultados obtenidos en los 3 casos evaluados. Finalmente el capítulo 5 describe las aportaciones de la investigación, conclusiones y trabajo futuro.

2. Conceptos básicos y Estado del Arte

En este capítulo se describen algunos conceptos necesarios para el entendimiento del resto del documento. Asimismo, se resumen algunos de los trabajos más importantes relacionados con el ámbito de esta investigación: desarrollo de metodologías ágiles, desarrollo dirigido por pruebas y sistemas de mejora de procesos en empresas pequeñas.

2.1 Conceptos Básicos

La Ingeniería de Software fue definida por Bauer a finales de los 60s como el establecimiento y uso de principios de ingeniería para obtener software confiable y funcional [Bauer, 1972], por lo que se le considera una disciplina relativamente joven que todavía está en estudio y evolución. La Ingeniería de Software tiene varias definiciones, siendo una de las más sencillas: “el establecimiento y aplicación de principios de la Ingeniería para obtener software” [Pressman, 2005]. Entonces, en términos generales, la ingeniería de software es una disciplina de la ingeniería que trata todos los aspectos de la producción de software, teniendo en cuenta factores como el costo económico, fiabilidad, rendimiento del sistema y un funcionamiento eficiente que satisfaga las necesidades del usuario, así como la aplicación de procesos a lo largo del ciclo de vida del software [Pressman, 2005].

Dentro de la Ingeniería de software se han propuesto un gran número de metodologías para el desarrollo de software, que inciden en distintas dimensiones del proceso de desarrollo. Por una parte existen propuestas que se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán. Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en otros muchos. A fin de resolver estos problemas, en los últimos años se han popularizado otro tipo de metodologías llamadas metodologías ágiles, que presentan buenas expectativas en algunos contextos de desarrollo. La siguiente sección describe detalladamente estas metodologías.

2.2. Modelos Ágiles.

Las metodologías ágiles tienen como característica fundamental el hecho de que dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Este enfoque ha mostrado su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad.

2.2.1. Definición.

El término “ágil” se refiere a una forma innovadora de desarrollar software, basada en prácticas como el desarrollo iterativo, entregas continuas, desarrollo dirigido por pruebas y fuerte involucramiento del cliente. Esta agilidad contrasta con algunas formas tradicionales de desarrollar software, en cuanto a que ésta es más flexible y adaptable que aquellas [Highsmith, 2001]. De acuerdo a Highsmith [Highsmith, 2001] y Cockburn [Cockburn, 2002], la esencia de los procesos ágiles es la gente y por consiguiente es el principal factor de éxito en los proyectos.

Las necesidades de un cliente pueden sufrir cambios importantes del momento de contratación de un proyecto al momento de su entrega; esto requiere procesos de software que, en lugar de rechazar los cambios, sean capaces de incorporarlos. Estos procesos deben conducir a los siguientes objetivos:

- Producir versiones ejecutables de software en pocas semanas, con el fin de satisfacer al cliente y obtener retroalimentación en cuanto al producto.
- Inventar soluciones sencillas, de tal forma que el impacto de los cambios se minimice.
- Mejorar la calidad en la etapa del diseño de manera continua, haciendo que la siguiente iteración requiera menos esfuerzo que la actual.
- Probar el producto desde etapas tempranas y de manera continua, para encontrar defectos antes de que tengan un alto impacto.

Según Highsmith [Highsmith, 2001] las organizaciones de TI pasan por tres etapas de madurez en cuanto a su forma de administrar proyectos de software. Empiezan por la etapa de caos o ausencia de control; posteriormente deciden adoptar metodologías con un alto grado de definición, documentación y puntos de control y/o autorización, esto funciona por un

tiempo o para cierto tipo de proyectos. Sin embargo cuando las organizaciones se enfrentan con proyectos más innovadores, con nuevas tecnologías o con requerimientos cambiantes, la estrategia de control prescrito tiende a fracasar. Es entonces que las organizaciones entienden la necesidad de administrar estos proyectos de una forma más flexible y adaptable. Estas 3 etapas son típicas de grandes organizaciones de Tecnología de Información. En el caso de organizaciones más pequeñas, sobre todo aquellas donde el giro principal del negocio es el software, la tendencia es pasar directamente del caos a los procesos ágiles. En cambio, en las áreas de sistemas en empresas de otro giro, el ritmo de adopción de métodos ágiles es más lento.

De acuerdo a Jim Highsmith, las fases de un proyecto ágil son las siguientes [Highsmith, 2001]:

- **Visualizar.** Se genera una visión colectiva sobre el producto de software a desarrollar.
- **Especular.** Se establecen hipótesis sobre las especificaciones del producto, sabiendo que conforme el proyecto avance, éstas irán evolucionando.
- **Explorar.** Se implementan las especificaciones de forma iterativa.
- **Adaptar.** Se analizan los resultados de dichos experimentos y se realizan los ajustes necesarios para las siguientes iteraciones. Se evalúan cuatro aspectos: funcionalidad del producto, calidad técnica del producto, estatus del proyecto y desempeño del equipo.

2.1.2. Tipos de modelos existentes.

El enfoque ágil da gran importancia a la flexibilidad, ya que está indicado para proyectos con requisitos poco definidos o cambiantes. Las metodologías ágiles se aplican bien en equipos pequeños que resuelven problemas concretos, lo que no está reñido con su aplicación en el desarrollo de grandes sistemas, ya que una correcta modularización de los mismos es fundamental para su exitosa implantación. Dividir el trabajo en módulos abordables minimiza los fallos y el costo. Estas prácticas se repiten en todas las metodologías ágiles existentes. Existe una gran variedad de metodologías ágiles; entre las más destacadas actualmente se encuentran:

- **Programación extrema (XP – *Extreme Programming*)**
 - La programación extrema o *eXtreme Programming* (XP) [Beck, 1999] es una metodología de desarrollo creada por Kent Beck, autor del primer libro sobre

la materia, *Extreme Programming Explained: Embrace Change* (1999). Según Beck, éste 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 XP consideran que los cambios de requisitos a lo largo del proyecto son un aspecto natural, inevitable e incluso deseable. 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.

• Scrum

- Según Schwaber [Schwaber, 2010], SCRUM es un modelo de referencia que define un conjunto de prácticas y roles, que puede tomarse como punto de partida para definir el proceso de desarrollo que se ejecutará durante un proyecto. Los roles principales en Scrum son el *ScrumMaster*, que mantiene los procesos y trabaja de forma similar al director de proyecto, el *ProductOwner*, que representa a los *stakeholders* (externos o internos), y el *Team* que incluye a los desarrolladores. Durante cada *sprint*, que es un periodo entre una y cuatro semanas (la magnitud es definida por el equipo), el equipo crea un incremento de software *potencialmente entregable* (utilizable). El conjunto de características que forma parte de cada *sprint* viene del *Product Backlog*, que es un conjunto de requisitos de alto nivel priorizados que definen el trabajo a realizar. Los elementos del *Product Backlog* que forman parte del *sprint* se determinan durante la reunión de *Sprint Planning*. Durante esta reunión, el *Product Owner* identifica los elementos del *Product Backlog* que quiere ver completados y los hace del conocimiento del equipo. Entonces, el equipo determina la cantidad de ese trabajo que puede comprometerse a completar durante el siguiente sprint. Durante el sprint, nadie puede cambiar el Sprint Backlog, lo que significa que los requisitos están congelados durante el sprint.

• Crystal Clear

- *Crystal Clear* no es una metodología en sí misma sino una familia de metodologías con un “código genético” común [Cockburn, 2004]. La idea es poder armar distintas metodologías para distintos tipos de proyectos. Cada proyecto y organización usará este código genético para generar su propia metodología. El nombre *Crystal* deriva de la caracterización de los proyectos según 2 dimensiones, tamaño y complejidad (como en los minerales, color y dureza). Por ejemplo, *Clear* es para equipos de hasta 8 personas, “amarillo” para equipos entre 10 – 20 miembros, “naranja” para 20 a 25 personas. Este modelo ve al desarrollo de software como una serie de partidos que consisten en inventar y comunicar. Cada partido es diferente y tiene como objetivo entregar software y prepararse para el siguiente juego. Esto permite al equipo trabajar concentrado y en forma efectiva con un objetivo claro cada vez. *Crystal Clear* establece un conjunto de prioridades y principios que sirven de guía para la toma de decisiones:
 - *Eficiencia en el desarrollo*: para hacer que los proyectos sean económicamente rentables.
 - *Seguridad* en lo que se entrega.
 - *Habitabilidad*: hacer que todos los miembros del equipo adopten y sigan las convenciones de trabajo establecidas por el equipo mismo.
 - *Frecuencia en las entregas*: entregar al usuario funcionalidad “usable” con una frecuencia de entre 2 semanas y no más de un mes
 - *Comunicación*: *Crystal Clear* toma como uno de sus pilares a la comunicación. Promueve prácticas como el uso de pizarrones, pizarras y espacios destinados a que todos (miembros del equipo y visitas) puedan ver claramente el progreso del trabajo
 - *Crecimiento reflexivo*: es necesario que el equipo lleve a cabo reuniones periódicas de reflexión que le permitan crecer y hacerse más eficientes.

DSDM – Dynamic Systems Development Method

- Según Davies [Davies, 2003] DSDM es el acrónimo de “modelo de procesos para el desarrollo de sistemas de software,” desarrollado y concebido por el denominado DSDM Consortium, que se fundó en Inglaterra en 1994, y que actualmente tiene presencia en Inglaterra, EE.UU. Dinamarca, Francia y Suiza y con interés y contactos para futuras representaciones en Australia, India y China. Tomando como fuente las bases teóricas y las experiencias de RAD (Rapid Application Development) el consorcio se fundó en enero de 1994 con la finalidad de desarrollar un modelo de desarrollo independiente de herramientas y que fuera de dominio público. Entre sus características tenemos:
 - La implicación activa de los usuarios es imprescindible. Los miembros de los equipos de desarrollo DSDM deben tener la autonomía y potestad necesarias para tomar decisiones.
 - Entrega frecuente de incrementos operativos del producto, el principal criterio de prioridad, desarrollo y validación de las entregas incrementales es el objetivo y la salud del negocio.
 - El desarrollo iterativo o incremental hace posible obtener la solución más adecuada a las necesidades del negocio, todos los cambios realizados en el desarrollo son reversibles.
 - Los requisitos se establecen a un nivel general.
 - Las pruebas forman parte del ciclo de desarrollo.
 - Es imprescindible trabajar con espíritu de colaboración con todos los agentes implicados en el sistema que se desarrolla.
 - Etapas:
 - Pre-proyecto
 - Estudio de viabilidad
 - Estudio de negocio
 - Iteración de modelado funcional
 - Iteración de diseño y desarrollo
 - Implementación
 - Post-desarrollo

- **FDD – Feature Driven Development**

- Es una metodología ágil para el desarrollo de sistemas [Palmer, 2002], basada en la **calidad** del software, que incluye un monitoreo constante de proyecto. *FDD* fue desarrollado por Jeff De Luca y Peter Coad a mediados de los años 90. Esta metodología se enfoca en:
 - Iteraciones cortas que permite entregas tangibles del **producto** en cortos periodos de tiempo que como máximo son de dos semanas.
 - No hace énfasis en la obtención de los requerimientos sino en cómo se realizan las fases de diseño y construcción.
 - Se preocupa por la calidad, por lo que incluye un monitoreo constante del proyecto. Ayuda a contrarrestar situaciones como el exceso en el presupuesto, fallas en el programa o el hecho de entregar menos de lo deseado.
 - Propone tener etapas de cierre cada dos semanas. Se obtienen resultados periódicos y tangibles.
 - Se basa en un proceso iterativo con entregas cortas que producen un software funcional que el cliente y la dirección de la empresa pueden ver y monitorear. Define claramente entregas tangibles y formas de evaluación del progreso del proyecto

- **ASD – Adaptive Software Development**

- Modelo de implementación de patrones ágiles [Highsmith, 2000] para desarrollo de software, diseñado por Jim Highsmith; el proyecto es orientado y guiado por la misión del proyecto. Sus características son:
 - Basado en Funcionalidades
 - Desarrollo Iterativo
 - Desarrollo acotado temporalmente
 - Guiado por los riesgos
 - Trabajo tolerante al cambio

- **Xbreed**

Xbredd también es conocido como *Agile Enterprise* [Bioul, 2010]. Propuesto por Mike Breedle, que colaboró con Ken Schwaber en la definición de Scrum, es una combinación de *Scrum* para la gestión del proyecto, y *Extreme Programming* como prácticas de desarrollo. Ésta es una combinación comúnmente empleada independientemente de su definición como *Xbreed* que hasta la fecha no ha tenido especial reelevancia.

Enseguida se explica a mayor profundidad XP y Scrum ya que son las más populares y cercanas a la metodología propuesta en esta investigación.

2.1.2.1. Programación Extrema.

La programación extrema (XP por sus siglas en inglés: *Extreme Programming*.) es una disciplina de desarrollo de software basada en la simplicidad, comunicación, retroalimentación y entregas frecuentes; fue introducida por Kent Beck en 1996 [Beck, 2002]. XP utiliza prácticas sencillas con el equipo de desarrollo y con suficiente retroalimentación para motivar al equipo a conocer el estado real del proyecto [Ruvalcaba, 2005]. XP está formado por valores, principios y prácticas. Las prácticas son actividades concretas que un equipo puede realizar día a día, los valores representan el conocimiento fundamental que sustenta dichas prácticas. Tanto los valores como las prácticas son necesarios, pero hay un gran espacio entre ambos; esta brecha es cubierta por los principios. La edición 2004 de XP define cinco valores, catorce principios, trece prácticas primarias y once prácticas adicionales [Beck, 2002]. Los cinco valores son:

- **Comunicación.** La comunicación entre los miembros del equipo y los clientes, se debe maximizar.
- **Simplicidad.** Usar la solución más sencilla que pueda funcionar.
- **Retroalimentación.** Siempre de ser posible, medir el producto que se está desarrollando, y conocer que le falta para satisfacer los requerimientos.
- **Coraje.** Es necesario armarse de valor para incorporar cambios durante un proyecto. El coraje por sí solo es peligroso, pero sustentado con comunicación, simplicidad y retroalimentación, es una poderosa herramienta.

• **Respeto.** Los valores anteriores implican respeto. Ninguna metodología puede funcionar mientras los miembros del equipo no se tengan respeto entre sí o al trabajo que realizan.

Entre las prácticas más significativas de XP están: el diseño incremental, programación en pares, integración continua de software (cada dos horas) y programación tipo “*test-first*” (antes de desarrollar código nuevo, crear las pruebas que van a verificar el código actual).

La Figura 2.1 muestra las principales actividades dentro un proyecto usando la metodología XP, desde la obtención del alcance del producto, la construcción de las historias de usuario, la definición de las tareas y finalmente la creación del producto usando desarrollo dirigido por pruebas [Schach, 2008].

2.1.2.2. Scrum.

Scrum toma su nombre de una jugada de rugby; fue creado originalmente por Ken Schwaber en el año 1995, quien lo define como “caos controlado” [2004, Schwaber]. Scrum, como todas las metodologías ágiles, es de naturaleza iterativa y pretende crear un producto listo al final de cada iteración. De acuerdo a Scrum el desarrollo de software, siendo una actividad tan profundamente intelectual y que necesita tanta creatividad, es un mal candidato al proceso definido. El desarrollo de software, entonces, no debería entenderse como una línea de montaje, sino como una serie de inspecciones constantes acompañadas de correcciones inmediatas. Schwaber llama a esta forma de trabajar al proceso *caótico*, y nota que espontáneamente, los equipos de trabajo se coordinan con muy poca intervención de entidades externas. Schwaber llama a este proceso la auto-organización.

Hay cuatro roles para los participantes de un proyecto que se administre con Scrum:

- Dueño del producto. Es quien determina las prioridades; debe ser una sola persona.
- Dueño del Scrum. Administra y facilita la ejecución del proceso.
- Equipo de trabajo. Son los encargados entregar resultados al final del *sprint*. Por *sprint* se entiende unidad de desarrollo de Scrum y dura de 1 a 4 semanas como máximo.
- Interesados. Asesoran y observan el proceso; tienen interés en el resultado final.

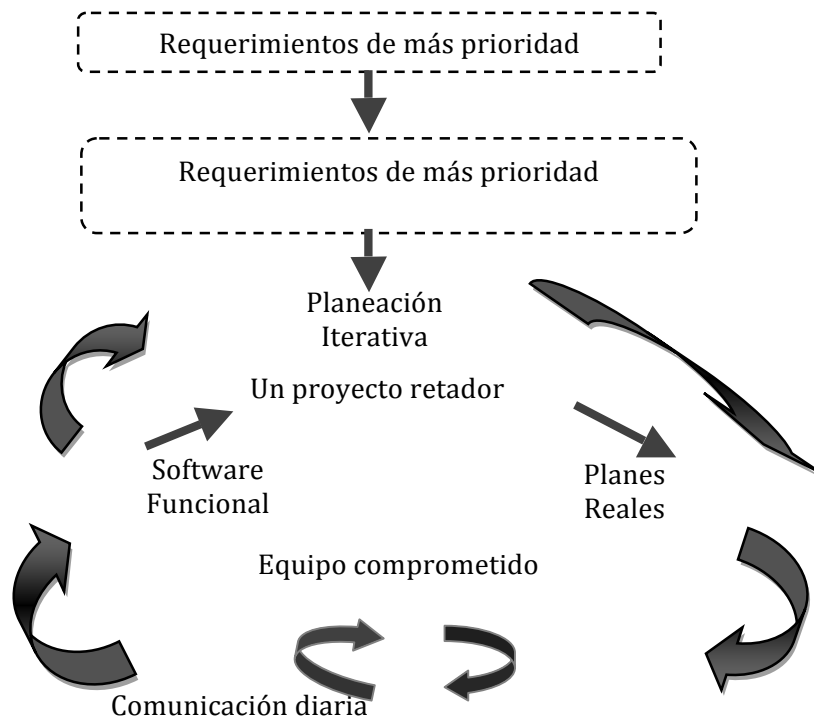


Figura 2.1. Proceso general de la metodología XP [Rosenberg, 2003].

Scrum es iterativo por naturaleza; cada iteración, o *sprint*, dura un mes calendario. Se inicia con una reunión entre el equipo de trabajo, el dueño del scrum (*scrum-master*) y los patrocinadores del proyecto. En esta reunión, los patrocinadores exponen y aclaran los requerimientos del proyecto, y también se encargan de priorizarlos. Este es el alcance acumulado del producto (*product backlog*). Posteriormente el equipo de trabajo decide que funcionalidad puede entregar al final del *sprint* – el alcance acumulado del *sprint* (*sprint backlog*)

Es de gran importancia que los patrocinadores del proyecto respeten el alcance, y no alteren o interfieran con el equipo de trabajo durante el *sprint*. También es indispensable no violar el principio ágil de trabajar únicamente 40 horas a la semana. Después de esta reunión, el equipo es libre de organizarse como considere conveniente. Cada día el equipo de trabajo se reúne con el dueño(a) del Scrum en la “junta Scrum,” la cual debe ser de no más de 15 minutos al día. Si no se puede mantener en esos niveles, significa que hay muchas personas y es necesario identificar quien debe participar en esa reunión. En esta reunión se le toma el

pulso diario al proyecto; el dueño(a) del Scrum detecta que problemas existen y busca eliminarlos, fungiendo como un facilitador(a) más que como un líder de proyecto. El objetivo principal de la junta Scrum es que los participantes hablen entre ellos y descubran que necesidades tienen, las resuelvan en el momento, y si no es posible, buscar los foros adecuados posteriormente. Es esta reunión se hacen estimaciones del trabajo restante de cada participante. Estas estimaciones se recogen y tabulan en las gráficas de consumo. Estas gráficas deben mostrar que el trabajo pendiente para entregar la funcionalidad acordada debe disminuir con el paso del tiempo, de otra manera, lo que demuestran es que el equipo fue demasiado optimista y ahora se está enfrentando con la realidad. El *Scrum master* puede tomar la decisión de acordar una disminución del alcance con el dueño del producto o informarle con tiempo que no se va a lograr la meta y que es realmente factible de alcanzarse. En el caso remoto de que los resultados sean de verdad terribles, es posible cancelar todo el *sprint*.

Cuando se planean las actividades del Scrum debe considerarse que los productos resultantes deben estar listos para consumo del usuario, es decir, no solo se debe contar con el sistema de software, también debe estar lista la documentación, el instalador, y el manual del usuario.

Al final del *sprint* se hace una presentación con el dueño del producto. En esta reunión se evalúa el resultado del *sprint*, y se revisa el alcance acumulado del producto, y con base al producto que el usuario final tiene ahora, se revaloran los requerimientos para escoger un nuevo alcance para el próximo *sprint*. Es importante que el producto final del *sprint* tenga la calidad necesaria para que el cliente pueda usarlo por sí mismo. Esto le permitirá tener un mejor entendimiento sobre a donde quiere dirigir el desarrollo de su producto y ver un progreso real de su proyecto, en un corto tiempo. Idealmente, la reunión de presentación al final del *sprint* debiera fungir como la junta de planeación del próximo *sprint*. *La figura 2.2 muestra el proceso general de Scrum.*

- Es fácil de entender e implementar.
- Está enfocado a los procesos y su mejoramiento para el alcance de objetivos organizaciones y no sólo como un marco de certificación.
- Ayuda a alcanzar los requisitos con otros modelos como el ISO 9000:2000, CMM y CMMI.
- Está pensado para las organizaciones mexicanas desarrolladoras de software.
- La estructura del modelo es adaptable debido a que sus procesos están divididos en tres principales categorías que van de acuerdo a las estructuras de cualquier organización.
- Cuenta con seis procesos principales y de tres subprocesos.
- Facilita la integración de la gestión de recursos humanos, de infraestructura y conocimiento.

De acuerdo al Nyce [nyce, 2013], Moprosoft fue elevado al rango de Norma Mexicana (NMX-I-059-NYCE-2011) y cuya primera versión fue publicada en 2006. Desde entonces ha sido implementada en empresas mexicanas micro, pequeñas y medianas. Este modelo ha sido adoptado por la organización internacional de estándares (ISO) quien lo nombró estándar 29110 – perfiles del ciclo de vida de software para empresas muy pequeñas (*very small entities*)

2.2.2. Descripción.

De acuerdo a la norma Mexicana NMX-I-059/02-NYCE-2005 los procesos para la creación de software se agrupan en 3 categorías: Alta Dirección, Gestión y Nivel Operativo, como se muestra en la figura 2.3.



Figura 2.3. Diagrama de categorías de procesos [Oktaba, 2005]

A continuación se listan los procesos que define Moprosoft para cada categoría:

Categoría de Alta Dirección: Está integrada por el proceso de:

- **Gestión de Negocio:** En el cual se establece la razón de ser de la organización, así como objetivos y las acciones necesarias para alcanzarlos- Se deben tomar en cuenta las necesidades de los clientes y los resultado de evaluaciones para poder determinar los cambios que permitan realizar una mejora continua. Esto ayuda a la organización a responder ante los cambios del medio ambiente y orientar a sus trabajadores de acuerdo a los objetivos previamente establecidos.

Categoría de Gestión: Está integrada por los siguientes procesos:

- **Gestión de Procesos:** es el encargado de realizar el establecimiento de los procesos de la organización de acuerdo a los procesos que se requieren de acuerdo a lo identificado en el plan estratégico, también se planifican e implementan las actividades de mejora de los mismos.
- **Gestión de Proyectos:** verifica que los proyectos sean los adecuados y sumen a los objetivos y las estrategias de la organización.
- **Gestión de Recursos:** se encarga de buscar y adquirir para la organización los recursos que necesita tales como: infraestructura, proveedores y recursos humanos. Asimismo, este proceso se encarga de crear una base del conocimiento de la organización con el fin de alcanzar los objetivos del plan estratégico. A su vez, este proceso se encuentra apoyado por tres subprocesos que a continuación se describen:
 - ✓ **Recursos Humanos y Ambiente de Trabajo:** tiene como principal propósito suministrar a la organización de recursos humanos adecuados para el cumplimiento de las responsabilidades establecidas en la organización. Asimismo, también evalúa el ambiente de trabajo.
 - ✓ **Bienes, Servicios e Infraestructura:** Proporciona proveedores de bienes, servicios e infraestructura que ayuden al cumplimiento de los requisitos de adquisición de procesos y proyectos.
 - ✓ **Conocimiento de la organización:** Tiene el objetivo de mantener y tener a disposición la base del conocimiento que está integrado por información y productos generados por la organización.

Categoría de Operación: está integrada por los siguientes procesos:

- **Administración de Proyectos Específicos:** En este proceso se realiza el establecimiento y lleva a cabo de forma sistemática las actividades que permiten cumplir con los objetivos de los proyectos en tiempo y costos planeados.
- **Desarrollo y Mantenimiento de Software:** Proceso en el cual se lleva a cabo sistemáticamente actividades tales como de diseño, construcción, integración y pruebas de productos de software nuevos o modificados de acuerdo a los requerimientos.

2.3. Desarrollo dirigido por pruebas

2.3.1. Definición.

El desarrollo dirigido por pruebas tiene sus orígenes en *Java* y *Smalltalk* [Newkirk, 2004]. Originalmente fue un marco de trabajo (framework) con el nombre de *SUnit* (*Smalltalk Unit*), seguido por el *framework JUnit* (*Java Unit*). Desde entonces, el término '*xUnit framework*' ha sido usado para representar las diferentes versiones de lenguajes del mismo *framework*. El *framework xUnit* ha sido implementado en diferentes plataformas y lenguajes.

2.3.2. Descripción.

El desarrollo dirigido por pruebas (*Test Driven Development* o TDD) es una de las 12 prácticas claves de la Programación Extrema. En TDD los desarrolladores de software "prueban primero, después codifican," enfocándose primero en la verificación y validación de los requerimientos de software mediante la construcción de pruebas unitarias automatizadas. La figura 2.4 muestra el proceso de desarrollo dirigido por pruebas. Algunos autores opinan que el desarrollo dirigido por pruebas es un nuevo enfoque que ofrece el potencial para mejorar significativamente el estado de construcción de software [Jones, 2004].

En este paradigma, el diseño evoluciona como un nuevo código que es escrito para satisfacer la pruebas que fallaron [Jones, 2004]. Debe hacerse notar que el desarrollo dirigido por pruebas es un método de desarrollo, no de pruebas, cuyo objetivo es: crear código de alta calidad que funciona correctamente, a través de la siguiente secuencia de pasos:

- 1) Escribir las pruebas unitarias antes que el código. Se entiende por pruebas unitarias aquellas que prueban una única funcionalidad, se realizan por y para los desarrolladores a nivel de clase/método, ayudan a resolver defectos y se ejecutan muchas veces.
- 2) Generar el código más simple que pase la prueba y
- 3) Refactorizar de manera incremental. Éste proceso consiste en mejorar el código existente, elevar la flexibilidad y hacer el producto tolerante al cambio.

En los desarrollos de software el código resulta ser el entregable con mayor valor, por lo cual la calidad del código se vuelve importante, éste debe tener características tales como ser fácil de probar, mantener, ser limpio y debe tener la finalidad de cumplir con un requerimiento

funcional. Keith define que hay dos tipos de entregables de gran peso en la finalización de proyectos de TDD, la primera es la colección de pruebas englobando tanto entradas y salidas esperadas; la segunda es un código que funciona con la capacidad de superar las pruebas [Keith, 2004].

2.3.3. Ventajas actuales.

Dentro de los principales beneficios que aporta TDD están los siguientes [Jones, 2004]:

- 1) Como el código se construye en base a las pruebas unitarias, no existe código sin pruebas asociadas. El código es sólido, las pruebas perduran y pueden ser usadas como documentación, hay entregas más rápidas, menos depuración, menos errores, más confianza del cliente, mayor productividad y mejor diseño.
- 2) En experimentos realizados en la industria usando el desarrollo dirigido por pruebas (TDD), se han reportado incrementos significativos en la calidad del código comparado con los enfoques tradicionales de desarrollo [Jones, 2004]. Los proponentes del desarrollo dirigido por pruebas (TDD) aseguran que la densidad de defectos del software comercial puede ser reducido desde un 18% a un 50 % cuando las pruebas son escritas al inicio, antes del fin del ciclo de desarrollo [Jones, 2004].

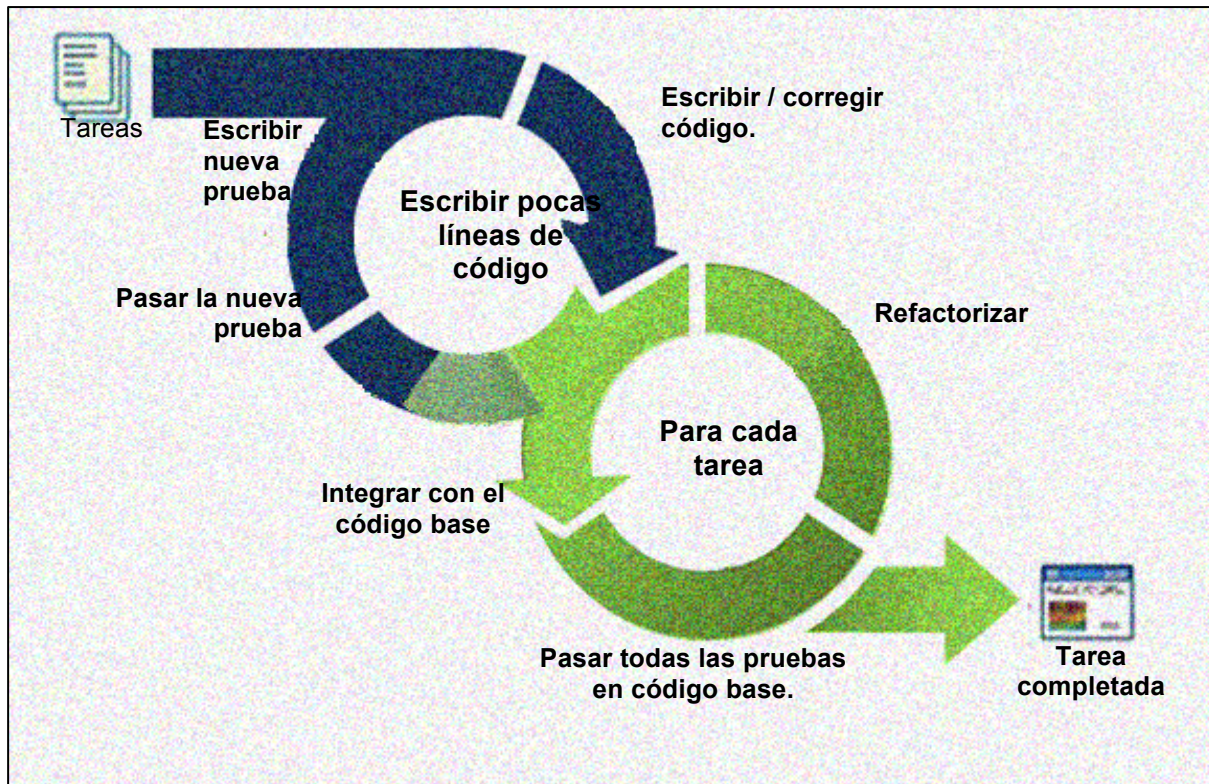


Figura 2.4. Modelo de procesos del Desarrollo Dirigido por pruebas [Erdogmus, 2005].

2.4. Modelo MVC (*Model View Controller*)

El modelo propuesto en esta investigación está enfocado a proyectos web y con arquitecturas basadas en el modelo MVC, por lo que se da una introducción de este patrón esta sección.

2.4.1 Definición.

Según Buchmann, un patrón de arquitectura de software describe un problema particular y recurrente del diseño, que surge en un contexto específico, y presenta un esquema genérico y probado de su solución

MVC es un patrón de arquitectura de software, mostrado en la figura 2.4, que es usado para separar una aplicación en tres principales aspectos [Reenskaug, 2003]:

- El Modelo (lógica de negocio): conjunto de clases que describen a los datos con los que trabaja la aplicación así como las reglas de negocio acerca de cómo los datos pueden ser cambiados y manejados.
- La Vista (interfaz de usuario): la aplicación de la interfaz de usuario.
- El Controlador (lógica de control): un conjunto de clases que maneja la comunicación desde el usuario, el flujo de la aplicación y lógica específica de la aplicación.

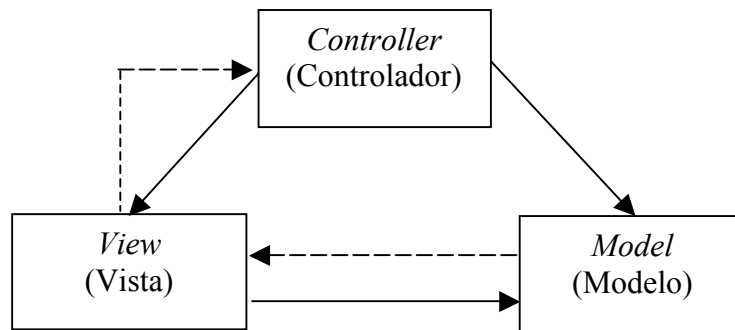


Figura 2.5. Diagrama de Modelo MVC (Model View Controller) [Leff, 2001].

2.4.2 Descripción

En el verano de 1978, el Dr. Trygve Reenskaug invirtió un año en Palo Alto, California trabajando en la introducción de Smalltalk-80, uno de los primeros lenguajes orientados a objetos. Con Smalltalk, Dr. Reenskaug refinó sus ideas y las sintetizó en el patrón MVC. Luego en el año de 1979 fue llamado *Thing-Model-View-Editor* y más tarde su nombre fue simplificado a *Model-View-Controller*.

El MVC es un patrón de arquitectura de software que ha sido muy importante a lo largo del tiempo, donde se separa la lógica del negocio (modelo) y la interfaz del usuario (vista) dando como resultado un código fácil de mantener. El controlador sirve para ocultar los detalles del protocolo que se utilizó en el modelo y la vista. Por otro lado el modelo es el encargado de extraer la lógica de información, la cual convierte a la vista y a la acción en independientes, resultando una poderosa y elegante propuesta de separación de código que se utiliza eficazmente en las aplicaciones web. La separación de código del patrón MVC solo agrega un poco de complejidad al diseño de la aplicación, comparado el gran beneficio que proporciona. MVC ha sido usado por docenas de marcos de desarrollo desde su introducción; se puede encontrar MVC en Java y C++ en ambientes de Mac, Windows y

Linux. Algo valioso para la investigación es que las implementaciones actuales del modelo MVC tienen una estructura para documentar pruebas unitarias.

2.4.3 Herramientas actuales que usan MVC

La tabla 2.1 muestra algunos de los entornos de desarrollo que actualmente incorporan el framework MVC.

Tabla 2.1. Herramientas actuales que usan el patrón MVC [Avraham, 2001].

Nombre de la Herramienta	Lenguaje
Django [Holovaty, 2005]	Python
Spring [Walls, 2010] Struts [Husted, 2004]	Java
Zend Framework [Zend Technologies, 2012]	PHP
MonoRail [James, 2010]	C#
Slim3 [google, 2013]	Java
ASP .NET MVC [Sanderson, 2009]	Visual Basic, C#

El MVC es utilizado comúnmente en aplicaciones web donde es más fácil identificar cada uno de elementos que lo componen. En esas aplicaciones la vista está representada por la página HTML, mientras que el controlador es el código que se encarga de generar el contenido de la página. El modelo se encuentra representado en el contenido actual que habitualmente está almacenado en una base de datos.

Los *frameworks* actuales como *Junit* o *Nunit* facilitan el uso de las herramientas de desarrollo dirigido por pruebas, ya que fueron diseñados con el enfoque TDD. Algunas implementaciones de MVC como ASP .NET y MVC, con sus principios de separación de código, usan TDD para realizar pruebas unitarias sobre las clases de la capa de Model.

Trabajos Relacionados

Existen un gran número de trabajos recientes que han intentado atacar el problema descrito en esta investigación. Enseguida se describen algunos que consideramos relevantes.

En un grupo de desarrollo de software de la *IBM Retail Store Solutions*, se construyó un software de sistemas usando el enfoque de *Test Driven Development* (TDD) [Maximilien, 2003]. El equipo estaba compuesto por 9 ingenieros de tiempo completo, cinco de los cuales localizados en Raleigh NC, Estados Unidos, (incluyendo al líder del equipo) y cuatro en Guadalajara, México. Adicionalmente se usaron recursos humanos de tiempo parcial para la administración y desarrollo del proyecto. Aunque el equipo de desarrollo tenía amplia experiencia con la especificación *JavaPOS 2* usada en sus proyectos y con dispositivos *POS* (*hardware* para punto de venta), se notó que en cada revisión de entregables la densidad de defectos después de las pruebas de verificación funcional (FVT) no se había reducido como se esperaba. Como resultado de esto, el equipo de desarrollo estuvo abierto a nuevos enfoques de desarrollo como el TDD. Usando TDD, se logró reducir la densidad de defectos en alrededor del 50% comparado a un sistema similar construido usando un enfoque de pruebas unitarias a la medida. El proyecto fue completado a tiempo y con mínimo impacto en la productividad. Adicionalmente, el conjunto de casos de prueba automatizados creados vía TDD fue usado como un activo reusable para mejorar la calidad durante el tiempo de vida del sistema. Como resultado, de acuerdo a IBM, se tuvo una relativa mejora de la calidad comparada con proyectos anteriores que usaron métodos tradicionales, así como una baja en la densidad de defectos por debajo del estándar de la industria. De acuerdo a estos resultados se observa que TDD puede ayudar en la generación de productos de alta calidad, aunque esta calidad implica una pérdida moderada de productividad y aumento en el tiempo de desarrollo.

Microsoft realizó una investigación acerca de la utilidad del TDD desde el punto de vista de la calidad del software y de la productividad, en términos del tiempo de desarrollo. Este caso de estudio se desarrolló con profesionales del software en equipos de desarrollo (5 a 8 ingenieros de software por equipo) de *Windows* y *MSN* con diferentes objetivos

organizacionales, plataformas y ambientes [Thirumalesh, 2006], Al igual que en el proyecto de IBM mencionado anteriormente, en este proyecto Microsoft construyó un activo de conocimientos empíricos sobre la eficacia de TDD, para ser replicado dentro de la Compañía. De acuerdo a Microsoft, se observó un incremento significativo de la calidad del código en proyectos usando TDD, comparados con proyectos similares de la misma organización que no usaron este enfoque. El proyecto tomó 15% más del tiempo planeado como consecuencia de escribir pruebas unitarias; posteriormente estas pruebas fueron usadas como documentación del proyecto para mantenimientos futuros.

Ahmad [Ahmad, 2010] realizó experimentos en la Universidad Integral de Lucknow, India, sobre el efecto de la Integración continua (CI) en Desarrollo dirigido por pruebas (TDD). De acuerdo al autor cuando un sistema está compuesto por cientos de componentes los cambios en cualquiera de ellos impactan activamente sobre los otros. TDD es un enfoque donde el proyecto se desarrolla de forma incremental, donde se generan primero las pruebas y después el código indispensable para cumplir con las pruebas. La integración continua es el proceso para recibir retroalimentación rápida constante del cliente y asegurar que el producto cumple con las expectativas. De acuerdo a Ahmad *TDD* y *CI* han cambiado la forma en que el software es probado. Frecuentemente la etapa de pruebas es un proceso separado al final del proyecto pero con *TDD* y *CI* es practicado en todo el proyecto a través de las pruebas unitarias. El autor realizó experimentos en 2 proyectos y el estudio proporciono evidencias sustanciales de que TDD es una herramienta efectiva para mejorar la calidad del código fuente. De acuerdo al autor con *CI* el programador se enfrenta constantemente a preguntas de los usuarios no solo a nivel funcional sino también de interfaz de usuario lo que les permite absorber los cambios de forma continua a lo largo de todo el proyecto. De acuerdo al autor la siguiente etapa del proyecto es usar este estudio para construir un *framework* de desarrollo de software basado en *TDD* y *CI*.

Kahan, en la Universidad de Gandhara, Pakistán [Kahan, 2010] estudió la reducción del esfuerzo de las pruebas usando TDD. De acuerdo a dicho autor, TDD es un proceso de

desarrollo de software basado en la repetición de pequeños ciclos de desarrollo. En cada ciclo de desarrollo se generan pruebas unitarias y el código asociado a esa prueba. Además, Kahan afirma que un ingeniero(a) que usa *TDD* de forma pura, raramente necesita usar la técnica de depuración de código (*debugger*). Cuando se usa en conjunción con sistema de control de versiones, cuando una prueba falla de forma inesperada, revertir el código a la última versión pasada es más eficiente y productivo que usar un depurador.

Segun Kahan, TDD ofrece la habilidad de avanzar pequeños pasos cuando se requiere. Esto permite a los programadores enfocarse en la tarea actual y en su objetivo, que es hacer que el código pase la prueba. Inicialmente no se consideran los casos excepcionales en el manejo de errores; las pruebas para crear esas circunstancias excepcionales son implementadas separadamente. De esta forma, *TDD* asegura que todo el código escrito está cubierto en al menos una prueba. Esto da al equipo de programación y usuarios subsecuentes un nivel de confianza más alto en el código. Kahan demuestra con sus resultados, que usando *TDD* se produce más código que sin *TDD*, porque el código de las pruebas unitarias incrementa el tamaño, pero el tiempo de implementación del código es típicamente menor; esto es debido a que un gran número de pruebas ayudan a limitar el número de defectos en el código. Asimismo, los resultados establecen que, debido a las pruebas se realicen en las primeras etapas del proyecto y se realicen frecuentemente, se facilita la detección de defectos en las primeras etapas del ciclo de desarrollo, previniendo problemas más complejos o más caros. Los resultados también muestran que *TDD* es más que una simple validación de correctez de código; también ayuda en el diseño del programa, pues al enfocarse en los casos de prueba primero, el/la ingeniero(a) debe imaginar cómo será la funcionalidad usada por los clientes, así que el programador está consciente de las interfaces antes de la implementación. El cliente tiene la oportunidad de probar estas funcionalidades completas tanto en interfaz como en código, desde etapas tempranas del desarrollo y en iteraciones incrementales.

2.4.4. El patrón *Arranging, Act Assert*

En el modelo DEPYM las pruebas unitarias de la etapa Generación de pruebas unitarias, definidas en la sección 2.6, usan el patrón *Arranging, Act Assert* definido por Willian C.

[Wake, 2004]. Esto permite definir pruebas fáciles de leer, de seguir, de entender y de mantener. Por otra parte, este patrón permite ordenar y formatear el código en los métodos de las pruebas unitarias. Según establece el patrón, cada método debería agruparse en 3 secciones funcionales separadas por líneas en blanco, que son las siguientes:

Arrange.- Todas las precondiciones y datos de entrada

Act.- el objeto o metodo bajo la pruebas

Assert.- el resultado esperado que ocurra

La figura 2.6 muestra un ejemplo del patrón en un método de una prueba unitaria.

```
[Test]
public void add_two_numbers ()
{
    // Arrange
    int firstNumber = 1;
    int secondNumber = 2;
    Calculator calculator = new Calculator();

    // Act
    var result = calculator.Add(firstNumber, secondNumber);

    //Assert
    result.ShouldEqual(3);
}
```

Figura 2.6. Ejemplo del uso del patrón *Arrange-Act-Assert* [arrange-act-assert, 2013]

3. Modelo DEPYM

En este capítulo se desarrolla el *script* general que rige el ciclo de vida de software del modelo DEPYM. Asimismo, se analiza el patrón de procesos a usar, así como cada una de las etapas que conforman cada entregable: planeación, requerimientos, diseño, generación de pruebas, codificación, pruebas de integración y cierre. También se analiza el *script* de procesos para llevar a cabo las reuniones.

3.1 Introducción

Como se nombró anteriormente, en esta investigación se propone una nueva metodología de desarrollo, llamada “Desarrollo Dirigido por Pruebas y Mejora de procesos para PyMEs.” La metodología propuesta las ventajas del framework TDD definido por Kent Beck [Beck, 2002] y se ajusta a las especificaciones de los métodos ágiles en su proceso de “Desarrollo del producto” y “Gestión del proyecto.” El modelo DEPYM incluye *scripts*, formas y estándares. Los roles que propone el modelo DEPYM para el proceso de desarrollo de software son: Líder de proyecto, analista, arquitecto de software, ingeniero de software y tester.

3.2 Descripción General

DEPYM define un documento maestro denominado *Project charter* cuya misión es servir como documento de visión y alcance. Aquí se puede definir información general sobre el proyecto tal como el nombre, descripción, estimación de fechas de arranque y terminación, personas involucradas (*stakeholders*), entregables por etapa (*releases*) en los que se dividirá el proyecto y las historias de usuario que compondrán cada entregable. Además, este documento permite definir cuál es el estándar de codificación que se usará para construir los programas. De igual forma aquí se define cuál será el documento de configuración de software que ayudará al nombrado de carpetas, archivos, formatos de fecha y horas, entre otros. La figura 3.1.muestra el diagrama general de procesos del modelo DEPYM.

El *Project charter* permite determinar el estado actual del proyecto a través del valor ganado obtenido por cada una de las historias de usuario. Las historias de usuario permiten definir de forma modular cada uno de los requerimientos funcionales del proyecto. Las historias determinan a detalle cada una de las tareas necesarias para llevar a cabo una historia, permitiendo además requerimientos específicos e información adicional sobre los procesos. El modelo DEPYM está totalmente orientado al desarrollo dirigido por pruebas, por tanto para construir el código de una historia de usuario, se definen sus tareas y de cada tarea se definen de forma detallada cada una de las pruebas unitarias que se deberán cumplir para terminar esa tarea. Para la definición de las pruebas de usuario se usa el patrón *Arranging-Act-Assert* definido en la sección 2.4.4. La figura 3.1 muestra el diagrama en bloques del proceso de desarrollo de DEPYM.

Los procesos definidos en DEPYM abarcan las siguientes áreas de procesos:

1. Procesos para realizar la definición de las historias de usuario, las cuales representan requerimientos funcionales o necesidades específicas del cliente. Asimismo, se incluye una definición de prioridades, asignación de responsables, cálculo del valor ganado y estimación de tiempos.
2. Procesos para detallar las historias de usuario con sus respectivas tareas, prioridades, estimación de tiempo en horas y valor planeado para cada tarea.
3. Procesos para la generación de las pruebas unitarias. En este paso se utiliza el patrón *Arranging-Act-Assert* [Beck, 2002].
4. Procesos para la generación de código a partir de las pruebas estableciendo las mejores prácticas del enfoque TDD.
5. Procesos para las pruebas de verificación funcional y de integración.

Las características principales de DEPYM que lo hacen tener un enfoque de procesos ágil son:

- Es un proceso iterativo e incremental
- Define entregas cortas y continuas
- Hay una alta participación del cliente en revisiones además de retroalimentación continua
- Existe una integración continua de funcionalidades al proyecto en cada entregable

- Considera un alto valor al recurso humano

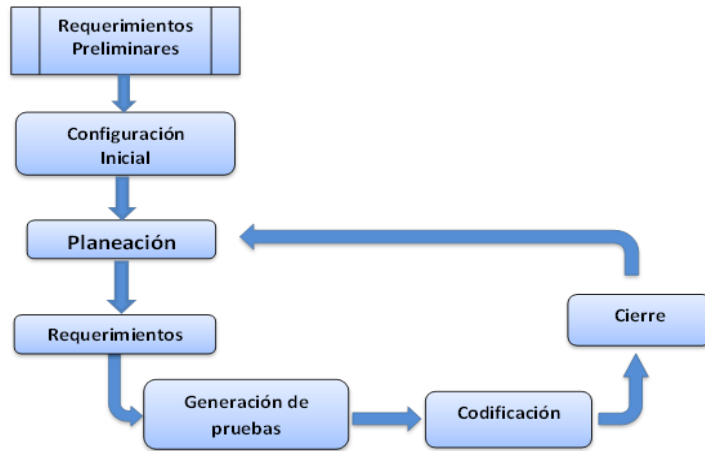


Figura 3.1. Diagrama general del modelo DEPYM.

La figura 3.2 muestra el diagrama en bloques del modelo DEPYM

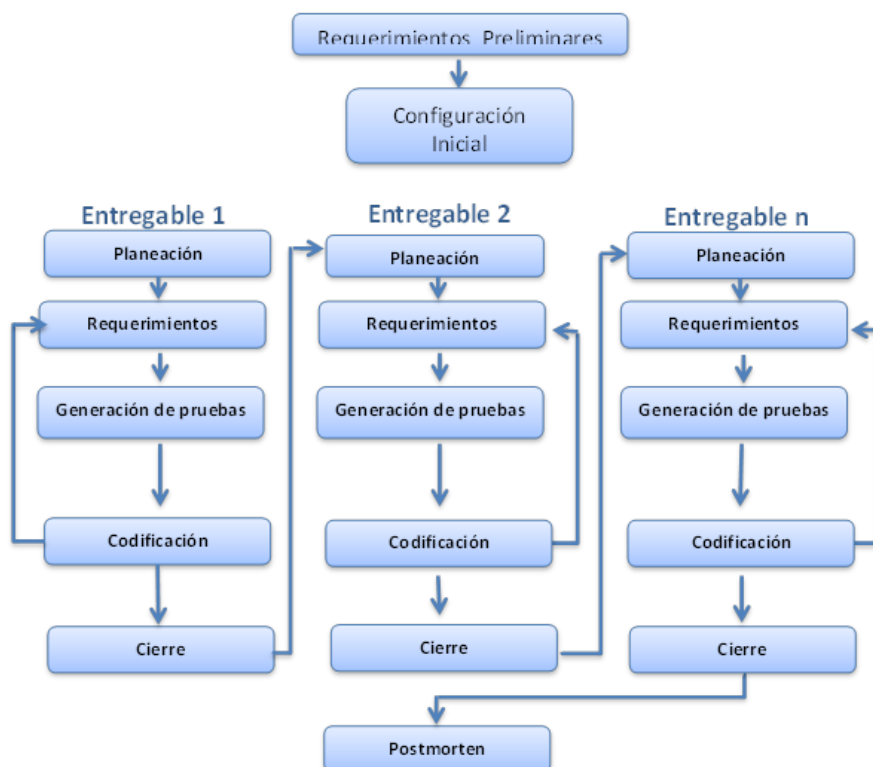


Figura 3.2. Diagrama en bloques del proceso de desarrollo del modelo DEPYM.

Formas Propuestas

El modelo DEPYM propone formas ágiles para la planeación, seguimiento y control del proyecto; la Tabla 3.1 muestra las 11 formas usadas en los diferentes *scripts* de procesos propuestos por el modelo. Cabe mencionar que en la memoria técnica completa del modelo (ver apéndice A), existen scripts para auxiliar en el llenado de dichas formas. En el experimento de validación de esta investigación se usó una hoja de cálculo para crear cada una de las formas y así facilitar tu manejo.

Tabla 3.1. Formas propuestas por el modelo DEPYM.

Id Forma	Nombre	Breve descripción
A.1.	Definición breve del proyecto	Esta forma deberá contener: breve descripción del proyecto, datos generales del cliente, alcances del proyecto, tiempo empleado en las reuniones diarias, número de semanas que tendrá un entregable, funcionalidades básicas del proyecto.
A.2.	Definición de requerimientos no funcionales	Breve descripción de requerimientos no funcionales como desempeño, velocidad, sistema operativo, versión.
A.3.	Definición de arquitectura	Definir la arquitectura a usar dentro del proyecto, n capas, MVC, SOA, entre otros.
A.4.	Entregables del proyecto	Fecha de compromiso para el entregable, definición breve del entregable
A.5.	Integrantes del equipo	Nombre del integrante, rol(es) que desempeñará de forma preponderante
A.6.	Planeación de historias de usuario (<i>Project Charter</i>)	Nombre de la historia de usuario, prioridad de 1 a 3, numero de <i>release</i> en que se asignará esta historia de acuerdo a su prioridad, desarrollador(a) responsable de esta historia, número de días en que el desarrollador estima se realizará esta historia, fecha inicio de la historia, fecha final de la historia, valor porcentual planeado en base al 100% que esta historia aporta al proyecto, puntos de valor ganados que esta historia aportará una vez terminada.
A.7.	Historia de usuario	Historia de usuario ID, nombre, descripción, requerimientos funcionales, notas adicionales, lista de tareas que deben realizarse para completar la historia (Id, nombre, prioridad, fecha inicio, tiempo en horas, valor planeado y valor ganado)
A.8	Definición de pruebas unitarias para cada Tarea	Lista de pruebas unitarias requeridas para completar la tarea actual (id tarea, id prueba, nombre, descripción, definición de la prueba unitaria de acuerdo al patrón <i>Arranging, Act Assert</i>)
A.9.	Plan de Riesgos	Descripción, impacto, probabilidad de que ocurra, estrategia para mitigarlo en caso que ocurra, fecha de levantamiento, realizado por, fecha de revisión
A.10.	Plan de comunicación	Información, medio, forma de comunicación, quien lo genera, quien lo recibe
A.11.	Mejora de procesos	Propósitos de mejora de procesos.

Estándares propuestos

El modelo DEPYM, al igual que otras metodologías Agiles, propone documentos que sirvan como estándar en algunos procesos o actividades, DEPYM propone solo 2 estándares: estándar de codificación, a ser usado en la etapa de codificación de código, y estándar de “Administración de Configuración del Software” usado en todo el proyecto. Al igual que en el caso de las formas, existe un *script* para auxiliar en la creación de estos artefactos. La tabla 3.2 muestra los estándares propuestos por DEPYM.

Tabla 3.2.- Estándares propuestos por el modelo DEPYM.

ID	Nombre	Descripción
B.1.	Estándar de codificación	Convención para definición de comentarios, declaración de clases e interfaces, indentación, longitud de líneas de código, convención de nombrado de clases, interfaces, variables, métodos.
B.2.	Administración de configuración del software	Acrónimos, estructura de carpetas, formatos de fecha y hora, nombrado de archivos y carpetas, manejo de versiones

3.3 Patrón de proceso.

Para la definición de los *scripts* del modelo DEPYM se definió un patrón de procesos, que se debe cumplir siempre. El patrón cuenta con los siguientes elementos:

- Nombre del proceso
- Propósito del proceso
- Responsables del proceso
- Diagrama del proceso
- Criterios o Requisitos de entrada
- Tareas
- Criterios o Requisitos de salida

Dentro del modelo DEPYM se definen criterios (requisitos que permiten determinar si podemos ingresar a un proceso o bien si el proceso ha sido terminado de forma satisfactoria) y documentos; para efectos de sintaxis en el llenado de las formas propuesta por el modelo; los criterios se pondrán en cursivas y los documentos en negritas. La Tabla 3.3 muestra la estructura del *script* de procesos que propone el modelo DEPYM.

Tabla 3.3. Estructura del patrón de procesos del Modelo DEPYM.

Nombre del proceso		
Propósito del proceso		
Responsable del proceso		
Diagrama del proceso		
Criterios de entrada	• <i>Criterio de entrada 1</i> • <i>Criterio de entrada 2</i> • Criterio de entrada n • Documento 1 • Documento 2 • Documento n	
Paso	Actividad	Descripción
1	Actividad 1	Descripción breve 1
2	Actividad 2	Descripción breve 2
n	Actividad n	Descripción breve n
Criterios de salida	• <i>Criterio de entrada 1</i> • <i>Criterio de entrada 2</i> • Criterio de entrada n • Documento 1 • Documento 2	

	Documento n
--	-------------

3.4 **Script general del proceso DEPYM.**

El modelo DEPYM está compuesto por 7 procesos. El Proceso general DEPYM está dividido en las etapas: configuración inicial, Planeación Ágil, Requerimientos, Generación de pruebas unitarias, Codificación, Pruebas de integración y Cierre.

La tabla 3.4.muestra el *script* general de procesos del modelo DEPYM, definiendo de manera general el *script* que se debe llevar a cabo, el llenado de formas y el uso de estándares que se usarán dentro del proyecto.

Dentro del proceso general del modelo DEPYM se propone una etapa inicial de configuración del proyecto, donde se definen los alcances del proyecto, arquitectura, equipo y los entregables. Para cada entregable se aplica una serie de etapas de forma iterativa, donde se realiza una planeación ágil y requisitos mínimos, para posteriormente aplicar la técnica de Desarrollo dirigido por pruebas. Además aquí se definen las pruebas previas a la codificación y que el proyecto deberá cumplir para ser aceptado. La siguiente etapa es la codificación, que se genera para cumplir única y exclusivamente con esas pruebas. Posteriormente se aplica una técnica de mejora del código llamada *refactoring*; finalmente se realiza la integración del código generado al ya existente. Este proceso es iterativo para cada uno de los requisitos de este entregable. Las formas, a las que hace referencia la tabla, se encuentran en el apéndice A.

Tabla 3.4. Script General de procesos de DEPYM.

Nombre del proceso: Script General de DEPYM		
Propósito del proceso: Guiar al líder del proyecto y al equipo en todo el proceso de desarrollo del proyecto		
Responsable del proceso: Líder del proyecto		
Capítulo 2 Diagrama de proceso		
<pre> graph TD RP[Requerimientos Preliminares] --> CI[Configuración Inicial] CI --> E1_P[Planeación] CI --> E2_P[Planeación] CI --> En_P[Planeación] E1_P --> E1_R[Requerimientos] E2_P --> E2_R[Requerimientos] En_P --> En_R[Requerimientos] E1_R --> E1_GP[Generación de pruebas] E2_R --> E2_GP[Generación de pruebas] En_R --> En_GP[Generación de pruebas] E1_GP --> E1_C[Codificación] E2_GP --> E2_C[Codificación] En_GP --> En_C[Codificación] E1_C --> E1_Cierre[Cierre] E2_C --> E2_Cierre[Cierre] En_C --> En_Cierre[Cierre] E1_Cierre --> PM[Postmortem] E2_Cierre --> PM En_Cierre --> PM </pre>		
Criterios de entrada		1. Proyecto autorizado y alineado a la misión de la empresa 2. Llenado de la forma de Descripción breve del proyecto (Forma A.1.- Definición breve del proyecto) 3. integrantes del equipo definidos 4. Roles asignados 5. Horarios de trabajo establecidos
Paso	Actividad	Descripción
1	Configuración inicial	1. Realizar la reunión inicial 2. Llenado de forma para la definición de arquitectura (Forma A.3. Definición de arquitectura) 3. Llenado de forma de Requerimientos no funcionales (Forma A.2.- Definición de

		requerimientos no funcionales) 4. Llenado del documento de Estándar de codificación (Forma B.1.- Estándar de codificación) 5. Llenado del documento de Administración de la configuración del software (Estándar B.2.- Administración de configuración del software) 6. <i>Determinar roles</i> 7. <i>Definir alcances del proyecto</i> 8. Llenado de la forma Equipo de trabajo (Forma A.5.- Integrantes del equipo) 9. Llenado de la forma Plan de comunicación (Forma A.10.- Plan de comunicación) 10. <i>Definir la lista de funcionalidades o requerimientos del proyecto</i> 11. <i>Determinar el número de entregables (releases) y funcionalidades (requerimientos) que cubrirá cada uno de ellos.</i> 12. Llenar la forma Entregables del proyecto (Forma A.4.- Entregables del proyecto)
2	Planeación	1. <i>Para el entregable actual realizar un proceso de Planeación ágil que permita alcanzar los tiempos y calidad requeridos.</i> 2. <i>Identificar las necesidades del cliente para convertirlas en historias de usuario</i> 3. <i>Realizar un análisis de tiempos</i> 4. <i>Valor planeado que representa cada historia</i> 5. Llenar la forma de Planeación de historias de usuario (Forma A.6.- Planeación de historias de usuario)
3	Requerimientos	1. <i>Identificar las historias de usuario</i> 2. Llenado de la forma Historia de usuario (Forma A.7. Historia de usuario) para cada historia de usuario. Además se definen cada una de las tareas necesarias para cumplir cada historia. 3. <i>Así mismo se analizan los riesgos, prioridades, esfuerzo requerido y valor planeado de cada tarea</i>
4	Generación de pruebas unitarias	1. <i>Analizar cada una de las tareas de las historias de usuario y se generan las pruebas que deberán pasarse para cumplir con los requerimientos de esa tarea.</i> 2. Llenado de la forma Pruebas unitarias (Forma A.8.- Definición de pruebas unitarias para cada tarea)
5	Codificación	1. <i>Realizar el código más simple que funcione y haga pasar las pruebas definidas en la etapa anterior de acuerdo al estándar de codificación</i> 2. <i>Refactorizar el código para mejorar la funcionalidad existente.</i> 3. <i>Probar la funcionalidad de todas las historias de usuario del entregable</i> 4. <i>Realizar el proceso de integración continua e integrar al código generado al existente del proyecto</i>

5	Cierre	1. <i>Planear el siguiente entregable</i> 2. <i>Realizar un análisis del proceso utilizado en este entregable para detectar errores y propósitos de mejora. Llenado de la forma A.11.</i> 3. <i>Si el entregable actual fuera el cierre del proyecto se pasa a la etapa de Post Mortem para realizar el proceso de mejora de todo el proyecto</i>
7	Post Mortem	1. <i>Si es el último entregable, realizar un análisis global del proyecto con el propósito de encontrar aéreas de mejora aplicables a todo el proyecto o que se hayan encontrado repetidamente en cada entregable. Actualizar la forma de propósitos de mejora Forma A.11</i>
	Criterios de salida	1. <i>Planeación del siguiente entregable</i> 2. Estándar de codificación actualizado 3. Documento de Configuración de software actualizado (SCM) 4. Definición de las historias de usuario del siguiente release 5. Plan de Riesgos actualizado para el siguiente release 6. Plan de comunicación actualizado

3.5 Etapa de Configuración inicial.

La configuración inicial del proyecto según DEPYM establece las líneas base de procesos, estándares, y lineamientos generales que se ocuparán en todos los entregables a lo largo del proyecto, se realiza una sola vez antes al inicio del proyecto y antes de iniciar el primer entregable. En esta parte se incluye: arquitectura, requerimientos no funcionales, estándar de codificación utilizado, administración de la configuración del software, plan de comunicación, entre otros.

La tabla 3.5 muestra el *script* de configuración inicial del modelo DEPYM, indicando los criterios de entrada, los actividades a realizar, formas a llenar y estándares usados.

Tabla 3.5. Script de configuración inicial de DEPYM.

Propósito del proceso: Guiar al líder del proyecto y al equipo en todo el proceso de Configuración inicial		49
Responsable del proceso: Líder del proyecto		
Diagrama del proceso		
Criterios de entrada ↓ Primera Reunión ↓ Alcances y funcionalidades del proyecto ↓ Definición de arquitectura ↓ Requerimientos no funcionales ↓ Definición del estándar de codificación ↓ Definición del estándar de Configuración ↓ Equipo y de trabajo y Roles ↓ Plan de riesgos ↓ Plan de comunicación ↓ Definición de entregables ↓ Criterios de salida		
	Criterios de entrada	1. Proyecto alineado a la misión de la empresa 2. Descripción breve del proyecto 3. Roles, integrantes del equipo, giro de la empresa, factores de comunicación, horarios, prioridad del proyecto
Paso	Actividad	Descripción
1	Primera reunión	1. Establecer el tiempo empleado en las reuniones diarias 2. Establecer los estándares que se utilizarán a la largo del proyecto 3. Determinar el número de semanas que tendrá un entregable. 4. Llenar la forma descripción del proyecto (Forma A.1.- Definición breve del proyecto) en las secciones tiempo empleado en las reuniones diarias y duración del entregable
2	Alcances y funcionalidades básicas del proyecto	1. Exponer el propósito del producto, para esto se realizará el análisis, revisión, consideración de las funcionalidades del proyecto 2. Llenar la forma descripción del proyecto (Forma A.1.- Definición breve del proyecto) en las secciones Alcance del

		proyecto y Funcionalidades básicas del proyecto.
3	Definición de arquitectura	1. Definición de arquitectura de software a ocupar dentro del proyecto (Forma A.3. Definición de arquitectura)
4	Requerimientos no funcionales	1. <i>Definir requerimientos no funcionales que deberá cumplir la aplicación tales como: desempeño, velocidad de respuesta, número de usuarios concurrentes</i> 2. Llenar la forma de requerimientos no funcionales (Forma A.2.- Definición de requerimientos no funcionales)
5	Definición del estándar de codificación	1. <i>Definir del documento de estándar de codificación de software a usar por todo el equipo de desarrollo a lo largo de todo el proyecto</i> 2. Llenar el estándar de codificación de acuerdo a la Forma B.1.- Estándar de codificación
6	Definición de la administración de la configuración del software	1. <i>Definir el estándar de configuración del software para el nombre de documentos y carpetas</i> 2. Llenar el estándar de codificación de acuerdo a la Estándar B.2.- Administración de configuración del software
7	Determinar el equipo de trabajo y roles	1. <i>Determinar a los integrantes del equipo: Líder del proyecto, Arquitecto(a) de Software, Analista, Desarrolladores y Tester.</i> 2. Llenar la forma integrantes del equipo (Forma A.5.- Integrantes del equipo)
8	Definición del plan de riesgos	1. <i>Determinar de forma ágil el documento de plan de riesgos con la información mínima indispensable.</i> 2. Llenar la forma Plan de riesgos (Forma A.9.- Plan de Riesgos)
9	Creación del plan de comunicaciones	1. <i>Crear el plan de comunicaciones para asegurar el compromiso y el flujo natural de retroalimentación del equipo con la información mínima requerida.</i> 2. Llenar la forma de plan de comunicaciones (Forma A.10. Plan de comunicación)
10	Definición de entregables	1. <i>Tomando la lista de funcionalidades de la forma Forma A.1.- Definición breve del proyecto establecer cuántos entregables se harán a lo largo del proyecto y determinar que funcionalidades cubrirá cada uno de ellos</i> 2. Llenar la forma entregables del proyecto (Forma A.4.-

		Entregables del proyecto)
	Criterios de salida	1. Arquitectura 2. Descripción breve del proyecto 3. Requerimientos no funcionales 4. Estándar de codificación 5. Administración de la configuración del software 6. Plan de comunicación 7. Plan de Riesgos 8. Lista de Entregables con sus funcionalidades

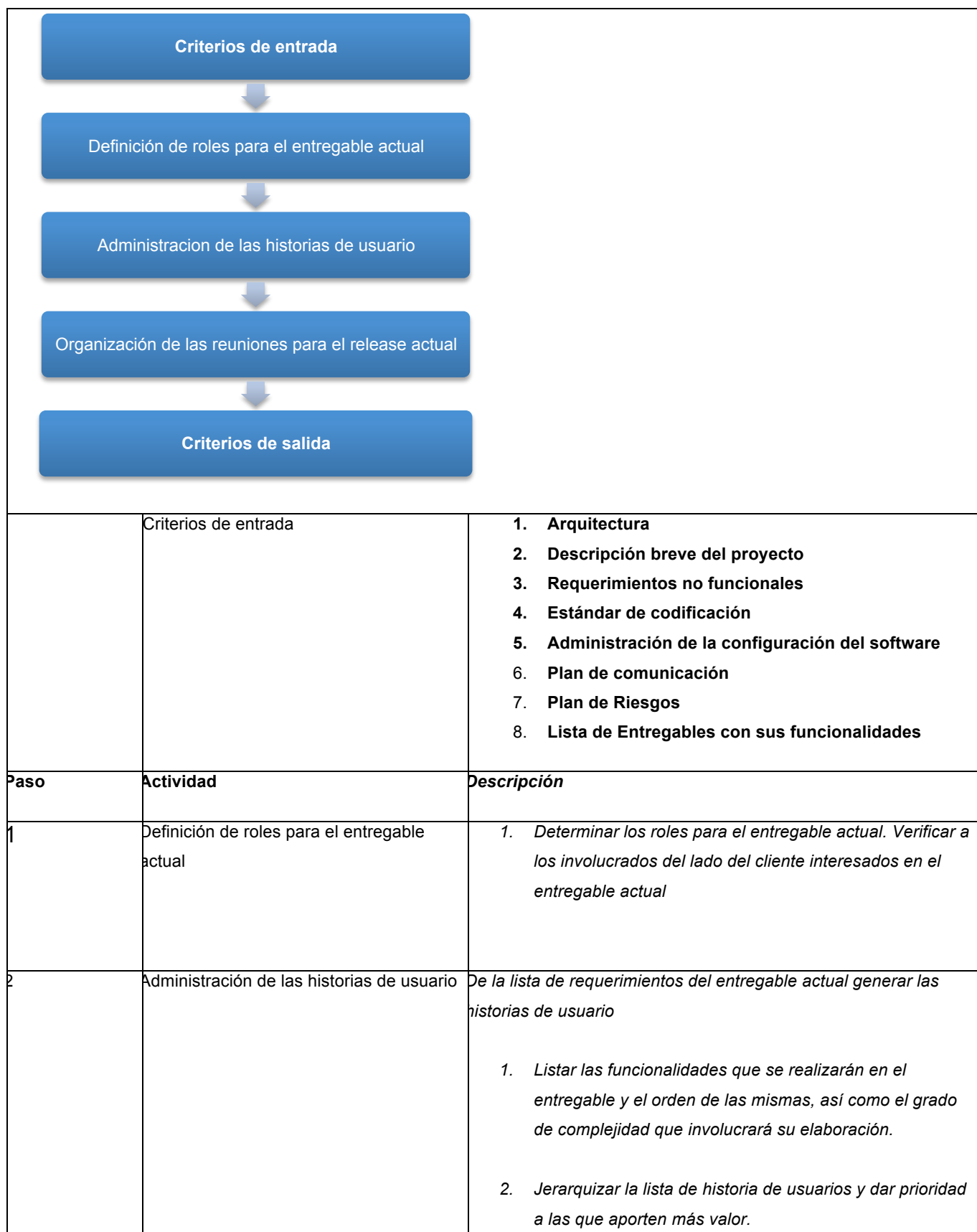
3.6 Etapa de planeación.

La etapa de planeación se realiza para cada entregable. En esta fase se establece todos los fundamentos mínimos de planeación necesarios para desarrollar el entregable actual en tiempo, costo y calidad, todo dentro de un marco de trabajo ágil. El/la cliente, como parte del equipo, debe proporcionar toda la información necesaria en esta etapa acerca de las funcionalidades que se tendrán que desarrollar en el entregable actual.

La tabla 3.6 muestra el *script* de planeación del modelo DEPYM, indicando los criterios de entrada, los actividades a realizar, formas a llenar y estándares usados.

Tabla 3.6. Script de Planeación del modelo DEPYM.

Script de Planeación de DEPYM
Propósito del proceso: Guiar al analista del proyecto y al equipo en todo el proceso de Planeación
Responsable del proceso: Analista
Capítulo 3 Diagrama del proceso



		3. <i>Determinar los involucrados en el desarrollo del software, así como definir las tareas asignadas para cada integrante</i> 4. <i>Dar las estimaciones de tiempo para cada una de las historias de usuario</i> 5. <i>Estimar el valor ganado que aportara cada historia de usuario.</i> 6. Llenar la forma de planeación de historias de usuario con la información analizada en los puntos anteriores (Forma A.6.- Planeación de historias de usuario (Project Charter))
3	Organización de las reuniones para el release actual	1. <i>Establecer horario de las reuniones diarias y tiempo, así como mecánica a seguir.</i> 2. <i>Establecer o la fecha de la reunión de revisión del entregable.</i> 3. Llenar o Actualizar la forma de entregables del proyecto en su campo de fecha de revisión del entregable (Forma A.4.- Entregables del proyecto.)
	Criterios de salida	1. <i>Historias de usuario definidas, para este entregable en la forma Forma A.6.- Planeación de historias de usuario (Project Charter)</i> 2. <i>Definición de roles y equipo de trabajo.</i>

3.7 Etapa de Requerimientos.

El propósito de la etapa de requerimientos es guiar a los involucrados en el proceso de requerimientos del modelo DEPYM; al igual que la etapa de planeación, esta etapa se realiza para cada entregable. Aquí se analizan cada una de las historias de usuario definidas en la etapa de planeación y se detallan usando la Forma A.7.- Historia de usuario

La tabla 3.7 muestra el *script* de requerimientos del modelo DEPYM, indicando las actividades, formas y estándares para llevar a cabo la etapa de requerimientos del entregable actual.

Para la comprensión de cada uno de los requerimientos de las historias de usuario de la Forma A.6.- Planeación de historias de usuario (*Project Charter*) se realiza el llenado de la Forma A.7.- Historia de usuario, incluyendo cada una de las tareas necesarias para completar la historia de usuario.

Tabla 3.7. Script de Requerimientos del modelo DEPYM.

<i>Script</i> de requerimientos de DEPYM		
<i>Propósito del proceso:</i> Guiar al analista del proyecto y al equipo en todo el proceso de Planeación		
<i>Responsable del proceso:</i> Analista		
<i>Diagrama del proceso:</i>		
<pre> graph TD A[Criterios de entrada] --> B[Correcto entendimiento de la lista de usuarios] B --> C[Llenado de las formas de Historias de usuario] C --> D[Criterios de salida] </pre>		
	Criterios de entrada	1. <i>Historias de usuarios de la Forma A.6.- Planeación de historias de usuario (Project Charter)</i>
Paso	Actividad	Descripción
1	Correcto entendimiento de la lista de usuarios	<i>Para cada historia de usuario:</i>

		1. <i>Analizar y verificar las necesidades del cliente</i> 2. <i>Intercambiar puntos de vista entre los integrantes del equipo para recopilar y modelar lo que el sistema va a realizar.</i> 3. <i>Establecer acuerdos sobre los cambios de los requerimientos entre los clientes y el equipo de trabajo.</i>
2	Llenado de las formas de Historias de usuario	1. <i>Convertir los requerimientos de las historias de usuario en tareas específicas, precisas, no ambiguas, consistentes y completas del comportamiento del sistema.</i> 2. Para cada una de las historias de usuario se realiza el llenado de la forma de historia de usuario (Forma A.7.- Historia de usuario), incluyendo cada una de las tareas necesarias para completar cada historia.
	Criterios de salida	1. <i>Formas de historias de usuario llenas.</i>

3.8 Etapa de Generación de Pruebas Unitarias de DEPYM

El propósito de la etapa de Generación de Pruebas Unitarias de DEPYM es guiar al equipo en la implementación de cada una de las tareas de las historias de usuario a fin de producir un conjunto de pruebas unitarias que dirijan la construcción del código.

La tabla 3.8 muestra el script de Generación de pruebas unitarias del modelo DEPYM, mencionando las actividades a realizar, formas y estándares a ocupar.

Tabla 3.8. Script de Generación de Pruebas unitarias del modelo DEPYM.

Script de Generación de Pruebas Unitarias de DEPYM		
Propósito del proceso: Guiar al Ingeniero(a) de software en el proceso de generación de pruebas unitarias		
Responsable del proceso: Ingeniero de software		
Diagrama del proceso: <pre> graph TD A[Criterios de entrada] --> B[Llenar la forma de Pruebas unitarias para cada tarea] B --> C[Criterios de salida] </pre>		
	Criterios de entrada	1. Historias de usuario llenas con tareas específicas definidas
Paso	Actividad	Descripción
1	Llenar el formato de Pruebas unitarias para cada Tarea	1. Para cada una de las tareas definir las “n” pruebas unitarias que el código deberá pasar de acuerdo del patrón Arraging-Act-Assert. 2. Para cada una de las tareas llenar la forma de pruebas unitarias (Forma A.8.- Definición de pruebas unitarias para cada Tarea)
	Criterios de salida	1. Pruebas Unitarias definidas para cada tarea definida en el Forma A.7. Historia de usuario

3.9 Etapa de Codificación de DEPYM.

El propósito esta etapa es guiar el proceso de codificación ágil del proyecto de software, el cual abarca la construcción de código a partir de las pruebas unitarias definidas. La tabla 3.9 muestra el *script* de codificación del modelo DEPYM, indicando la secuencia de actividades a seguir, formas y estándares a usar dentro de esta etapa.

Tabla 3.9. Script de Codificación del modelo DEPYM.

Script de Codificación de DEPYM
Propósito del proceso: Guiar al Ingeniero(a) de software en el proceso de generación de pruebas unitarias
Responsable del proceso: Ingeniero de software
Diagrama del proceso
<pre>graph TD; A[Criterios de entrada] --> B[Codificación]; B --> C[Refactoring]; C --> D[Integración al código base]; D --> E[Criterios de salida];</pre>

	Criterios de entrada	1. Pruebas unitarias para cada tarea definidas 2. Estándar de codificación
Paso	Actividad	Descripción
1	Codificación	1. De acuerdo al estándar de codificación realizar el código más simple que pase las pruebas unitarias definidas para cada tarea
2	Refactoring	1. Aplicar técnicas de refactoring para mejorar el código existente, tomando en cuenta que las pruebas sigan pasando.
3	Integración al código base	1. Al terminar las pruebas para una tarea integrar al código base del proyecto
	Criterios de salida	1. Estándar de codificación actualizado 2. Código que pase las pruebas

3.10 Etapa de Cierre del Modelo DEPYM

La etapa de cierre en el modelo DEPYM dirige el proceso de revisión y cierre del entregable y da una retroalimentación de lo aprendido a lo largo del *entregable*. La tabla 3.10 muestra el script de cierre del modelo DEPYM.

Tabla 3.10. Script de Cierre del modelo DEPYM.

Script de Cierre de DEPYME		
Propósito del proceso: Guiar al líder del proyecto y al equipo en todo el proceso de Cierre		
Responsable del proceso: líder de proyecto		
Diagrama del proceso		
<pre> graph TD A[Criterios de entrada] --> B[Reunión para revisión por etapas] B --> C[Medición del logro de objetivos] C --> D[Presentación de resultados al cliente] D --> E[Planificación del siguiente entregable] E --> F[Criterios de salida] </pre>		
	Criterios de entrada	1. <i>Formas de historias de usuario terminadas para este entregable,</i> 2. <i>Formas de pruebas unitarias terminadas</i> 3. <i>Código probado</i>
Paso	Actividad	Descripción

1	Reunión para revisión de Etapas	1. <i>.Comentar los problemas que aparecieron en cada una de las etapas o documentos.</i> 2. <i>Comentar las experiencias del equipo en general.</i>
2	Medición del logro de objetivos	1. <i>Comparar el valor ganado obtenido a la fecha con lo planeado en la etapa de planeación.</i> 2. <i>Evaluar los resultados.</i> 3. Se actualiza la forma de Planeación de historias de usuario (Forma A.6.- Planeación de historias de usuario (Project Charter))
4	Presentación de resultados al Cliente	1. <i>El equipo elabora una presentación de lo realizado al cliente y lo presenta.</i> 2. <i>El cliente aporta comentarios y mejoras al proceso.</i>
5	Planificación del Siguiente Entregable	1. <i>Dar soluciones a los problemas que se han presentado por cada entregable</i> 3. <i>Determinar las funcionalidades que se desarrollarán en el entregable siguiente.</i> 4. <i>Definir qué se entregará al final de cada iteración y cuantos incrementos habrá en total y cada cuando</i> 5. Actualizar la forma de entregables del proyecto con la información del siguiente entregable(Forma A.4.- Entregables del proyecto) 6. Actualizar la forma de Planeación de historias de usuario para el siguiente entregable (Forma A.6.- Planeación de historias de usuario (Project Charter))
	Criterios de salida	1. <i>Forma de entregables actualizada (Forma A.4.- Entregables del proyecto)</i> 2. <i>Forma de planeación de historias de usuario actualizada(Forma A.6.- Planeación de historias de usuario (Project Charter))</i>

3.11 Etapa Post Mortem del Modelo DEPYM

La etapa de proceso de Post Mortem se realiza una sola vez, al final del último release. Dirige el proceso de revisión y cierre del proyecto con el objetivo de obtener retroalimentación de lecciones aprendidas y propósitos de mejora a lo largo de todo el proyecto. La tabla 3.11 muestra el script de Post Mortem del modelo DEPYM.

Tabla 3.11. Script de Cierre del modelo DEPYM.

Script de Post Mortem de DEPYME		
Propósito del proceso: Guiar al Líder del proyecto y al equipo en el proceso de Post Mortem		
Responsable del proceso: Líder de proyecto		
Diagrama del proceso		
<pre> graph TD A[Criterios de entrada] --> B[Reunión para mejora de procesos] B --> C[Criterios de salida] </pre>		
	Criterios de entrada	1. Releases terminados y validados con el cliente 2. Código probado
Paso	Actividad	Descripción
1	Reunión de mejora de procesos	1. Comentar los problemas que aparecieron en cada una de los releases. 2. Comentar las experiencias del equipo en general a lo largo de todo el proyecto 3. Realizar un análisis global del proyecto con el propósito de

		<i>encontrar áreas de mejora aplicables a todo el proyecto o que se hayan encontrado repetidamente en cada entregable. Actualizar la forma de propósitos de mejora Forma A.11</i>
	Criterios de salida	1. <i>Forma de mejora de procesos actualizada Forma A.11</i>

3.11 Instrumento de Evaluación

Para el instrumento de evaluación se definieron métricas apropiadas para entornos ágiles, que son:

1.- Valor entregado al cliente.- Es una medida cualitativa y de opinión que permite determinar si el proyecto está cumpliendo con las expectativas del cliente de acuerdo a cada entregable [Dubinsky, 2005]. En un entorno ágil se debe maximizar el valor que se entrega y los resultados deben ser medidos en términos de valor entregado al cliente. Mediante esta métrica, el cliente puede conocer la velocidad con que retorna su inversión. La evaluación se realizó mediante la siguiente ponderación:

A - Se cumple con la expectativa y se supera

B – Cumple la expectativa satisfactoriamente

C - Cumple regularmente la expectativa

D - Cumple las expectativas con problemas de calidad

E - No cumple la expectativa.

Esta métrica es evaluada por el cliente, en cada *release*.

2.- Velocidad de desarrollo.- Permite evaluar la fecha de finalización del proyecto y/o saber de qué requerimiento/entregable se dispondrá en una fecha determinada en términos de días hábiles de proyecto [Johnsen, 2002]. Esta métrica se calcula usando la siguiente fórmula

$$Vd = \sum(hg) / (i * h)$$

Donde:

Vd= Velocidad de desarrollo en días

hg= horas de desarrollo acumuladas

i = número de integrantes del equipo

h = número de horas por día (regularmente 8)

Esta métrica debe ser calculada por el líder de proyecto al término de cada tarea, historia de usuario o del entregable actual.

3.- Horas pendientes en la iteración.- Permite conocer el porcentaje de avance del proyecto en base al valor ganado de las horas invertidas en el proyecto y a los entregables realizados [Kan, 2002]. Se calcula con la siguiente fórmula:

$$hp = n - \text{sum}(hg)$$

Donde:

hp= Horas pendientes

i = total de horas estimadas del proyecto

hg= horas de desarrollo acumuladas

De igual forma que la métrica anterior, ésta deberá ser calculada por el líder de proyecto al término de cada tarea, historia de usuario o cada fin de entregable.

4.- Número de defectos por KLOC (porcentaje de defectos por cada 1,000 líneas de código).- Con esta métrica se mide la densidad de defectos en cada entregable con el propósito de disminuirla [Talby, 2005], se calcula con la siguiente fórmula:

$$dl = \text{sum}(d) / 100$$

donde:

dl = densidad de defectos

d = números de defectos por cada 1000 líneas de código

Esta métrica es calculada y evaluada por el líder de proyecto, al final de cada tarea, historia de usuario o iteración.

4. Resultados y discusión.

En este capítulo se presentan los resultados obtenidos al aplicar el modelo DEPYM. Como ya se ha explicado a lo largo del documento, el modelo DEPYM se ha desarrollado como un modelo ágil, con la característica de ser dirigido por pruebas, y su ámbito de aplicación está en las micros y pequeñas empresas y para proyectos de 18 meses como máximo. Asimismo, los equipos de desarrollo deberán ser de máximo 9 integrantes.

Para la validación del experimento se seleccionaron 3 empresas del estado de Veracruz que cumplieran con estas características. Estas compañías ya habían recibido capacitación en modelos como PSP/TSP y Moprosoft, además de conocer de manera general métodos ágiles como XP. Estas empresas no ocupaban formalmente ningún modelo de mejora de procesos por considerarlos demasiado complejos, grandes y pesados para sus equipos de desarrollo, así como para sus tiempos de entrega y presupuestos económicos. Además consideraban que usar al 100% algún método como PSP/TSP o Moprosoft los pondría fuera del mercado por el tiempo excesivo que requerirían para terminar el proyecto.

En este trabajo de tesis, para aplicar el modelo DEPYM se seleccionó un proyecto que cumpliera con los requisitos de ser menor de 18 meses y cuyos equipos de desarrollo fueran pequeños para cada una de las 3 empresas. Además se seleccionó un proyecto que, por sus tiempos de entrega, fuera factible de ser analizado durante el desarrollo de esta investigación. Para la aplicación del modelo se concientizó a los directivos de las empresas en la importancia de aplicar un modelo de procesos en la construcción de sus productos de software, además de que era indispensable que el personal de recursos humanos de la empresa fuera parte del equipo de desarrollo, para participar activamente en la obtención de requisitos y en la revisión de las entregas frecuentes y en la retroalimentación de los entregables.

4.1. Descripción de los experimentos de prueba.

Esta investigación tomó como referencia al instrumento de evaluación descrito en la sección 3.11, que fue aplicado a las 3 empresas que participaron en esta investigación. Para lograr

la participación de las empresas, se invitó a empresas del sector de TI del estado de Veracruz e instituciones del sector educativo superior. El nombre de las empresas no se muestra a fin de cumplir con el secreto empresarial, es por ello que se refieren como empresa 1, empresa 2 y empresa 3. En las siguientes secciones se muestran los resultados obtenidos en cada uno de los experimentos de las empresas participantes.

4.2. Resultados de la aplicación del modelo.

4.2.1. Caso 1

El proyecto 1 de la empresa 1, consiste en un Software basado en la nube de Microsoft Azure, para el control de ventas de alimentos congelados a través de un dispositivo móvil con el sistema operativo Android. La información de los clientes y sus rutas de reparto deben ser visualizadas en mapas de Google. La información de ventas deberá ser procesada para efectos de administración de pagos, inventario, compras y facturación. El sistema deberá contar con un módulo de reportes de información ejecutiva. Se aplicó el modelo DEPYM para determinar los requerimientos mínimos del proyecto, definir la arquitectura de la aplicación, el equipo de trabajo y los entregables en que iba a ser dividido el proyecto, así como las historias de usuario para cada entregable. El proyecto tuvo una duración de 9 meses generando 11 entregables. El equipo estuvo formando por 3 ingenieros de tiempo completo con jornadas de trabajo de 40 hrs/sem. El proyecto tuvo un total de 4,320 hrs. y 180 días hábiles.

Los ingenieros de software mostraron una actitud positiva al aplicar el modelo DEPYM, ya que era un reto para ellos aplicar un nuevo paradigma donde se definieran primero las pruebas y después se construyera el código mínimo para cumplirlas. Asimismo, los ingenieros se mostraron muy motivados por la tecnología que usaba el proyecto, les resultó atractiva e innovadora.

Los clientes que formaron parte del proyecto se involucraron activamente durante las etapas de análisis y requerimientos de cada entregable, además de que conocían y operaban el sistema anterior. Su actitud fue favorable al empezar a probar módulos

funcionales desde el primer *reléase* y pudiendo dar retroalimentación desde la primera etapa.

El director de la empresa estuvo de acuerdo con probar el software continuamente y dar sus opiniones y puntos de vista para mejoras. Al final del proyecto el software había sido tan revisado y retroalimentado que todos los usuarios, mandos medios y directivos lo conocían y usaban a la perfección.

Cada entregable fue categorizado de acuerdo al valor que aportaba al proyecto global, entonces los entregables más importantes se realizaron primero, dejando los de menor importancia al final del proyecto. Para este proyecto se definieron los siguientes entregables (releases):

1. Rutas.- Administración de las rutas de ventas.
2. Ventas.- Módulo de ventas con el dispositivo móvil con sistema operativo Android.
3. Control de Activos.- Administración de conservadores que entregan en comodato.
4. Almacén.- Administración de la cámara donde se almacenan los alimentos congelados.
5. Reportes.- Módulo de información ejecutiva.
6. Facturación.- Facturación electrónica por internet (CFDi).
7. Compras.- Administración de compras y proveedores.
8. Adm. Financiera.- Administración de cuentas por cobrar y cuentas por pagar.
9. Interfaz contable.- Reportes para la interfaz con el sistema de contabilidad.
10. Control de bancos.- Administración de las cuentas bancarias.
11. Interfaz de Recursos Humanos.- Generación de reportes de comisiones de ventas para nómina.

La tabla 4.1 muestra los resultados obtenidos aplicando las 4 métricas del modelo DEPYM en el proyecto 1.

Tabla 4.1. Resultados obtenidos del proyecto 1 del experimento de evaluación de DEPYM.

ID	Nombre Entregable	Horas planeadas	Prioridad	%Valor	Horas Acumuladas	Evaluación del Cliente	Vel. Días	horas pendientes	Defectos (LOC Plan)	Defectos KLOC Actual
1	Rutas	777.6	ALTA	18	777.6	A	32.4	3,542.4	35	55
2	Ventas	648	ALTA	15	1425.6	A	59.4	2,894.4	32	49

3	Control Activos	475.2	ALTA	11	1,900.8	A	79.2	2,419.2	30	51
4	Almacén	388.8	ALTA	9	2,289.6	A	95.4	2,030.4	27	36
5	Reportes	388.8	MEDIANA	9	2,678.4	A	111.6	1,641.6	26	50
6	Facturación	432	MEDIANA	10	3,110.4	A	129.6	1,209.6	24	47
7	Compras	259.2	MEDIANA	6	3,369.6	A	140.4	950.4	22	42
8	Adm. Financiera	259.2	MEDIANA	6	3,628.8	B	151.2	691.2	19	37
9	Interfaz Contable	172.8	BAJA	4	3,801.6	B	158.4	518.4	15	32
10	Control de Bancos	216	BAJA	5	4,017.6	B	167.4	302.4	13	25
11	Interfaz Rec. Hum.	302.4	BAJA	7	4,320	C	180	0	13	19
Totales		4320		100	4320		180			

4.2.2. Caso 2.

El proyecto 2, desarrollado en la empresa 2, consiste en un software en ambiente web, usando la arquitectura MVC, y que puede ser instalado en la nube de Microsoft Azure, Hosting tradicional o en un servidor local corporativo, para el control del presupuesto de obras de empresas de la construcción. La información de las obras y sus gastos deben ser administradas a través de los ingenieros responsables para conocer en todo momento el monto ejercido del presupuesto de la obra y el monto por ejercer. Además, se podrá contar con información detallada de los gastos en materiales, maquinaria, mano de obra y gastos sociales como impuestos. La información de compras deberá ser procesada para efectos de administración de pagos, inventario, compras y facturación. El sistema deberá contar con un módulo de reportes de información ejecutiva. Se aplicó el modelo DEPYM para determinar los requerimientos mínimos indispensables del proyecto para iniciar la etapa de configuración inicial incluyendo: definir la arquitectura de la aplicación, el equipo de trabajo y los entregables en que debía a ser dividido el proyecto, así como las historias de usuario para cada entregable.

Al igual que en el caso 1, los Ingenieros de software mostraron una actitud positiva y motivados al aplicar el modelo DEPYM. Los stakeholders por parte del cliente se involucraron en el proyecto en todas las etapas y en cada entregable. Esto fue muy importante ya que solo ellos conocían bien las reglas del negocio de una empresa constructora.. El gerente de la empresa estuvo también de acuerdo con probar el software y final de cada entregable el usuario ya lo había probado y usado

Este proyecto tuvo una duración de 7 meses (140 días hábiles) generando 7 entregables, con un equipo de 2 personas. La tabla 4.2 muestra los resultados obtenidos aplicando DEPYM en el caso 2

Al igual que en el caso 1 cada entregable fue categorizado de acuerdo al valor que aportaba al proyecto global, de la misma forma los entregables más importantes se programaron primero, dejando los que aportaban menos valor al final del proyecto. Para este proyecto se definieron los siguientes entregables (releases):

1. Control de obra.- Administración del presupuesto de obra, determinación del presupuesto ejercido, por ejercer y presupuesto original.
2. Estimación de obra.- Determinación de los precios unitarios para una obra nueva y determinación del costo y utilidad.
3. Control de materiales.- Administración materiales y almacén, que se ejercen en requisiciones de material por parte de los residentes de obra
4. Control de mano de obra.- Administración de los recursos humanos, cálculo de impuestos.
5. Maquinaria y equipo.- Administración de maquinaria, control de horas de uso, maquinaria arrendada..
6. Facturación.- Facturación electrónica por internet (CFDi).
7. Compras.- Administración de compras de materiales, administración de proveedores.

Tabla 4.2. Resultados obtenidos del proyecto 2 del experimento de evaluación de DEPYM

ID	Nombre Entregable	Horas planeadas	Prioridad	%Valor	Horas Acumuladas	Evaluación del Cliente	Vel. Días	horas pendientes	Defectos KLOC Plan	Defectos KLOC Actual
1	Control Obra	515	ALTA	23	515.2	A	32.2	1,724.8	34	55
2	Estimación de obra	381	ALTA	17	896	A	56	1344	32	49
3	Control de Materiales	336	ALTA	15	1232	B	77	1008	31	51
4	Control Mano Obra	314	MEDIANA	14	1,545.6	C	96.6	694.4	29	36
5	Maquinaria y Equipo	269	MEDIANA	12	1,814.4	A	113.4	425.6	29	50
6	Adquisiciones	246	MEDIANA	11	2,060.8	B	128.8	179.2	27	47
7	Facturación	179	BAJA	8	2,240	A	140	0	25	31
Totales		2240		100	2,240		140			

4.2.3. Caso 3.

El proyecto 3 desarrollado en la empresa 3, consiste en un software de una institución académica, que cuenta con un modelo probado que ayuda a las empresas a realizar un proceso completo de planeación estratégica. Este proyecto permite organizar a un grupo de líderes de la empresa para generar los valores, misión, visión, programas estratégicos y un cuadro de mando integral para el seguimiento de esos programas estratégicos. El proyecto consiste en construir la herramienta de software en la nube de Microsoft Azure, para automatizar este modelo que actualmente se realiza de forma manual con ayuda de *post it* y una hoja de cálculo. La información recolectada de las técnicas de lluvias de ideas podrá ser capturada a través de una página web o bien desde un dispositivo móvil. La información se guardará para su posterior procesamiento a través de técnicas de agrupamiento y minería de texto. El sistema deberá contar con un módulo para la creación de un cuadro de mando integral para darle seguimiento a los programas estratégicos.

Igual que en los dos casos anteriores, DEPYM se aplicó en este proyecto para determinar los requerimientos mínimos para arrancar el proyecto, definir la arquitectura, el equipo de trabajo, los entregables, las historias de usuario y las tareas para cada entregable. Los ingenieros de software mostraron una actitud positiva al aplicar el modelo DEPYM, ya que la técnica TDD era nueva para ellos. Los involucrados de la institución educativa y que formaron parte del proyecto se involucraron de forma favorable en las diferentes etapas de de cada entregable. Su participación fue indispensable ya que solo ellos conocían el modelo de planeación estratégica que se trataba de automatizar, su actitud fue favorable al empezar a probar módulos funcionales desde el primer *release* y pudiendo dar retroalimentación desde la primera etapa. El autor del modelo formó parte del equipo de desarrollo, lo cual resultó muy enriquecedor a su experiencia de 15 años aplicando su modelo en al menos 40 empresas de forma manual. Asimismo, los involucrados de la institución académica estuvieron muy interesados en probar el software continuamente y dar sus propósitos de mejora. El equipo de desarrollo está formado por 2 ingenieros;

Este proyecto está planeado a 14 meses con 17 entregables, y se encontraba en desarrollo al momento de escribir este documento con un **25%** de avance, de acuerdo al valor ganado obtenido a la fecha y al número de horas acumuladas ejercidas de 1,120 contra un total de 4,480 del proyecto. La tabla 4.3 muestra los resultados obtenidos en el proyecto 3.

Igual que en los casos anteriores cada entregable fue categorizado de acuerdo al valor que aportaba al proyecto global. Para este proyecto se definieron los siguientes:

1. Conocer a la empresa.- Captura de datos generales de la empresa a la que se aplicara el modelo de planeación estratégica.
2. Compromiso del CEO.- Captura de datos generales del director general de la empresa, para refrendar su compromiso con el proyecto.
3. Definición del equipo de trabajo.- Determinación en base a técnicas de selección al grupo de líderes de la empresa que participará en el proceso de planeación estratégica.
4. Integración del equipo de trabajo.- Captura de datos de los integrantes del *staff* o integrantes por parte de la empresa que impartirá la consultoría de planeación estratégica.
5. Organizar el plan de acción.- Planeación de las actividades a realizar para llevar a cabo de forma eficiente cada una de las etapas del modelo.
6. Determinar la visión.- Mostrar información que explica el concepto de visión de una empresa, a través de ejemplos, videos, imágenes, entre otros.
7. Lluvia de ideas.- Captura de la información a través de una técnica de lluvias de ideas de las diferentes ideas que llevarán a determinar la visión de la empresa
8. Definir valores.- Captura de los diferentes valores que cada líder de opinión cree deben ser parte de los valores de la empresa.
9. Análisis del entorno.- Determinar los diferentes entornos en los que se aplicará la técnica del FODA, ejemplo: nivel interno, externo, macroeconómico, con la competencia, ámbito internacional, nacional, entre otros.
10. Clasificación de la información.- Captura de la información, clasificación y depuración de la información de los n elementos del FODA, en un proceso semi-automático por parte del equipo staff y la ayuda del software.
11. Definición de proyectos estratégicos.- Determinar y almacenar los n proyectos estratégicos resultantes del análisis de los elementos del FODA

12. Determinación de prioridades.- De acuerdo al modelo, se deben elegir los proyectos estratégicos que aporten mas a la visión de la empresa o que a juicio de los líderes de opinión sean más importantes

13. Formulación de la misión.- Generación de la misión de la empresa en base a los programas estratégicos elegidos.

14. Direccionar los objetivos estratégicos.- Proceso para alinear los programas estratégicos a la visión de la empresa.

15. Desarrollar el mapa estratégico.- Graficar los programas estratégicas en alguna herramienta como *wbs* o *ishikawa*

16. Diseñar el sistema de medición.- Definir los indicadores, cédulas de control y responsables para cada uno de los programas estratégicos

17. Elaborar el cuadro de mando integral.- Elaboración del cuadro de mando integral para medir el avance de cada programa en base a semáforos.

Tabla 4.3. Resultados obtenidos del proyecto 3 del experimento de evaluación de DEPYM.

ID	Nombre Entregable	Horas planeadas	Prioridad	%Valor	Horas Acumuladas	Evaluación del Cliente	Vel. Días	horas pendientes	Defectos KLOC Plan	Defectos KLOC Actual
1	Conocer la empresa	134.4	ALTA	3	134.4	A	8.4	4345.6	35	55
2	Compromiso del CEO	89.6	ALTA	2	224	A	14	4256	32	49
3	Definición del equipo de trabajo	134.4	ALTA	3	358.4	A	22.4	4121.6	21	51
4	Integración del equipo de trabajo	179.2	ALTA	4	537.6	A	33.6	3942.4	29	51
5	Organizar el plan de acción	403.2	MEDIANA	9	940.8	A	58.8	3539.2	28	51
6	Determinar la visión	179.2	MEDIANA	4	1120	A	70	3360	28	51
7	Lluvia ideas	268.8	MEDIANA	6	0	-	0	4480	27	0
8	Definir Valores	268.8	MEDIANA	6	0	-	0	4480	25	0
9	Análisis del entorno	313.6	MEDIANA	7	0	-	0	4480	24	0
10	Clasificación de la información	313.6	MEDIANA	7	0	-	0	4480	26	0
11	Definición de proyectos estratégicos	448	MEDIANA	10	0	-	0	4480	25	0

1	Determinación de prioridades	134.4	BAJA	3	0	-	0	4480	23	0
1	Formulación de la misión	179.2	BAJA	4	0	-	0	4480	21	0
1	Direccionar los objetivos estratégicos	224	BAJA	5	0	-	0	4480	20	0
1	Desarrollar el mapa estratégico	313.6	BAJA	7	0	-	0	4480	17	0
1	Diseñar el sistema de medición	224	BAJA	5	0	-	0	4480	15	0
1	Elaborar el cuadro integral de mando	672	BAJA	15	0	-	0	1120	13	0
Totales		4480		100	1120		280			

4.3 Análisis de resultados.

Los resultados obtenidos en el caso 1 en cuanto a las 4 métricas propuestas por DEPYM se analizan de la siguiente manera

1.- Valor entregado al cliente.- Se obtuvieron resultados satisfactorias con respecto a la evaluación. De 11 entregables, se obtuvieron 7 con calificación A (se cumple con la expectativa y se supera), 3 con B (cumple la expectativa satisfactoriamente) y solo 1 con C (cumple la expectativas con problemas de calidad). Consideramos que este resultado resultó satisfactorio debido a que el cliente fue parte del equipo y ayudó a decidir cuáles eran las prioridades del proyecto, dando oportunidad a detectar defectos y corregirlos en tiempos adecuados.

2.- Velocidad de desarrollo.- Se pudo conocer la velocidad de desarrollo y conocer con exactitud el avance por día de los 180 días planeados. Esto fue gracias a la simplicidad del modelo y las formas propuestas que permitieron conocer en todo momento el avance real del proyecto.

3.- Horas pendientes en la iteración.- Usando esta métrica se conoció con exactitud el número de horas pendientes para terminar el proyecto, esto es debido a las formas del modelo DEPYM que permiten registrar de forma simple las tareas, historias de usuario y

entregables terminados y conocer el valor obtenido a una fecha determinada, y por diferencia el tiempo restante del proyecto.

4.- Número de defectos por KLOC.- En cuanto a la densidad de defectos se logró disminuir a medida que se avanzaba en cada entregable, aunque siempre se estuvo por encima de lo planeado. Esto fue debido a las entregas frecuentes, a la integración continua que nos permitieron obtener retroalimentación constante que ayudó a la detección de defectos.

En el caso 2 se obtuvieron los siguientes resultados en las 4 métricas propuestas:

1.- Valor entregado al cliente.- De 7 entregables se obtuvieron 3 con A (se cumple con la expectativa y se supera), 3 con B (cumple la expectativa satisfactoriamente) y solo 1 con C (cumple la expectativas con problemas de calidad). Igual que en el caso anterior, consideramos que el resultado fue producto de incorporar al equipo al cliente.

2.- Velocidad de desarrollo.- Se pudo conocer la velocidad de desarrollo y conocer con exactitud el avance por día de los 140 días planeados, esto se logró con la ayuda de la del modelo y las formas simples.

3.- Horas pendientes en la iteración.- Usando esta métrica se conoció con exactitud el número de horas pendientes para terminar el proyecto, al igual que el caso 1 se logro debido a las formas del modelo Depym.

4.- Número de defectos por KLOC.- En cuanto a la densidad de defectos se logró disminuir a medida que se avanzaba en cada entregable, aunque siempre se estuvo por encima de lo planeado. Al igual que el caso 1 las entregas frecuentes y la integración continua permitieron obtener constante retroalimentación que ayudo a la detección de defectos en el código.

Para el experimento 3 se obtuvieron los siguientes resultados para el avance del 25% obtenido a la fecha de publicación de este documento:

1.- Valor entregado al cliente.- Se obtuvieron resultados satisfactorias con respecto a la evaluación, de 17 entregables, se han completado 6 a la fecha y se obtuvieron 6 con A (Se cumple con la expectativa y se supera), El resultado parcial se obtuvo debido a que el autor del modelo a automatizar fue parte del equipo, y los demás miembros del equipo estaban muy conscientes de que el uso de procesos ayuda a mejorar la calidad de los productos. Por tanto participaron activamente a decidir cuáles eran las prioridades del proyecto, evaluar y retroalimentar las entregas, dando oportunidad a eliminar defectos en el código en tiempos adecuados.

2.- Velocidad de desarrollo.- Se pudo conocer la velocidad de desarrollo y conocer con exactitud el avance por día de los 280 días planeados, al igual que el caso 1 y 2, el modelo permitió capturar los avances reales del proyecto en los 6 entregables desarrollados y así conocer en todo momento el avance real del proyecto.

3.- Horas pendientes en la iteración.- Usando esta métrica se conoció con exactitud el número de horas pendientes para terminar el proyecto, en los 6 entregables terminados las formas del modelo Depym-.

4.- Número de defectos por KLOC.- En cuanto a la densidad de defectos se logró disminuir a medida que se avanzaba en cada uno de los 6 entregables terminados, aunque siempre se estuvo por encima de lo planeado. Esto fue debido a las entregas frecuentes en los *releases* terminados, a la integración continua que nos permitieron obtener retroalimentación directa del autor del modelo y su grupo de colaboradores y que ayudo a la detección de defectos.

5. Conclusiones.

En base a los experimentos realizados utilizando DEPYM en 3 diferentes empresas, consideramos que el resultado de la investigación es favorable, ya que se han obtenido resultados de evaluación satisfactorios a la fecha por parte de los clientes y de acuerdo a las métricas establecidas. Además, de los resultados de este trabajo podemos concluir que

es de vital importancia que las empresas utilicen un modelo de mejora de procesos, en este caso uno Ágil y dirigido por pruebas, para mejorar sus procesos y asegurar la terminación en tiempo y calidad de sus compromisos con clientes. Hacer esta correcta implementación de los procesos redundará en proyectos más exitosos.

Un dato relevante para las empresas mexicanas es que esta investigación ha mostrado que es factible usar un modelo Ágil con enfoque TDD en empresas mexicanas micro y pequeñas, donde la mayoría de sus proyectos son menores de 18 meses y sus equipos de ingenieros de desarrollo no superan a los 10 integrantes por proyecto. Sin embargo, es importante hacer notar que, de los resultados mostrados en el capítulo 4, encontramos que todas las empresas que usaron DEPYM necesitan hacer cambios culturales que les permitan adoptar de manera adecuada un modelo de mejora de procesos.

5.1. Aportaciones.

Las contribuciones de esta investigación son las siguientes:

1. Un modelo de mejora de procesos ágil adaptable a la industria mexicana y con. Área de aplicación son empresas micro y pequeñas con equipos de desarrollo menores de 10 integrantes y en proyectos de no mayor 18 meses de duración.
2. El Diseño de un script de Procesos, de formas y de estándares adaptables a empresas PyMEs

5.2. Consideraciones finales

Es de suma importancia el uso eficaz de modelos de desarrollo de software adaptables a la cultura y tamaño de la industria del país, con los procesos mínimos requeridos para garantizar el cumplimiento del calendario y dar el mayor valor al cliente, sin perder de vista la calidad.

En la introducción del primer capítulo, habíamos planteado que “posiblemente”, el uso de una metodología ágil basado en el enfoque dirigido por pruebas podría ayudar a las empresas Pymes mexicanas a crear productos software con calidad en los tiempos y costos presupuestados. Ahora, después de llevar a cabo esta investigación, podemos “afirmar” que para mejorar la calidad de los productos de software, es necesario utilizar de manera formal algún modelo de mejorar de procesos, además de adaptar la cultura de la organización que involucre tanto a directivos de la empresa, mandos medios, ingenieros de software, analistas, *tester*. Además, encontramos que es fundamental incluir al clientes como parte del equipo de desarrollo y a los directivos de la empresa cliente; con esto podemos lograr disminuir el riesgo de de fracaso del proyecto, disminuir el riesgo de salirse del presupuesto de tiempo, disminuir el riesgo de rebasar el presupuesto de costos, aumentar la calidad del producto y mejorar el ambiente laboral.

Por tanto podemos concluir que institucionalizar un modelo de mejora de procesos dentro de una empresa de desarrollo de software es una forma de garantizar el éxito en el proceso de desarrollos de productos de software dentro de una empresa.

5.3. Trabajo futuro

Un punto importante a continuar en esta investigación está en refinar el modelo, en base a resultados de los casos 1, 2 y 3 y en la construcción de una herramienta de software en la nube para automatizar el modelo y que esté disponible para la comunidad en general. Actualmente ya se tiene reservado el dominio <http://depym.com>, donde se publicará información del proyecto y las publicaciones que se han obtenido relacionadas al modelo. Además, en este portal se creará una herramienta de software para automatizar el modelo, para este fin se involucrarán al menos 2 tesis de licenciatura en alguna carrera afín y 1 tesis de maestría. Posteriormente se pretende crear un *framework* de desarrollo que permita automatizar la generación de código a partir de las pruebas unitarias definidas en la herramienta y asociarlo al entorno de desarrollo del ingeniero(a) de software como Java o C++. Para esto se requerirá la colaboración de al menos 2 tesis de maestría.

Publicaciones generadas por esta investigación.

2010. Centeno-Téllez A, Gómez-Gil P. "Metodología de desarrollo para la mejora de procesos de software a través del desarrollo dirigido por pruebas." Memorias del 3°. Coloquio Interdisciplinario de Posgrado, Innovación e Investigación en la Solución de Problemas Sociales." Universidad Popular Autónoma del Estado de Puebla. 19 de Marzo 2010.

2012. Centeno-Téllez A, Gómez-Gil, P. "DEPYM: modelo de mejora del procesos de software con enfoque de desarrollo dirigido por pruebas". Publicadas en las Memorias del 5°. Coloquio interdisciplinario de doctorado Universidad Popular Autónoma del Estado de Puebla. 2012.

2012. Centeno-Téllez A, Gómez-Gil, P. "DEPYM: Modelo de mejora del procesos de software con enfoque de desarrollo dirigido por pruebas". Publicadas en las Memorias del Coloquio de Investigación Multidisciplinaria CIM-2012 , organizado por el Instituto Tecnológico de Orizaba, llevadas a cabo en Orizaba, Ver., los días 25 y 26 de octubre de 2012. Pp. 1-8

Referencias.

- [Ambler's, 2008] Resultados de Scott Ambler's December 2008 Software Development Project Success Survey publicado en www.ambysoft.com/surveys/success2008.html,
- [Bauer, 1972] Bauer, FL " Ingeniería del software ", Procesamiento de la Información, 71, 1972.
- [Barriocanal, 2002] E. Barriocanal, M. Urb'an, I. Cuevas, and P. Pérez. An experience in integrating automated unit testing practices in an introductory programming course. *ACM SIGCSE Bulletin*, 34(4):125–128, December 2002.
- [Beck, 2002] Beck Kent, *Test Driven Development by example*, Addison Wesley, Páginas 6-10
- [Bujaidar, 2008] Bujaidar Andres, *MexicoFirst, Software guru*, Agosto-Octubre 2008, Mexico 2008
- [Christensen, 2003] H. B. Christensen. Systematic testing should not be a topic in the computer science curriculum! In *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education*, pages 7–10. ACM Press, 2003.
- [Cockburn, 2002] Alistair Cockburn. "Crystal Clear. Una metodología de tracción humana para los equipos pequeños, incluyendo las siete propiedades de proyectos de software eficaz". Borrador. Los seres humanos y la tecnología, la versión del 27 de febrero de 2002.
- [Erdogmus, 2005] H. Erdogmus. On the effectiveness of test-first approach to programming. *IEEE Transactions on Software Engineering*, 31(1):1–12, January 2005.
- [Highsmith, 2001] Jim Highsmith. "El Gran Debate Metodologías. Part 1". *Cortador IT Journal*, 14 (12), diciembre de 2001
- [Jones, 2004] Jones Christopher G. *TEST-DRIVEN DEVELOPMENT GOES TO SCHOOL*, CCSC: Rocky Mountain Conference, Pags. 1 Octubre 2004
- [Janzen, 2005] Janzen David S. *Software Architecture Improvement through Test Driven Development*, OOPSLA'05, October 16–20, 2005, San Diego, California, USA. ACM, October 2005
- [Janzen, 2006] D. Janzen and H. Saiedian. Test-driven learning: Intrinsic integration of testing into the CS/SE curriculum. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, pages 254–258. ACM Press, 2006.
- [Janzen, 2007] Janzen David S. *A leveled Examination of Test-Driven Development Acceptance*, 29th International Conference on Software Engineering (ICSE'07), IEEE Computer Society, Pag. 1, USA 2007
- [Larman, 2003] Craig Larman & V. Basili (2003). "Iterative and Incremental Development: A Brief History", *Computer (IEEE Computer Society)* 36 (6): 47-56.
- [Maximilien, 2003] Maximilien Michael E. *Assessing Test-Driven Development at IBM.*, pages 1. *Computer (IEEE Computer Society)* 2003

[Miller, 2004] Miller Keith W. *TEST DRIVEN DEVELOPMENT ON THE CHEAP: TEXT FILES AND EXPLICIT SCAFFOLDING*, CCSC: Northwestern Conference, Pags. 1 October 2004

[Miller, 2004] Miller Keith W. *TEST DRIVEN DEVELOPMENT ON THE CHEAP: TEXT FILES AND EXPLICIT SCAFFOLDING*, CCSC: Northwestern Conference, Pags. 2 Octubre 2004

[Newkirk, 2004] Newkirk James W. *Test Driven Development in Microsoft .NET*, Microsoft Press, Páginas 3-7

[Schwaber, 2010] K. Advanced Development Methods. SCRUM Development Process, July 2010.

[Beck, 1999] *Extreme Programming Explained: Embrace Change*. Addison-Wesley. ISBN 978-0-321-27865-4.

[Cockburn, 2004] Alistair Cockburn. *Crystal Clear, A Human-Powered Methodology for Small Teams*, Alistair Cockburn, October 2004, pages 336, paperback, Addison-Wesley Professional, ISBN 0-201-69947-8

[Hightsmith, 2000] Hightsmith, *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*, Jan 2000 New York: Dorset House, 392pp, ISBN 0-932633-40-4

[Davies, 2003] Davies, *The diffusion of information systems development methods*, *Journal of Strategic Information Systems* 12 p. 29-46 (2003)

[Palmer, 2002] Palmer & Felsing J.M. *A Practical Guide to Feature-Driven Development*. Prentice Hall. ISBN 0-13-067615-2)

[Bioul, 2010] G. Bioul, F. Escobar, M. Alvarez, A. Nardin, E. Ricci Aparicio, *Metodologías Ágiles, análisis de su implementación y nuevas propuestas*, CACIC 2010 - XVI CONGRESO ARGENTINO DE CIENCIAS DE LA COMPUTACIÓN, Argentina 2010

[Rosenberg, 2003] Rosenberg, Doug; Stephens, Matt (2003). *Extreme Programming Refactored: The Case Against XP*. Apress ISBN 1-590559-096-1.

[Schwaber, 2004] *Agile Project Management with Scrum*, Ken Schwaber, Microsoft Press, January 2004, 163pp, ISBN 0-7356-1993-X

[Erdogmus, 2005] Erdogmus, Hakan; Morisio, Torchiano. "On the Effectiveness of Test-first Approach to Programming". *Proceedings of the IEEE Transactions on Software Engineering*, 31(1). January 2005. (NRC 47445)

[Leff, 2001] Leff, Avraham; James T. Rayfield (September 2001). "Web-Application Development Using the Model/View/Controller Design Pattern". IEEE Enterprise Distributed Object Computing Conference. pp. 118–127

[slim3, 2013] url = google <http://code.google.com/p/slim3/>, tomado 01/sep/2013

[Wake, 2004] Wake William C., Refactoring workbook, Addison-Wesley Professional, Pags 240, 241.

[arrange-act-assert, 2013] url =arrangeactassert.com, tomada 02/sep/2013

[Reenskaug, 2003] Trygve Reenskaug. *The Model-View-Controller (MVC) Its Past and Present*, University of Oslo. Pags. 16. Agosto 2003, fecha de consulta 20.12.12, http://heim.ifi.uio.no/~trygver/2003/javazone-jaoov/MVC_pattern.pdf

[Rethinking, 2003] S. Edwards. *Rethinking computer science education from a test-first perspective*. In *Proceedings of the 18th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications: Educators' Symposium*, pages 148–155, 2003.

[Ruvalcaba, 2005] Ruvalcaba Mara, *Software Guru Edición Enero-Febrero 2005*, México Página 4.

[Standish Group, 2004] Standish Group International Inc, *CHAOS survey otoño 1994 - otoño 2004*

[SNIITI, 2009] Sistema Nacional de Indicadores de la Industria de TI (SNIITI), *Empresas Certificadas en Modelos de Calidad*, fecha de consulta 16.04.09, <http://www.edigital.economia.gob.mx/infoagent.aspx?docid=14>

[Thirumalesh, 2006] Bat Thirumalesh, Nagappan Nachiappan. *Evaluating the Efficacy of Test-Driven Development: Industrial Case Studies*. ISESE'06, September 2006, ACM. Página 2. Rio de Janeiro, Brazil

[Oktaba, 2005] Oktaba Hanna, *Modelo de procesos para Industria del software Moprosoft Version 1.3 por Niveles de Capacidad de Procesos*, Página 133, México 2005

[Patterson, 2003] A. Patterson, M. Kolling, and J. Rosenberg. *Introducing unit testing with BlueJ*. In *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education*, pages 11–15. ACM Press, 2003.

[Pressman, 2005] Roger Pressman. "Ingeniería del Software: Un Enfoque Práctico" Sexta Edición, McGraw-Hill.

[nyce, 2013] <http://www.nyce.org.mx/blog/?p=2284>

[Holovaty, 2005] Adrian Holovaty. *Django*. <https://www.djangoproject.com/>, 10 septiembre 2013

[Zend Technologies, 2012]. Zend Technologies .*Zend Framework*. 19 de septiembre de 2013; <http://www.framework.zend.com/>

[Avraham, 2001] Avraham Leff; Rayfield James T. (September 2001). «Web-Application Development Using the Model/View/Controller Design Pattern». *IEEE Enterprise Distributed Object Computing Conference*. pp. 118–127.

[Walls, 2010] Walls, Craig (November 28, 2010). *Spring in Action (Third edición)*. [Manning](#). pp. 700. ISBN 1-935182-35-8

[Husted, 2004] *Struts in Action: Building Web Applications with the Leading Java Framework*. Manning, pp. 90
ISBN-10: 1930110502

[James, 2010] Avery James, Holmes Jim, *Windows developer power tools*, O'Reilly Media, 2006, ISBN 0-596-52754-3 pp. 963–972 (18.6 Simplifying Web Development with Castle MonoRail)

[Sanderson, 2009] Sanderson Steven, *Pro ASP.NET MVC Framework*, Apress, 2009, ISBN 1-4302-1007-9, p. 12

[Ahmad, 2010] Sheikh Fahad Ahmad, *Test Driven Development with Continuous Integration: A Literature Review*, *International Journal of Computer Applications Technology and Research* Volume 2– Issue 3, 281 - 285, 2013

[Kahan, 2010] Naveed Khan, Muhammad Shahid Khan, Muhammad Ahmed Javed & Muhammad Abid Khan, *Global Journal of Computer Science and Technology Software & Data Engineering* Volume 13 Issue 7 Version 1.0 Year 2013 Type: Double Blind Peer Reviewed International Research Journal Publisher: Global Journals Inc. (USA) Online ISSN: 0975-4172 & Print ISSN: 0975-4350

[Dubinsky, 2005] Y. Dubinsky, O. Hazzan, and A. Keren, "Introducing Extreme Programming into a Software Project at the Israeli Air Force", *6th International Conference on Extreme Programming and Agile Processes in Software Engineering*, 2005.

[Johnsen, 2002] K. Johnsen, R. Stauffer, and D. Turner, "Learning by Doing: Why XP Doesn't Sell", *Proceedings of the XP/Agile Universe*, 2001.

[Kan, 2002] Kan, S., *Metrics and Models in Software Quality Engineering (2nd Edition)*, Addison-Wesley 2002.

[Talby, 2005] D. Talby, et al, "A Process-Complete Automatic Acceptance Testing Framework", *SwSTE*, 2005.

6. Apéndice A. Formas del modelo DEPYM

Forma A.1.- Definición breve del proyecto

Descripción del proyecto
Breve descripción del proyecto]
Datos generales del clientes]
Alcances del proyecto]
Tiempo empleado en las reuniones diarias]
Número de semanas que tendrá un entregable]
Funcionalidades básicas del proyecto]

Forma A.2.- Definición de requerimientos no funcionales

Requerimientos no funcionales	
Identificador	Descripción
F1	Breve descripción de requerimientos no funcionales como desempeño, velocidad, sistema operativo, versión]
F2	

Forma A.3. Definición de arquitectura

Definición de arquitectura	
Identificador	Descripción
A1	Definir la arquitectura a usar dentro del proyecto, n capas, MVC, SOA, entre otros]
A2	

Forma A.4.- Entregables del proyecto.

Entregables del proyecto		
Identificador	Fecha	Descripción
R1	[Fecha de compromiso para el entregable]	Definición breve del entregable]
R2		
R3		

Forma A.5.- Integrantes del equipo

Equipo			
Identificador	Nombre	ROL	e-mail
E1	[Nombre del integrante]	[Rol(es) que desempeñara de forma preponderante]	[correo electrónico]
E2			

Forma A.6.- Planeación de historias de usuario (Project Charter)

Identificador	User Stories	Prioridad (1-3)	Release	Resp.	Tiempo (días)	Fecha inicio	Fecha fin	Valor Plan	Valor ganado
	Nombre de la historia de usuario]	Prioridad de 1 a 3]	Numero de release en que se asignara esta historia de acuerdo a su prioridad]	Desarrollador responsable de esta historia]	Numero de días en que el desarrollador estima se realizara esta historia]	Fecha inicio de la historia]	Fecha final de la historia]	Valor porcentual planeado en base al 100% que esta historia aporta al proyecto]	Valor actual una vez desarrollada esta historia]
								Total Valor ganado del Proyecto	

								Porcentaje de avance del proyecto	%
--	--	--	--	--	--	--	--	-----------------------------------	---

Forma A.7.- Historia de usuario

Historia de usuario ID: [Ejemplo: reporte]		Nombre: [Nombre de la Historia de usuarios, ejemplo: Generar el reporte de nómina]				
Descripción: Ejemplo: Generar el reporte en cristal reports via web del 2% sobre nómina que cumpla con la legislación actual]						
Requerimientos funcionales (investigar cómo se llama): • [Ejemplo: Debe ser exportado a PDF] • [Ejemplo: Reqs. De ley, imss]						
Notas adicionales Aquí van notas adicionales, bugs, links, entre otros]						
Tasks	tareas que se requieren para cumplir con esta historia de usuario]					
ID	Area	Prioridad (1-3)	Fecha inicio	Tiempo planeado (horas)	Valor plan	Valor ganado
Interfazreporte	Ejemplo: Construir la interfaz para mandar a procesar al reporte]	Prioridad de 1 a 3]	Fecha inicio de la tarea]	Tiempo planeado en horas]	Valor porcentual planeado en base al 100% que esta tarea aporta a la historia]	Valor actual una vez desarrollada esta tarea]
Reporteconcentrado	Ejemplo: Contruir el reporte concentrado del 2% sobre nómina]	Prioridad de 1 a 3]	Fecha inicio de la tarea]	Tiempo planeado en horas]		0
Reportedetallado	Ejemplo: Construir el reporte detallado del 2% sobre nómina]	Prioridad de 1 a 3]	Fecha inicio de la tarea]	Tiempo planeado en horas]		0

Forma A.8.- Definición de pruebas unitarias para cada Tarea

Task ID: [Ejemplo: reportedetallado]	Nombre: [Construir el reporte detallado del 2% sobre nómina]	
Nombre Prueba Unitaria: [Ejemplo: testImpresionReporte]		Descripción: [Ejemplo: impresión del reporte detallado via web]
[TestMethod] void testImpresionReporte() { Arranging: <i>ReporteDetallado rptDetallado= new ReporteDetallado();</i> Act [Aquí van las operaciones] rptDetallado.procesarReporte(100); Assert: [Aquí va el assert que va a validar la prueba] <i>Assert.IsNotNull (rptDetallado)</i> ?		

Forma A.10.- Plan de comunicación

Información	Medio	Formas de Comunicación	Quien lo Genera	Quien lo Recibe
[Información a compartir, Ejemplo: Requisitos]	[Medio en que proporcionara: Impreso /Electrónico]	[Forma en que se realizara la comunicación: ejemplo: interpersonal, teléfono, correo electrónico]	[Integrante(s) que genera la información]	[Integrante(s) que reciben la información]

Forma A.11.- Forma de mejora de procesos

Propósitos de mejora de procesos	
Identificador	Descripción
F1	[Breve descripción de proposito de mejora detectado en alguna etapa del proyecto]
F2	

Forma A.9.- Plan de Riesgos

Descripción	Impacto	Probabilidad	Estrategia de Mitigación	Realizado Por	Fecha de levantamiento	Revisado Por	Fecha de Revisión
Breve descripción del riesgo]	[Impacto en caso que suceda: Alto, Mediano, Bajo]	[Probabilidad de suceda: Alto, Mediano, Bajo]	[Medidas que se pueden tomar para contrarrestar o minimizar los riesgos]	[Nombre del integrante que detecto el riesgo]	[Fecha en que se detecto el riesgo]	[Nombre del integrante que reviso el riesgo]	[Fecha de revisión del riesgo]

7. Apéndice B. Estándares del modelo DEPYM

Forma B.1.- Estándar de codificación

Propósito	Describir el propósito del estándar de codificación, ambiente de aplicación, vigencia.
Comentarios al inicio de cada clase	[Breve descripción del propósito del programa o clase]
Declaración de clases e interfaces	[Describir las declaración de las partes de una clase o interfaz y el orden en que deben aparecer]
Indentación	[El número de espacios que deberá tener el tabulador]
Longitud de las líneas	[Determinar el número de caracteres máximo que podrá contener una línea]
Comentarios	[Determinar los estilos de comentarios que deben tener los programas]
Convención de nombres	[Definir las convenciones de nombrado de los siguientes componentes de una programa:] Clases : [Estándar para nombrado de clases] Interfaces: [Estándar para nombrado de interfaces] Variables: [Estándar para nombrado de variables] Métodos: [Estándar para nombrado de métodos] Constantes:[Estándar para nombrado de constantes]

Estándar B.2.- Administración de configuración del software

Concepto	Descripción
Introducción	Breve descripción del propósito del estándar de configuración de software]
Acrónimos	Lista de acrónimos usados en el proyecto]
Estructura de Carpetas	Definición de la estructura de carpetas en que se organizará el proyecto]
Formatos de fecha y hora	Formato de fecha y hora que se usara en cada documento del proyecto]
Nombrado de archivos y carpetas	Estándar para el nombrado de archivos y carpetas]
Manejo de versiones	Estándar para el nombrado de versiones de programas y documentos.]