UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES
"A C A T L Á N"

GUÍA METODOLÓGICA PARA LA MEJORA DEL PROCESO DE PRODUCCIÓN DE SOFTWARE EN MÉXICO

SEMINARIO TALLER EXTRACURRICULAR

QUE PARA OBTENER EL TÍTULO DE

LICENCIADO EN MATEMÁTICAS APLICADAS Y COMPUTACIÓN

P R E S E N T A

ROGELIO SÁNCHEZ SANTIAGO

ASESOR: LIC. JUAN TORRES LOVERA

JUNIO, 2006

Dedico este trabajo a mis padres, ya que gracias a ellos he aprendido lo mas valioso de la vida, la honestidad.

A mis hermanos por el apoyo incondicional que me han confiado a lo largo de mi vida, Marcela, Jesús, Rosa, Virginia y Adela.

A mí amada esposa Laura y a mi hija Diana, por su apoyo, confianza y paciencia durante este periodo en el cual compartimos menos tiempo. Por que ellas son la motivación para seguir adelante.

A mis amigos, compañeros de seminario y de trabajo, con los cuales compartí la ilusión de alcanzar esta importante meta, y me brindaron su apoyo para llevarla a cabo.

A mis profesores, por que aquí se ve culminado su esfuerzo de enseñanza impartido a lo largo de muchos años.

ÍNDICE

INTRODUCCIÓN

En la actualidad, la importancia que la producción de software toma en el ámbito mundial se fortalece, cada vez más organizaciones en todo el mundo invierten en la sistematización y/o automatización de sus procesos a través de la implantación de un producto de software. Al mismo tiempo, los procesos de generación de software se diversifican orientando sus soluciones a plataformas[1] y problemas bien definidos. Esto da la pauta para considerar al mercado de software como uno de los más importantes y de gran potencial de crecimiento a nivel mundial. En consecuencia, las empresas y países que logren incorporarse a este mercado experimentaran un desarrollo y crecimiento sostenido, y por ende, mayor nivel de vida para sus empleados.

Desafortunadamente, la industria mexicana dedicada a la producción de software, adolece de varios males que impiden que México se consolide como uno de los principales productores de software en el mundo y que por lo tanto, no sea participe de los beneficios que ofrece este mercado.

Además de las deficiencias en esta industria, el proceso de producción de software es riesgoso y difícil de controlar, y si aunado a esto no se utiliza o elige la metodología más adecuada, el resultado son productos nunca terminados o de baja calidad. Es por esto que el presente trabajo de investigación, se enfoca a atacar este aspecto tan importante con la intención de presentar una propuesta actualizada que ayude a mejorar dicho proceso. Este trabajo tiene como objetivo proporcionar una guía que indique los pasos o actividades básicas a seguir durante el proceso de producción de un software, eligiendo las mejores prácticas de las metodologías prevalecientes hasta el momento: *Rational Unified Process (RUP), Extreme Programming (XP) y Microsoft Solution Framework (MSF)*. Y cuya hipótesis tentativa es: El análisis y la evaluación de dichas metodologías permitirá generar la guía metodológica que considere las mejores prácticas del proceso de producción de software, permitiendo la mejora del mismo.

La metodología que se sigue en el presente trabajo de investigación, consiste en el análisis del proceso de producción de un software a través de las situaciones pasadas y presentes, a fin de dar

[1] Se refiere a las plataformas informáticas. En sistemas operativos: Linux, Unix, MS-DOS, Windows, etc. En lenguajes de programación: Java, C, Microsoft, etc.

una propuesta de mejora a este proceso en un futuro. A continuación se describen los pasos seguidos en esta investigación:

1. Estudio de las situaciones pasadas, presentes y tendencias de la industria dedicada a la producción de software en México, aciertos, errores, oportunidades y deficiencias.
2. Estudio de los factores que impactan directamente en el crecimiento de la industria (Infraestructura, Recursos humanos, Educación). Se pone especial atención en los recursos humanos como materia prima para la industria.
3. Análisis de las metodologías de producción de software y su importancia como parte fundamental del proceso. Evaluación de las metodologías actuales e Identificación de sus propuestas (practicas) para resolver los problemas más comunes que enfrenta el proceso de producción de software.
4. Propuesta de mejora para el proceso de producción de un software. Con base en el análisis de las metodologías, se complementan las prácticas de las mismas a fin de cubrir los problemas más frecuentes durante el ciclo de vida del proceso.
5. Evaluación de la propuesta con respecto a cada una de las metodologías estudiadas. Ventajas que proporciona el uso de la propuesta del presente trabajo de investigación.

Los pasos descritos anteriormente son cubiertos a lo largo de los tres capítulos que componen esta investigación.

Los principales factores que impactan para el estancamiento de la industria de la producción de software en México se identifican en el primer capítulo de este trabajo de investigación. En este mismo capítulo, se describen las estrategias a seguir en el plan nacional para el desarrollo de la industria del software en México, que el gobierno federal emprendió en el año 2002 y que ataca la mayoría de estos problemas. Además, se describe una reseña histórica sobre la producción de software en México, su importancia, problemática, la infraestructura con la que se cuenta actualmente y sus tendencias. Se trata de dar a conocer cuales son las deficiencias que enfrenta nuestro país en este proceso, enfocándose particularmente a la revisión del uso de metodologías.

Una vez superados los factores anteriores, el principal problema de la producción de software se convierte en la elección y el uso de la metodología mas adecuada. Con el fin de proporcionar una alternativa para enfrentar este problema, en el presente trabajo de investigación se estudian las

siguientes metodologías: Rational Unified Process (RUP), Extreme Programming (XP) y Microsoft Solution Framework (MSF), Scrum, Crystal Family, Open Source y Feature Driven Development.

El análisis comparativo de estas metodologías, que permitirá reconocer sus mejores prácticas y la forma en que son aplicadas, se muestra en el segundo capítulo. Así mismo, se describen las principales variables que inciden en el éxito o fracaso de un proyecto de producción de software. Este apartado arroja como resultado una lista de prácticas que sirven de base para el diseño de la guía metodológica, que se define en el siguiente capítulo.

Por último, en el tercer capítulo, se describe la propuesta metodológica que recopilara las mejores practicas de las metodologías analizadas en el capítulo dos, conformando una guía que indique los pasos a seguir para afrontar con eficiencia la producción de software.

El principal problema enfrentado durante el desarrollo del presente trabajo, fue la enorme cantidad de información existente sobre el tema, lo que obstaculizó el proceso y en consecuencia requirió de un mayor esfuerzo para el análisis y diseño de la propuesta.

Se espera que con esta guía, las empresas y personas dedicadas a la producción de software, cuenten con una herramienta actualizada y efectiva que les permita llevar con mayor control el proceso de producción de software.

Como parte innovadora de este trabajo, es importante mencionar que a pesar de los diferentes análisis de las metodologías aquí en estudio, nunca se ha documentado una propuesta que en lugar de contraponer las diferencias entre ellas, permita su complementación con el fin de la mejora del proceso de producción de software.

Como parte fundamental de este trabajo, cabe resaltar que a pesar de los avances tecnológicos y la evolución de los procesos para la producción de software, de los cuales se ha sido testigo en casi ocho años de experiencia laboral, se puede afirmar que la constante más importante a lo largo de este tiempo, es sin duda, la planeación. Algo a lo que comúnmente no se le da la importancia necesaria, pero que es parte fundamental para el control de cualquier proceso en una organización. Para la realización de la presente investigación se usaron los diferentes elementos

que componen la teoría de la planeacion, teniendo como base principal el trabajo de Russell Ackoff como un pensador prominente en el campo de la planeación.

Finalmente en la evaluación de la propuesta se realizo un análisis FODA para verificar las ventajas y oportunidades del proceso propuesto.

CAPÍTULO 1

LA PRODUCCIÓN DE SOFTWARE EN MÉXICO

En el presente capítulo se muestra un contexto general sobre el desarrollo de la industria del software en México, los elementos que lo componen y la importancia que juegan las metodologías como parte de su proceso. Esta dividido en tres apartados principales, en el primero se describen los antecedentes, la situación actual y las tendencias sobre la industria. Muestra una visión en cifras de lo que hasta ahora ha sido y sus posibles tendencias para el futuro. Aborda en forma específica el contenido del plan nacional para el desarrollo de la industria del software en México, implementado por el gobierno federal a partir del año 2002. En el siguiente apartado se describen los elementos más importantes para la producción de un software, tales como: La infraestructura, los recursos humanos y las metodologías. Se pretende mostrar la infraestructura con la que cuenta actualmente México en apoyo a esta industria. Así mismo, la importancia de los recursos humanos, su nivel educativo y capacitación. Se presenta una comparación cuantitativa de las personas que cursan, egresan y se titulan en las carreras relacionadas a la tecnología informática en México con respecto a Estados Unidos de América. Por ultimo, en el tercer apartado, se explica de la importancia de las metodologías en el proceso de producción de software, las ventajas y desventajas de su uso y elección.

1.1 El desarrollo de la producción de software en México

El proceso de producción de software puede definirse como un conjunto de herramientas, métodos y prácticas que se emplean para producir software. Como cualquier otra organización, las dedicadas a la producción de software mantienen entre sus principales fines, la producción de software de acuerdo con la planificación inicial realizada, además de una constante mejora con el fin de lograr los tres objetivos principales de cualquier proceso de producción: alta calidad y bajo costo, en el mínimo tiempo.

1.1.1 Antecedentes.

En los últimos años se ha insistido en la conveniencia de establecer una industria de producción de software en México capaz de competir en los mercados internacionales. Lo que ha desembocado en la existencia de diversas propuestas plasmadas en planes gubernamentales y de cámaras industriales, los cuales definen las acciones a seguir para promover esta importantísima industria en México.

Se conoce y continuamente se menciona el hecho de que en la India se reciben más divisas por producción de software que por cualquier otra fuente; se percibe a Irlanda como un país que ha apostado por esta industria, y se comenta de la importancia que tiene Brasil para la misma en Sudamérica.

Para darnos una idea sobre el desarrollo de la producción de software en los últimos años en el caso mexicano, la Asociación Mexicana de la Industria de Tecnologías de Información (AMITI[2]) en su sitio Web da a conocer algunos datos interesantes[3]:

- ✓ "El segmento de desarrollo de software ha crecido, desde 1995, a tazas mayores al 30%, tanto en número de empresas que integran el mercado como en valor.

- ✓ Fuga de cerebros propiciada por la diferencia entre los salarios de los países desarrollados y los que están en vías de desarrollo, que trae aparejada una reducción de la capacidad técnica de las empresas.

- ✓ Dentro de la actividad de Servicios Profesionales, el desarrollo de software, en el año 2001, tuvo un valor de $380 millones de dólares y equivalió al 8.5% del valor del mercado de Servicios Profesionales, que ascendió a $2,056 millones de dólares.

- ✓ La actividad de desarrollo de software a la medida creció a una tasa de crecimiento compuesto anual de 34% (1996-2001) para este periodo.

- ✓ El mercado de TI en México para el año 2001 (Hardware, Software empaquetado, Servicios profesionales – software a la medida - y Servicios de soporte) fue de alrededor de $9,264 millones de dólares, en el que el valor del segmento de servicios profesionales (desarrollo

[2] Fundada en 1985 como la Asociación Nacional de la Industria de Programas de Cómputo (ANIPCO), AMITI, A.C. se constituye en 1997 con el propósito de participar en el desarrollo tecnológico del país para incorporar a los sectores de Hardware, Software, Integradores, Consultores, Proveedores de Servicios y Canal de Distribución siendo un claro habilitador de la competitividad en México. Actualmente, la AMITI cuenta con 260 socios en 19 estados del país, además de convenios de colaboración y relaciones con universidades, embajadas, dependencias gubernamentales, asociaciones y cámaras para promover el desarrollo de la industria de Tecnologías de Información.

[3] http://www.amiti.org.mx

de software a la medida) fue de $2,534 millones de dólares y represento el 27%"[4].

Esta información concuerda con los datos revelados por el INEGI, donde se mencionan una serie de indicadores que muestran la situación de la informática en México en los años más recientes.

- ✓ "El Producto Interno Bruto Informático creció 27.2% en términos reales en el año 2000 respecto a 1999, participando con 3.5% del total de la economía. El sector más dinámico fue el de las telecomunicaciones, las cuales aumentaron 28.4%, seguido por el equipo y periféricos para procesamiento informático que lo hizo en 22.9 %.

- ✓ Las exportaciones totales de equipo informático sumaron 8,141 Millones de Dólares (MD) en el año 2000, monto que significó 1,742MD más que el año anterior, lo cual es muestra de la dinámica que presenta este sector. Las importaciones fueron de 8,258MD dando como resultado una reducción muy significativa en el déficit comercial de bienes informáticos.

- ✓ Prácticamente todas las secretarías de Estado cuentan con una página en Internet donde proporcionan información sobre los trámites que ofrecen a la ciudadanía.

- ✓ Únicamente 9.3% de las viviendas cuentan con una computadora en su hogar. En el Distrito Federal 21.6% de las viviendas poseen computadora, en el extremo opuesto 7 entidades presentan porcentajes inferiores a 5%"[5].

Los indicadores económicos divididos en los cuatro aspectos fundamentales del desarrollo informático nacional, según los resultados de las encuestas y censos del INEGI, para el año 2000:

- ✓ Economía Digital
  - o Crecimiento del PIB en 27.2% con respecto a 1999 (4 veces más que la economía en su conjunto). Participación del sector informático en 3.5% del total de la economía. En su interior, telecomunicaciones creció 28.4%, equipo y periféricos

[4] Asociación Mexicana de la Industria de Tecnologías de Información, AC. AMITI. Noticias de la industria. http://www.amiti.org.mx.
[5] Instituto Nacional de Estadística Geografía e Informática. INEGI. Sala de Prensa. Comunicados de Prensa. Comunicados anteriores. Por tema. Situación de la Informática en México. http://www.inegi.gob.mx

para procesamiento informático 22.9% y Servicios profesionales en informática 4.9%. La industria manufacturera informática se concentra en seis entidades (Baja California, Chihuahua, Nuevo León, Sonora, Tamaulipas y Jalisco), absorben el 69% de los establecimientos y el 94% del personal ocupado. Las exportaciones totales en equipo informático sumaron 8,141 millones de dólares, cifra mayor a la de 1999 que fue de 6,399 millones de dólares. Las importaciones en equipo informático ascendieron a 8,258 millones de dólares. El déficit comercial de bienes informáticos en (-) 177 millones de dólares. Las empresas dedicadas al servicio de "análisis de sistemas y procesamiento informático", se encuentran en su mayoría en el Distrito Federal y Nuevo León, ya que en estas dos entidades se ubican casi 5 de cada 10 empresas del ramo.

- ✓ Infraestructura Tecnológica
  - o Estimación preliminar de la existen de 65 equipos por cada mil habitantes (Estados Unidos y Canadá con 500 y 260 computadoras por cada mil habitantes). Densidad telefónica de 11.2% (número de líneas telefónicas por cada 100 habitantes).

- ✓ Gobierno en Línea
  - o Todas las secretarías de Estado cuentan con una página en Internet donde proporcionan información sobre su sector y los servicios que ofrecen a la ciudadanía. De las entidades paraestatales del gobierno, 120 cuentan también con un sitio en Internet, donde presentan información correspondiente a sus atribuciones y ámbitos de competencia.

- ✓ Sociedad de la Información
  - o En el Distrito Federal 21.6% de las viviendas poseen computadora, y en las entidades de Baja California, Sonora, Chihuahua, Nuevo León y Jalisco alrededor de 15% disponen de esta tecnología, en el extremo opuesto siete entidades presentan porcentajes inferiores a 5 por ciento. El 81% de los hogares que disponen de computadora en el país, el jefe de familia tiene un nivel académico de preparatoria o superior

1.1.2 Situación actual.

Según la AMITI, actualmente México cuanta con las condiciones idóneas para explotar la tecnología informática, dentro de la cual se encuentra la producción de software[6]. Algunos de los puntos que se subrayan para determinar esta situación se listan a continuación:

- ✓ Ubicación privilegiada para atender al principal cliente potencial, EE.UU. y el beneficio del idioma, para atender al mercado hispánico en los EE.UU. y al latinoamericano, el de mayor potencial de crecimiento.
- ✓ Incremento en el número de profesionales en disciplinas afines a la informática y con niveles de sueldo competitivos respecto al mercado internacional.
- ✓ México no tiene restricciones para la importación de tecnología, lo que facilita su transferencia.
- ✓ Sólida estructura de telecomunicaciones.
- ✓ Derechos de propiedad intelectual debidamente protegidos por las leyes.
- ✓ Arancel cero para la prestación de servicios de tecnología de la información con los EE.UU.
- ✓ Reconocida creatividad e ingenio.
- ✓ Adopción rápida de nuevas tecnologías.

En consecuencia con esto, en el año 2002 la secretaria de economía puso en marcha un programa para el desarrollo de la industria del software en México.

Entre los objetivos planteados se encuentran:

- ✓ Alcanzar el promedio mundial de gasto en tecnologías de información.
- ✓ Lograr una producción de software por 5,000 millones de dólares anuales.
- ✓ Convertir a México en el líder latinoamericano de soporte y desarrollo de servicios basados en TI.

Las estrategias propuestas por el gobierno federal para lograr estos objetivos son:

1. Promover las exportaciones y la atracción de inversiones.
    - ✓ Atracción de inversión extranjera para desarrollo en México.
        - ➢ Portal para promoción de México como productor de software.

[6] Asociación Mexicana de la Industria de Tecnologías de Información, AC. Op. cit.

- Estudio para detectar oportunidades en mercado internacional y capacidad de la oferta de la industria mexicana de software.

✓ Difusión internacional de las capacidades de las empresas mexicanas y de casos de éxito.

✓ Investigación de nichos en mercado externo y encuentros empresariales para subcontratación.

✓ Repatriación de talento orientado a formar empresas e integración de una red de contactos para atraer demanda a empresas mexicanas.

2. Educación y formación de personal competente en el desarrollo de software.

✓ Adecuación y mejoramiento de los planes y programas de estudio.

- Propuesta de adecuación de planes de estudio en los niveles técnico superior y educación superior.
- Proyecto de formación de capital humano en software y electrónica de alta tecnología.

✓ Fomento al desarrollo de sistemas de formación y certificación de profesores, mediante el otorgamiento de becas e incentivos.

✓ Fortalecimiento de la vinculación entre instituciones educativas y empresas de software.

3. Contar con un marco legal promotor de la industria.

✓ Comercio electrónico.

- Firma y comprobantes fiscales electrónicos.
- Comprobantes fiscales electrónicos.

✓ Diseño de un marco fiscal acorde a la naturaleza de la industria.

- Iniciativa de ley sobre firma digital.

✓ Adecuación de la normatividad en el orden local.

- NOM - Conservación de Mensajes de Datos
- Códigos civiles reconocimiento de contratos electrónicos.

4. Desarrollar el mercado interno.

✓ Creación de la fundación México digital.

- Proyectos de integración digital en las cadenas de abarrotes, alimentos procesados, industria maquiladora y hotelería.
- Diagnóstico, rediseño de procesos y plan de capacitación.
- Solución tecnológica, medición del impacto y documentación del caso.

5. Fortalecer a la industria local.

- ✓ Metodología para la incubación de empresas de software.
  - ➢ Gobiernos estatales, Universidades y Gobierno Federal.
  - ➢ Habilitación y operación de incubadoras.
  - ➢ Nuevas empresas de software y aumento en la capacidad de las existentes.
- ✓ Promoción de la reserva nacional para las compras gubernamentales de software y servicios relacionados.
  - ➢ IMSS, ISSSTE, e-México

6. Alcanzar niveles internacionales en capacidad de procesos.

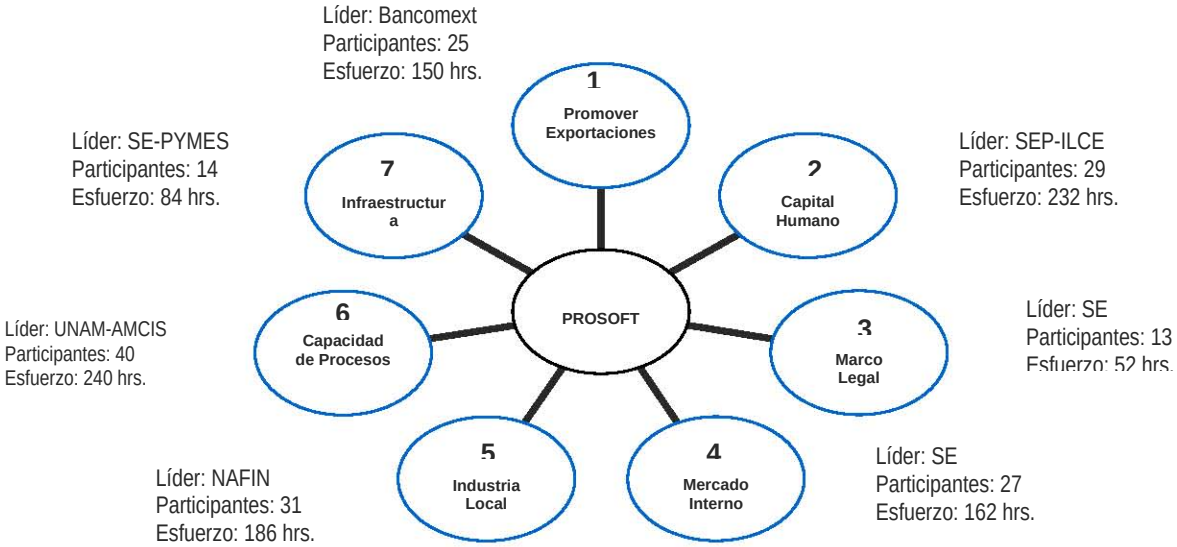
- ✓ Modelo de procesos para la industria de software. (MOPROSOFT)
  - ➢ Método de evaluación del MOPROSOFT.
  - ➢ NMX de calidad para el desarrollo de software.
  - ➢ Proyectos piloto de implantación del MOPROSOFT.
  - ➢ Metodología para la difusión y esquema de certificación de la NMX.
- ✓ Modelo de gestión tecnológica para la industria de software.
- ✓ Creación de la categoría de tecnologías de información en el Premio Nacional de Tecnología.
- ✓ Difusión y cursos de capacitación en 20 entidades federativas.

7. Promover la construcción de infraestructura física y de telecomunicaciones.

Modelo de gestión tecnológica para la industria de software.

- ✓ Estudio para identificar el potencial de las entidades federativas para desarrollar agrupamientos de industria de software (clusters)
  - ➢ Promoción de la figura de empresa integradora.
  - ➢ Inversión conjunta en proyectos productivos.
  - ➢ Fortalecimiento de la oferta local.
  - ➢ Impulso a la formación de parques tecnológicos.

Diagrama de subcomités por estrategia[7]. (Figura 1.1)



Lo revisado anteriormente, son algunos de los puntos más importantes que se están llevando a cabo como parte de los esfuerzos que organizaciones públicas y privadas plantean para promover la TI. Por otro lado, existen tendencias más de la índole técnica que sugieren datos muy interesantes y que a continuación se describen.

- ✓ El segmento que más ha crecido en el mercado de producción de software es el desarrollo en Web, mientras que el desarrollo de software a la medida esta disminuyendo en importancia y frecuencia, ya solo se da como desarrollo en nichos específicos como el Web, los PDAs, la tecnología móvil y la integración de aplicaciones ERPs.

- ✓ De acuerdo con estos resultados las ERPs son las aplicaciones que más se han implementado en el segmento de negocios, en general las aplicaciones relacionadas con la planeación de recursos internos.

- ✓ Es importante destacar que los establecimientos o empresas dedicados al servicio de "análisis de sistemas y procesamiento informático", se encuentran en su mayoría en el

[7] Software.net.mx. PROSOFT. Avances. Avances 2003. http://www.software.net.mx

Distrito Federal y Nuevo León, ya que en estas dos entidades se ubican casi 5 de cada 10 empresas del ramo.

Para dar una mejor idea sobre la situación actual de la industria informática en México, a continuación se muestra la tabla 1.1, donde se puede observar la relevancia que tiene esta industria dentro de la economía mexicana.

Producto Interno Bruto Informático, en miles de pesos a precios de 1993[8] (Tabla 1.1).

(En miles de pesos a precios de 1993)

Año	PIB Total, a Precios de Mercado	PIB Informático				Participación de PIB Informático %	Variación anual del PIB Informático %
		Total	5402 Equipo y Periféricos para Procesamiento Informático	6511 Telecomunicaciones	6821 Servicios Profesionales en Informática y Actividades Conexas		
1994	1 311 661 116	24 614 015	1 350 516	22 485 456	778 043	1.9	NA
1995	1 230 771 052	26 030 429	1 518 469	23 965 631	546 329	2.1	5.8
1996	1 294 196 562	30 238 292	2 457 776	27 152 662	627 854	2.3	16.2
1997	1 381 839 196	33 816 005	3 924 462	29 137 472	754 071	2.4	11.8
1998	1 451 350 909	38 043 011	4 827 024	32 367 204	848 783	2.6	12.5
1999	1 503 930 030	43 965 538	5 054 342	37 977 112	934 084	2.9	15.6
2000 R	1 602 640 366	50 703 467	6 325 431	43 357 023	1 021 013	3.2	15.3
2001 P	1 602 711 216	56 701 330	5 833 784	49 877 756	989 790	3.5	11.8
2002	1 613 206 366	59 647 799	4 986 229	53 725 870	935 700	3.7	5.2
2003	1 633 075 722	64 067 428	3 234 887	59 896 334	936 207	3.9	7.4
2004 a	1 679 150 127	75 246 937	3 442 952	70 846 973	957 012	4.5	17.4

NOTA: Ramas de actividad del clasificador del SCNM.
a Las cifras corresponden al cálculo del PIB Trimestral para el tercer trimestre.
NA No aplicable.
R Cifras revisadas a partir de la fecha en que se indica.
P Cifras preliminares.
FUENTE: **INEGI.** Sistema de Cuentas Nacionales de México. Cuentas de Bienes y Servicios 1988-1999, Tomo II.
INEGI. Sistema de Cuentas Nacionales de México. Cuentas de Bienes y Servicios 1995-2000, Tomo II.
INEGI. Sistema de Cuentas Nacionales de México. Cuentas de Bienes y Servicios 1996-2001, Tomo II.
INEGI. Sistema de Cuentas Nacionales de México. Cuentas de Bienes y Servicios 1997-2002, Tomo II.
INEGI. Sistema de Cuentas Nacionales de México. Producto Interno Bruto Trimestral 2002-2004.

En la tabla 1.1, se puede observar que la participación del PIB informático con respecto al PIB nacional, se ha incrementado en forma constante por los últimos once años. Los incrementos anuales reflejan un crecimiento mayor al promedio nacional. Aunque los rubros principales de esta industria no han crecido por igual, en su conjunto se ve un buen desempeño. El rubro con mayor

[8] Instituto Nacional de Estadística Geografía e Informática. INEGI. Información estadística. Estadísticas por tema. Ciencia y tecnología. Indicadores sobre tecnología de la información y comunicaciones. Producto interno bruto total e informático, 1994 a 2004. http://www.inegi.gob.mx

crecimiento en la actualidad es el referente a *telecomunicaciones*, mientras que *Equipo y periféricos para procesamiento informático* es el menos favorecido. Por su parte *Servicios profesionales en informática y actividades conexas* se mantuvo con altibajos hasta el año 2000 donde alcanzo su mayor crecimiento de los últimos años, y a partir del 2001 este decreció e incremento su crecimiento en forma poco significativa. En resumen, la tabla 1.1 muestra de forma general el desempeño de la industria de la informática en México y el crecimiento sostenido de la misma a través de los últimos años.

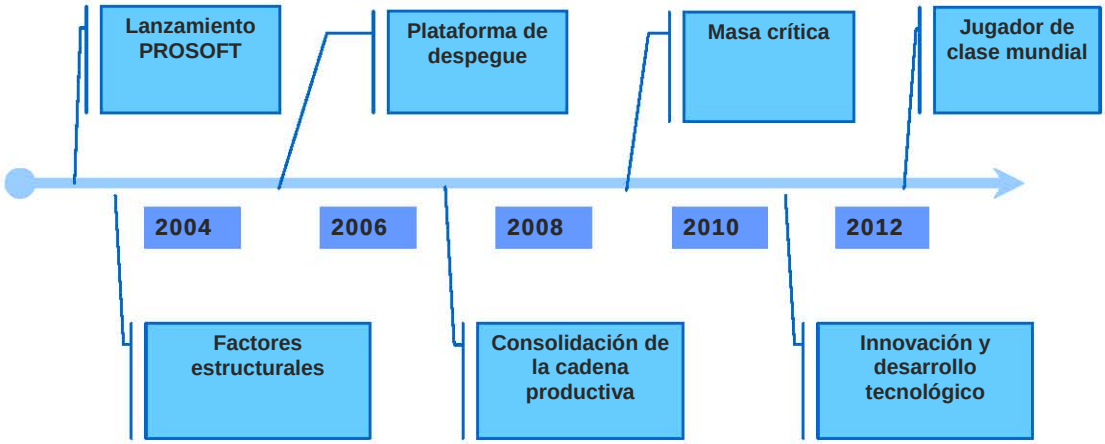
En el documento "Programa para el desarrollo de la industria del software (PROSOFT), versión 1.3", que difunde la secretaria de economía en el sitio web: http://www.software.net.mx, afirma que "el mercado de las tecnologías de información y comunicación (TIC) represento el 6.6% del valor de la producción económica mundial en la década de los noventas. Y que durante esta misma, la mayor parte de los países, aún los que enfrentaron crisis financieras y recesiones económicas, incrementaron su gasto en tecnologías de información y comunicación". Además, sugiere que gran parte de producción de software a nivel mundial se esta realizando en países en vías de desarrollo, donde la India e Irlanda representan los casos más notables de éxito para esta industria. La experiencia de estos dos países, nos indica que mediante una buena política gubernamental de apoyo a la industria del TIC, México puede consolidarse como un país productor y exportador de software, y por ende, recibir los grandes beneficios de este mercado.

1.1.3 Tendencias.

Una vez revisada la situación actual sobre la TI y en particular la producción de software en México, se observa que el momento histórico, con las condiciones idóneas para proyectar a esta industria son ahora. Dependiendo de la correcta aplicación de los planes gubernamentales, y de la promoción y fomento de la industria (por parte de asociaciones y empresas del sector), las expectativas de crecimiento a mediano y corto plazo son muy favorables.

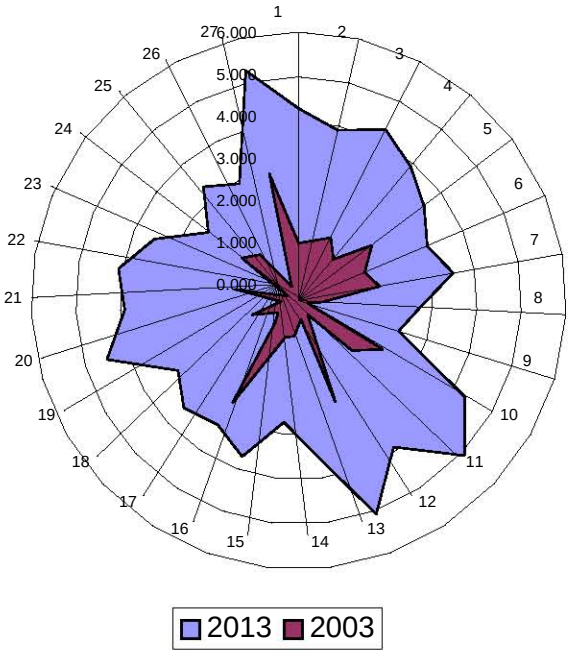
A continuación se presentan dos diagramas que representan las tendencias a mediano plazo para la industria, conforme se va aplicando el programa para el desarrollo de la industria del software por la secretaria de economía.

Diagrama visión a 10 años TI en México[9]. (Figura 1.2)



La figura 1.2 muestra las metas en los intervalos de tiempo que pretende el gobierno federal con la implementación del programa PROSOFT. El objetivo del programa es que México se convierta en un jugador de clase mundial en la industria del software para el año 2013.

Avance de paquetes PROSOFT 2003[10]. (Grafica 1.1)



- Porcentaje de avance del PROSOFT al término del 2003: 18%
- Total de paquetes: 27
- Paquetes con avance: 21

[9] Software.net.mx. PROSOFT. Avances. Avances 2003. http://www.software.net.mx
[10] Idem.

1.2 Elementos de la producción de software

1.2.1 Infraestructura

Cuando se habla del término infraestructura nos referimos a los elementos que darán el soporte necesario para la consecuencia de otra actividad o proceso. Por ejemplo, para tener una buena comercialización de productos dentro de un país, este debe de contar con una infraestructura de comunicación para el transporte, es decir, debe contar con las carreteras, puertos, aeropuertos y maquinaria necesarios para la distribución de estos productos.

Para la producción de software, de la misma forma que para la distribución de productos, es necesario contar con ciertos elementos que ayudaran a llevar a cabo esta actividad. Algunos de los elementos más importantes que sirven de apoyo para la producción de software en un país se listan a continuación:

- ✓ Educación.
- ✓ Tecnología en telecomunicaciones y cómputo.
- ✓ Marco Jurídico.

En este apartado se dará a la tarea de mostrar los más recientes indicadores sobre la infraestructura en tecnología de telecomunicaciones y computo. El tema de educación se tratara a fondo en el siguiente apartado titulado *recursos humanos*. Por su parte, el marco jurídico queda contenido en las leyes gubernamentales que se están legislando a fin de darle mayor sustento jurídico a los empresarios e inversionistas del sector.

A continuación se listan datos relevantes sobre el crecimiento de la infraestructura tecnología en cómputo y telecomunicaciones en México (fuente INEGI):

- ✓ El Producto Interno Bruto Informático creció 17.4% (Las cifras corresponden al cálculo del PIB Trimestral para el tercer trimestre) en el año 2004 con respecto al 2003. La participación del sector informático es del 4.5% del total de la economía.

- ✓ A su interior, el sector de mayor crecimiento fue el de las telecomunicaciones, con un

incremento del 18.3% durante el año 2004, seguido del equipo y periféricos para procesamiento informático que lo hizo en 6.4%. El sector Servicios profesionales en informática creció 2.2%.

- ✓ No obstante esta dinámica, el desarrollo de la industria manufacturera informática se encuentra ampliamente concentrada regionalmente en los estados de la frontera norte y el occidente del país (Baja California, Chihuahua, Nuevo León, Sonora, Tamaulipas y Jalisco), de tal forma que estas 6 entidades en conjunto absorben el 69% de los establecimientos y el 94% del personal ocupado, con una destacada presencia de establecimientos dedicados a la maquila de exportación.

- ✓ Por lo que se refiere a la balanza comercial de equipo informático, las exportaciones totales sumaron 10,941 millones de dólares (MD) el año 2004, cifra que se compara favorablemente con los 10,030 MD que se exportaron en 2003. Por su parte las importaciones ascendieron a 11,333 MD. Esto da como resultado un déficit de (-) 392 MD.

- ✓ En este resultado sobresale el dinamismo de la Industria Maquiladora de equipos y partes informáticas, la cual para el año 2003, participa con el 75.2% del valor total de las exportaciones de equipo informático que realiza México.

- ✓ En cuanto al comercio doméstico al por mayor de equipo informático, se puede apreciar que dicha actividad se concentra en el Distrito Federal, Jalisco y Nuevo León, en donde se realizan el 56%, 19% y 10% de las ventas de computadoras, periféricos y consumibles, respectivamente.

- ✓ Por tipo de bien informático, el 43% de las ventas mayoristas correspondió a computadoras (portátiles, personales y servidores), el 24% a periféricos (monitores, impresoras, scanners, etc.), el 12% a consumibles como papel, tinta y diskettes, y el resto a otros productos de la misma clase.

- ✓ El parque instalado de computadoras personales (PCs) en México muestra un crecimiento constante. Se estima de manera preliminar que existen 65 equipos por cada mil habitantes, cifra muy superior a la que se tenía hace 5 años de 26, pero resulta notablemente inferior a la

que observan nuestros principales socios comerciales como son Estados Unidos y Canadá con 500 y 260 computadoras por cada mil habitantes, en cada uno de ellos.

- ✓ Con relación a la densidad de líneas telefónicas fijas en servicio por entidad federativa para el año 2004 (número de líneas telefónicas por cada 100 habitantes) se tiene: A nivel nacional la densidad telefónica es de 17.1%. Con base en este indicador es posible establecer tres grandes grupos: el primero conformado por 6 entidades cuya densidad telefónica es superior 20% (Baja California, Baja California Sur, Colima, Distrito Federal, Jalisco y Nuevo León); un segundo grupo de 22 estados con índices entre 10 y 20% (Aguascalientes, Campeche, Coahuila, Chihuahua, Durango, Guanajuato, Guerrero, Estado de México, Michoacán, Morelos, Nayarit, Puebla, Querétaro, Quintana Roo, San Luís Potosí, Sinaloa, Sonora, Tamaulipas, Tlaxcala, Veracruz, Yucatán y Zacatecas); el tercer grupo de 4 entidades con índices entre el 5 y 10% (Chiapas, Hidalgo, Oaxaca y Tabasco); A manera de comparación, se puede señalar que para el conjunto de los países miembros de la Organización para la Cooperación y Desarrollo Económico, este indicador es de 50.5%.

- ✓ Con base en las encuestas que el INEGI realiza en la Administración Pública, se informa que, prácticamente todas las secretarías de Estado cuentan con una página en Internet donde proporcionan información sobre su sector y los servicios que ofrecen a la ciudadanía. De las entidades paraestatales del gobierno, 120 cuentan también con un sitio en Internet, donde presentan información correspondiente a sus atribuciones y ámbitos de competencia. Por su parte, en la totalidad de las entidades federativas, los gobiernos estatales brindan información a través de un sitio electrónico sobre las diferentes actividades económicas de su región, su industria, lugares turísticos más importantes, así como sobre la administración estatal.

- ✓ Por su parte, el uso de redes de datos en la Administración Pública, también se ha incrementado. En efecto, actualmente la mayor parte de las dependencias y entidades del Gobierno Federal cuentan con una red para la transmisión de voz y datos (redes institucionales)

- ✓ Es importante señalar que el INEGI cuenta, además de su sitio nacional, con 32 páginas de Internet, una por entidad federativa, donde se puede consultar información sobre aspectos

sociales, demográficos, económicos, geográficos y acerca del desarrollo de las tecnologías de la información en el ámbito local.

- ✓ Para el año 2005 los hogares que cuentan con equipamiento de tecnología de información y comunicaciones se ubican en: con computadora 18.4%, Con conexión a Internet 9%, con televisión 92.7%, con televisión de paga 19.3%, con línea telefónica fija 48.8 y con telefonía celular 42%.

- ✓ Existe un acceso desigual a las tecnologías de la información, mientras en el Distrito Federal 21.6% de las viviendas poseen computadora, y en las de Baja California, Sonora, Chihuahua, Nuevo León y Jalisco alrededor de 15% disponen de esta tecnología, en el extremo opuesto siete entidades presentan porcentajes inferiores a 5% (según el INEGI para el año 2000).

- ✓ Existen grandes diferencias según las características socioeconómicas de los hogares con computadora. De éstos, únicamente el 3.2% reciben ingresos mensuales de hasta cuatro salarios mínimos. El 18.5% se ubica entre más de 4 y 8 salarios mínimos; en cambio de los hogares que disponen de al menos una computadora en la vivienda, el 78.3% percibe ingresos superiores a 8 salarios mínimos (según el INEGI para el año 2002).

- ✓ En función de la edad del jefe(a) de familia también se aprecian diferencias entre los hogares con computadora. En los casos donde el jefe(a) tiene 31 años o más, el 91% de los hogares dispone al menos de una, situación que contrasta significativamente con los hogares encabezados por jóvenes menores de 20 años donde el indicador se ubica por debajo del 1% (según el INEGI para el año 2002).

- ✓ Así mismo, la escolaridad está fuertemente relacionada con la posesión de una computadora. En efecto, el 85.1% de los hogares que disponen de computadora en el país, el jefe de familia tiene un nivel académico de secundaria o superior (según el INEGI para el año 2002).

- ✓ De esta forma podemos apreciar que los niveles de ingreso, la edad del jefe(a) y su nivel de escolaridad, son elementos decisivos para la posesión y el posible aprovechamiento de una computadora en beneficio familiar, y la correlación es positiva.

En suma, con la información señalada acerca de la infraestructura tecnológica, se puede apreciar que existen grandes contrastes en el desarrollo informático en México. Por un lado, el PIB informático crece a una tasa elevada, y al mismo tiempo, se observa una importante concentración geográfica de la industria y una gran disparidad para su acceso relacionada con la situación socioeconómica de la población.

1.2.2 Recursos humanos

Uno de los factores más importantes para el proceso de producción de software, son los recursos humanos, ya que depende de su capacidad y preparación el buen funcionamiento del proceso, y es que a fin de cuentas ellos son los encargados de la planeación, análisis, diseño y construcción del software.

En México, actualmente uno de las fortalezas de esta industria, es la cantidad de desarrolladores que existen, los cuales son muy capaces, pero carecen de certificaciones, de mayor nivel de calidad en las entregas y sobre todo de conocimiento de la administración, coordinación y ejecución de los proyectos.

En el año 2000 existían ciento cincuenta mil estudiantes en programas de licenciatura en Informática y Computación. El dato es importante, ya que la industria de desarrollo de software requiere de mucho capital humano, debido a que su principal insumo son fuertes cantidades de profesionistas.

A continuación se presenta algunas tablas que proporcionan información sobre los profesionistas de las áreas de informática en nuestro país. Esta información fue tomada del documento *"Análisis de la evolución (1971-2001) y tendencias (2002-2005) de los programas de Informática y Computación en México*[11]", el cual esta apoyado en los anuarios generados por la ANUIES (Asociación Nacional de Universidades e Instituciones de Educación Superior). Con base en las Tablas 1.2, 1.3, 1.4 y 1.5 se concluye lo siguiente:

- ✓ El crecimiento en egresados de las profesiones de Informática y Computación en México ha sido constante, de tal forma que para el año 2001 se han logrado acumular 144,212 egresados.

[11] Software.net.mx. Desarrolladores. Software Profesional. Capacitación.Evaluación de programas de computación e informática. Sergio Ellerbracke Román http://www.software.net.mx

Si se compara con los Estados Unidos, entonces México tiene un profesionista en Informática y Computación por cinco profesionistas que tiene Estados Unidos.

- ✓ Estados Unidos tiene más profesionistas con maestría que México con licenciatura. Con 212,584 maestros en *Computer Sciences*, cuenta con un maestro por cada mil cien habitantes, mientras que México sólo tiene 5,194 maestros, uno por cada dieciocho mil ochocientos habitantes.

- ✓ En Estados Unidos han egresado 15,141 doctores, en México 61. Según la doctora Cristina Loyo (2002), en México hay 136 doctores que realizan investigación en Tecnologías de Información, en doce centros de excelencia reconocidos por el CONACYT. El diferencial de doctores seguramente proviene de doctores egresados de postgrados en el extranjero.

- ✓ México cuenta con una capacidad respetable de egresados en redes, telecomunicaciones y desarrollo de software (aunque en procesos de bajo valor agregado). Estos son fundamentalmente a nivel licenciatura, aunque el egreso de los técnicos superiores universitarios ya alcanzó el 10% de la licenciatura.

- ✓ Estos egresados son fundamentalmente de nivel licenciatura, aunque el egreso de los Técnicos Superiores Universitarios ya alcanzó el 10% de estos. Sin embargo, se carece de una masa crítica de profesionistas con maestría, preparados para acometer procesos de alto valor agregado, y se tiene una ausencia casi absoluta de profesionales competentes para la investigación y el desarrollo de nuevas tecnologías.

Tabla 1.2 Primer Ingreso, estudiantes, egresados y titulados de Licenciaturas en Informática y Computación en México y egresados de Computer Science en Estados Unidos, por año y genero, 1969-2001

		México												EU		
		Nuevo Ingreso			Estudiantes			Egresados			Tit			Egreso		
		H	M	Tot	H	M	Tot	H	M	Tot	H	M	Tot	H	M	Tot
Licenciatura	1966													76	13	89
	1967													198	24	222
	1968													404	55	459
	1969													812	121	933
	1970													1345	199	1544
	1971			118			208			6				2064	324	2388
	1972			239			376			10				2941	461	3402
	1973			318			572			50				3665	640	4305
	1974			362			720			57				3977	780	4757
	1975			874			1327			38				4083	956	5039
	1976			807			1825			65				4540	1124	5664
	1977			1067	1921	523	2444			132				4887	1539	6426
	1978			1261	2915	842	3757			121				5360	1864	7224
	1979			1246	3158	904	4062			156				6306	2463	8769
	1980			1773	4428	1362	5790			354				7814	3399	11213
	1981			2133	5761	1895	7656			513				10280	4953	15233
	1982			2729	7050	2746	9796			749				13316	7115	20431
	1983			3502	8718	3673	12391			923				15690	8992	24682
	1984			4465	11357	5812	17169			862				20369	12066	32435
	1985			5351	13931	7474	21405			1245				24690	14431	39121
	1986			7783	19413	10319	29732			1589				27069	15126	42195
	1987			10161	22823	13277	36100			1698				26038	13889	39927
	1988			10493	26346	16084	42430			2340				23543	11353	34896
	1989			13422	31602	20734	52336			3518				21418	9545	30963
	1990			15231	38923	25978	64901			3823				19321	8374	27695
	1991			16381	41745	28190	69935			5004			1838	17896	7514	25410
	1992			17044	45178	32321	77499			6665			2392	17748	7210	24958
	1993			19843	48614	34225	82839			8546			3060	17629	6951	24580
	1994			21697	53011	37816	90827			9036			4220	17533	7020	24553
	1995			24285	58011	41129	99140			10444			4805	17706	7063	24769
	1996	15991	10580	26571	64223	44088	108311	6710	5189	11899	2895	2635	5530	17773	6772	24545
	1997	19576	12627	32203	72898	48047	120945	7269	5579	12848	3524	2962	6486	18490	6903	25393
	1998	23344	14273	37617	82158	52294	134452	7777	6084	13861	3497	2792	6289	20235	7439	27674
	1999	26028	15873	41901	92541	59085	151626	8041	5787	13828	3835	3061	6896	23574	8956	32530
	2000	28804	16637	45441	101142	62479	163621	9129	6663	15792	4402	3754	8156	26914	10474	37388
	2001	31614	18560	50174	112051	69491	181542	10292	7748	18040	5189	4449	9638			
	Suma	145357	88550	416492				49218	37050	144212	23342	19653	59310	445704	196108	641812

Fuente: Anuarios estadísticos de población de licenciatura de ANUIES, y Hill, S.T. (2002)

Nota: En 1999, el National Center for Education Statistics no liberó datos detallados, por lo que las cifras de Estados Unidos de ese año, son promedios de los años 1998 y 2000.

Tabla 1.3 Primer Ingreso, estudiantes, egresados y titulados de Técnicos Superiores Universitarios en Informática y Computación en México, por año y genero, 1998-2001.

		México											
		Nuevo Ingreso			Estudiantes			Egresados			Titulados		
		H	M	Tot	H	M	Tot	H	M	Tot	H	M	Tot
TSU	1998	575	579	1154	1396	1241	2637	184	144	328	65	53	118
	1999	1555	1325	2880	2431	2087	4518	243	279	522	117	106	223
	2000	2347	1715	4062	4102	3167	7269	453	412	865	280	283	563
	2001	3524	2563	6087	5883	4203	10086	1024	845	1869	445	412	857
	Suma	8001	6182	14183				1904	1680	3584	907	854	1761

Fuente: Anuarios estadísticos de población de posgrado de ANUIES.

Tabla 1.4 Primer Ingreso, estudiantes, egresados y titulados de Maestrías en Informática y Computación en México y egresados de Computer Science en Estados Unidos, por año y género, 1966-2001.

		México												EU		
		Nuevo Ingreso			Estudiantes			Egresados			Titulados			Egresados		
		H	M	Tot	H	M	Tot	H	M	Tot	H	M	Tot	H	M	Tot
Maestría	1966													221	17	238
	1967													423	26	449
	1968													518	30	548
	1969													939	73	1012
	1970													1324	135	1459
	1971													1424	164	1588
	1972													1752	225	1977
	1973													1888	225	2113
	1974													1983	293	2276
	1975													1961	338	2299
	1976													2226	377	2603
	1977													2332	466	2798
	1978													2471	567	3038
	1979			57	300	28	328			17			0	2480	575	3055
	1980			96	381	34	415			25			0	2883	764	3647
	1981			105	265	39	304			38			0	3247	971	4218
	1982			171	470	46	516			78			0	3625	1310	4935
	1983			267	485	81	566			75			0	3813	1508	5321
	1984			216	603	115	718			68			0	4379	1811	6190
	1985			287	699	133	832			129			0	5064	2037	7101
	1986			233	653	107	760			152			0	5658	2412	8070
	1987			337	719	144	863			104			0	5985	2496	8481
	1988			283	728	156	884			126			0	6702	2464	9166
	1989			257	709	211	920			105			0	6773	2626	9399
	1990			269	812	214	1026			88			0	6968	2675	9643
	1991			310	839	213	1052			86			0	6563	2761	9324
	1992			340	1012	286	1298			161			0	6980	2675	9655
	1993			397	963	360	1323			147			0	7554	2795	10349
	1994			281	1070	393	1463			168			0	7817	2729	10546
	1995			549	1381	575	1956			245			0	7777	2786	10563
	1996	311	129	440	1394	509	1903	174	89	263	0	0	0	7763	2850	10613
	1997	567	186	753	1614	566	2180	370	122	492	0	0	0	7510	2979	10489
	1998	511	225	736	1882	737	2619	314	122	436	0	0	0	8338	3414	11752
	1999	597	220	817	2111	905	3016	438	190	628	157	55	212	8999	4141	13140
	2000	824	336	1160	2468	1023	3491	499	220	719	197	114	311	9661	4988	14529
	2001	1039	433	1472	3095	1279	4374	564	280	844	348	190	538			
	Suma	3849	1529	9833				2359	1023	5194	702	359	1061	156001	56583	212584

Fuentes: Anuarios estadísticos de población de posgrado de ANUIES, y Hill, S.T. (2002)

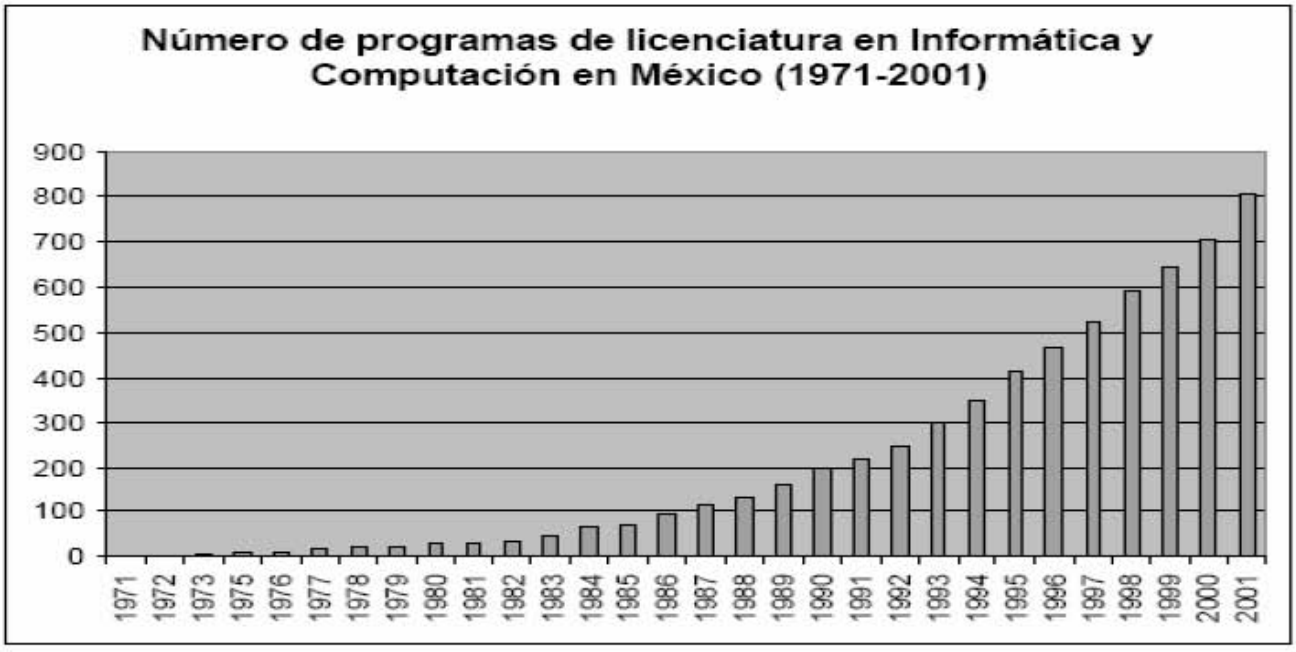
Nota: En 1999, el National Center for Education Statistics no liberó datos detallados, así que las cifras de Estados Unidos de 1999, son promedios simples de los años 98 y 2000.

Tabla 1.5 Primer Ingreso, estudiantes, egresados y titulados de Doctorados en Informática y Computación en México y egresados de Computer Science en Estados Unidos, por año y género, 1966-2001.

		México												Estados Unidos		
		Nuevo Ingreso			Estudiantes			Egresados			Titulados			Egresados		
		H	M	Tot	H	M	Tot	H	M	Tot	H	M	Tot	H	M	Tot
Doctorado	1966													19	0	19
	1967													37	1	38
	1968													36	0	36
	1969													62	2	64
	1970													105	2	107
	1971													125	3	128
	1972													155	12	167
	1973													181	15	196
	1974													189	9	198
	1975													199	14	213
	1976													221	23	244
	1977													197	19	216
	1978													181	15	196
	1979			0	0	0	0			0			0	183	27	210
	1980			0	0	0	0			0			0	197	21	218
	1981			0	0	0	0			0			0	206	26	232
	1982			0	0	0	0			0			0	200	20	220
	1983			0	0	0	0			0			0	250	36	286
	1984			0	0	0	0			0			0	258	37	295
	1985			0	0	0	0			0			0	277	33	310
	1986			0	0	0	0			0			0	351	48	399
	1987			0	0	0	0			0			0	385	65	450
	1988			0	0	0	0			0			0	459	56	515
	1989			0	0	0	0			0			0	504	108	612
	1990			0	0	0	0			0			0	595	110	705
	1991			0	0	0	0			0			0	679	117	796
	1992			6	6	0	6			0			0	747	120	867
	1993			2	6	1	7			0			0	737	138	875
	1994			0	11	1	12			0			0	762	137	899
	1995			0	11	2	13			0			0	808	186	994
	1996	4	0	4	17	4	21	0	0	0	0	0	0	776	139	915
	1997	8	1	9	41	8	49	1	0	1	0	0	0	744	150	894
	1998	19	4	23	77	15	92	3	0	3	0	0	0	763	159	922
	1999	25	6	31	98	22	120	4	0	4	2	0	2	690	156	846
	2000	51	8	59	136	28	164	16	0	16	12	0	12	717	142	859
	2001	38	9	47	150	33	183	31	6	37	3	1	4			
	Suma	145	28	181				55	6	61	17	1	18	12995	2146	15141

Fuente: Anuarios estadísticos de población de posgrado de ANUIES, y Hill, S.T. (2002)

Programas de licenciatura en Informática y Computación en México1971-2001. (Gráfica 1.2)



Como se puede observar en la gráfica 1.2, el número de programas de licenciatura ha tenido un crecimiento constante, acentuándose la pendiente en los últimos años. A consecuencia de esta gráfica se desprende que en México se han usado noventa nombres diferentes para los programas de licenciatura en Informática y Computación.

Tabla 1.6 Equivalencia de los nombres de los programas a los perfiles de la ANIEI[12]

Ing. en Computacion (IC)
Ing. Electricista en Computación (IEC)
Ing. Electrónico en Computación (IEC)
Ing. en Cibernética (ICIB)
Ing. en Cibernética Electrónica (ICE)
Ing. en Cibernética y Ciencias Computacionales (ICCC)
Ing. en Cibernética y en Sistemas Computacionales (ICSC)
Ing. en Ciencias Computacionales y Telecomunicaciones (ICCT)
Ing. en Computación e Informática (ICI)
Ing. en Computación y Redes de Computadoras (ICR)
Ing. en Computación y Sistemas (ICS)
Ing. en Computación y Sistemas Digitales (ICSD)
Ing. en Control y Computación (ICOC)
Ing. en Electrónica y Computación (IEC)
Ing. en Procesos Discretos y Automáticos: Robótica Ind. (IPDA)
Ing. en Sistemas Digitales y Redes (SDR)
Ing. en Sistemas Electrónicos (ISE)
Ing. en Sistemas y Comunicaciones (ISCO)
Ing. en Tecnología de la Información (ITI)
Ing. en Tecnologías de la Informática y la Computación (ITIC)
Ing. en Tecnologías de Información y Telecomunicaciones (ITIT)
Ing. en Telemática (IT)
Lic. en Automatización en Sistemas (LAUS)
Lic. en Computación (LC)
Lic. en Desarrollo de Sistemas Electrónicos y de Inf. (LDSEI)
Lic. en Teleinformática (LT)

Ing. en Sistemas Computacionales (ISC)
Ing. de Sistemas (IS)
Ing. de Sistemas Computacionales (ISC)
Ing. en Desarrollo Computacional (IDC)
Ing. en Informática (II)
Ing. en Sistemas (IS)
Ing. en Sistemas Computarizados e Informática (ISCI)
Ing. en Sistemas de Información (ISI)
Ing. en Sistemas Operacionales (ISO)
Ing. en Tecnologías Estratégicas de Información (ITEI)
Ing. Técnico en Sistemas Computacionales (ITSC)
Lic. en Computación y Sistemas (LCS)
Lic. en Informática en Sistemas Computacionales (LISC)
Lic. en Informática en Sistemas de Información (LISI)
Lic. en Informática Industrial (LII)
Lic. en Informática Programador (LIP)
Lic. en Informática y Sistemas (LIS)
Lic. en Sistemas Computacionales (LSC)
Lic. en Sistemas Computacionales e Informática (LSCI)
Lic. en Sistemas Computarizados e Informática (LSCI)
Lic. en Sistemas e Informática (LSI)
Lic. en Sistemas Informáticos (LSI)
Lic. en Tecnología de la Información (LTI)
Lic. Técnico en Informática (LTEI)
Lic. Técnico Industrial en Sistemas Computacionales (LTISC)
Lic. en Sistemas de Información (LSI)

Lic. En Informatica (LI)
Ing. Administrador de Sistemas (IAS)
Ing. Administrador y de Sistemas (IAS)
Ing. en Computación Administrativa y de Producción (ICAP)
Ing. en Informática Corporativa (IIC)
Ing. en Sistemas Administrativos (ISA)
Ing. en Sistemas Computacionales y Administrativos (ISCA)
Ing. en Sistemas de Computación Administrativa (ISCA)
Lic. en Administración de Computación (LAC)
Lic. en Administración de Empresas y Sistemas (LAES)
Lic. en Administración de la Tecnología Informática (LATI)
Lic. en Administración de Sistemas Computacionales (LASC)
Lic. en Administración y Sistemas (LAS)
Lic. en Administración: Computación (LAC)
Lic. en Computación Administrativa (LCA)
Lic. en Informática Administrativa (LIA)
Lic. en Informática Aplicada (LIAP)
Lic. en Informática Financiera (LIF)
Lic. en Sistemas Administrativos (LSA)
Lic. en Sistemas Administrativos e Informática (LSAI)
Lic. en Sistemas Administrativos y Contables (LSAC)
Lic. en Sistemas Automatizados para la Administración (LSAA)
Lic. en Sistemas Comerciales (LSCO)
Lic. en Sistemas Computacionales Aplicados a la Admón. (LSCA)
Lic. en Sistemas Computacionales Administrativos (LSCA)
Lic. en Sistemas Computacionales, Contables y Admin. (LSCCA)
Lic. en Sistemas Computacionales y Administrativos (LSCA)
Lic. en Sistemas de Computación Administrativa (LSCA)
Lic. en Sistemas de Información Administrativa (LSIA)
Lic. en Sistemas de Informática y Auditoría (LSIAU)
Lic. en Sistemas de la Computación Administrativa (LSCA)

Lic. En Ciencias Computacionales (LCC)
Ing. en Ciencias Computacionales (ICC)
Lic. en Ciencias de la Computación (LCC)
Lic. en Ciencias de la Informática (LCI)
Lic. en Informática y Estadística (LIE)
Lic. en Matemáticas Aplicadas y Computación (LMAC)

[12] Software.net.mx. Op. cit.

La tabla 1.6 muestra la lista de carreras relacionadas con la Tecnología Informática y que son impartidas en México. Esta lista esta dividida en cuatro grandes bloques: Ingenierías en Computación (IC), Ingenierías en Sistemas Computacionales (ISC), Licenciaturas En Informática (LI) y Licenciaturas en Ciencias Computacionales. El extenso número y su súbito crecimiento nos da una idea de la importancia que esta tomando la tecnología informática en la economía mexicana; Y da pie para meditar sobre la importancia de unificar estas carreras en un único plan de estudio, tal vez como una ciencia, con lo que se tendría un nivel de conocimientos más estándar para los egresados de dichas carreras.

1.2.2.1 Importancia del líder de proyecto

En el apartado 1.2.2 de la presente investigación se describió la importancia de los recursos humanos para la industria del software. Además, se dieron a conocer cifras importantes sobre la disponibilidad y el nivel académico con que hoy en día se cuenta en México. Otro aspecto importante de los recursos humanos como parte fundamental para un proceso de producción de software, es el referente al lider de proyecto. En este apartado se describen las características más importantes de un líder de proyectos.

Un líder, es aquella persona que cuenta con la capacidad para influir sobre los demás; un líder de proyecto en el ámbito de la producción de un software es aquella persona encargada de administrar el proceso de producción del mismo. Es el encargado, junto con el cliente, de llevarlo a buen termino.

Russell L. Ackoff, en su libro “El Arte de resolver problemas”, describe algunos aspectos importantes sobre la administración y el administrador. Las características que considera esenciales para una buena administración son:

- Capacidad.
- Comunicación.
- Consciencia.
- Constancia.
- Creatividad.

Resaltando la importancia de la creatividad ante todas.

Sobre el administrador dice:

- Un administrador puede realizar un buen trabajo sin la creatividad, pero no uno sobresaliente.
- El administrador creativo genera sus propias oportunidades.

De lo anterior se puede concluir que el líder de proyecto no solo debe ser una persona altamente capacitada en los diferentes aspectos de la administración y el manejo de recursos humanos, sino también, debe contar con las habilidades naturales que le permitan persuadir a las personas para seguirlo y apoyarlo en las actividades. La característica principal de una líder de proyecto seria la creatividad, la forma de ver los problemas y su capacidad para resolverlos.

1.2.3 Metodologías

Antes de iniciar a explicar en que consisten las metodologías existentes para el proceso de producción de software, se definirá este término de la siguiente manera:

- ✓ Manera sistémica de hacer cierta cosa.
- ✓ Parte de la lógica que estudia los métodos. Se divide en dos partes: la sistemática, que fija las normas de la definición, de la división, de la clasificación y de la prueba, y la inventiva, que fija las normas de los métodos de investigación propios de cada ciencia

En consecuencia, las metodologías de producción de software son un conjunto de técnicas y procedimientos que permiten controlar el proceso de producción de productos de software. Además, estas hacen uso de herramientas de software que permiten la automatización de dicho proceso.

Haciendo una analogía, se podría decir que es como un libro de recetas de cocina, en el que se van indicando paso a paso todas las actividades a realizar para lograr el producto informático deseado, indicando qué personas deben participar en el desarrollo de las actividades y qué papel deben de tener. Además detallan la información que se debe producir como resultado de una actividad y la necesaria para comenzarla.

Las técnicas indican cómo debe ser realizada una determinada actividad que es descrita en la metodología. Combina el empleo de modelos o representaciones gráficas junto con el empleo de procedimientos detallados. Se debe tener en consideración que una determinada técnica puede ser utilizada en una o más actividades de la metodología de producción de software.

Las metodologías de producción de software, aparecen por la necesidad de poner orden al proceso de construcción del software. Regularmente estas abarcan las siguientes etapas en su ciclo de vida:

- ✓ Planificación: ámbito del proyecto, estudio de viabilidad, análisis de riesgos, planificación temporal, asignación de recursos.
- ✓ Análisis (¿qué?): levantamiento de requerimientos
- ✓ Diseño (¿cómo?): estudio de alternativas, diseño arquitectónico
- ✓ Implementación: adquisición, creación e integración de los recursos necesarios para que el sistema funcione.
- ✓ Pruebas: pruebas de unidad, pruebas de integración, pruebas alfa, pruebas beta, pruebas de aceptación.
- ✓ Mantenimiento (correctivo y adaptativo).

Hoy en día existe un sin número de metodologías que sirven de apoyo para el proceso de producción de software, para el estudio del presente apartado las dividiremos en dos grupos:

Tradicionales.

- Imponen un proceso disciplinado con el objetivo de hacer el trabajo más predecible, eficiente y planificado.
- Suelen ser burocráticas y “orientadas por documentos”.
- No se han destacado por ser exitosas.

Ligeras.

- Estas son basadas en la adaptabilidad, más que en el carácter predictivo.
- Son más orientadas a las personas que a los procesos.
- El diseño especifica las piezas y como ellas se relacionan.
- El diseño es la base del plan de construcción.

- El plan define las tareas y sus dependencias, permite definir la agenda y el presupuesto de construcción.
- El diseño requiere de gente más preparada, creativa y costosa.
- La construcción requiere de gente menos preparada y costosa.
- Un buen diseño establece una forma directa y planificada de construir la aplicación.

A pesar de lo anterior, se puede decir que ninguna metodología hará el trabajo por ti, por que ninguna metodología trabaja sola.

1.3 La importancia de las metodologías de producción de software

Una de las consideraciones más importantes para el éxito en la producción de un producto de software es, sin duda, la elección del mecanismo o metodología a seguir para lograr el objetivo planteado. Toda producción de software es riesgosa y difícil de controlar, pero si aunado a esto no se lleva una metodología de por medio, lo que obtenemos son clientes insatisfechos y productos nunca terminados o de baja calidad.

En su libro, Agile Software Development, Alistair Cockburn define el término metodología de software como: "todas aquellas actividades que se realizan regularmente para conseguir un software. Esto incluye a quién se emplea, para qué se emplea, cómo es su trabajo en conjunto, qué él produce, y cómo comparte. Además, la descripción del trabajo combinado, los procedimientos, y las convenciones de cada uno en su equipo. Es el producto de un ecosistema particular y es por lo tanto una construcción única de cada organización[13]".

De lo anterior se puede decir que una metodología de software es un conjunto de actividades relacionadas entre si con el fin de conseguir un objetivo común, *la producción de un producto de software de calidad*. Estas actividades describen la forma en la cual deben interactuar los diferentes elementos que componen esta tarea a fin cumplir su objetivo. Además considera las herramientas, estrategias, recursos humanos, reglas y procedimientos a utilizar dentro de la misma.

La mayoría de las veces no se le da la suficiente importancia al hecho de utilizar una metodología

[13] Alistair Cockburn. Agile Software Development. Ed. Highsmith Series. 2001. E.U.A. p. 101.

adecuada y generalmente se opta por utilizar la que ya se conoce, sin considerar que esta cubra con los escenarios mínimos del tipo de software a desarrollar. Esta práctica se vuelve más común cuando los productos a generar abarcan lapsos de tiempo corto (dos o tres meses), en donde en ocasiones se realiza la separación del aplicativo en procesos y con base en esto se inicia la construcción del producto.

Generalmente cuando los productos de software a desarrollar son de mayor dimensión, se le da más importancia al uso de una metodología adecuada, y se realiza un proceso de evaluación para elegir la mas apropiada para el caso particular. Lo cierto es que en la mayoría de las ocasiones no se encuentra la más adecuada y se termina por hacer uso de técnicas de diferentes metodologías o, en algunas ocasiones, diseñar la propia.

A continuación se listan algunos de los problemas que pueden surgir por el no uso de una metodología:

- Pérdida del control sobre el proceso.
- Toma de decisiones erróneas (insuficiente información).
- Resultados impredecibles.
- Detección tardía de errores.
- Desconocimiento de roles y responsabilidades.

Las empresas dedicadas a la producción de software en México, utilizan como herramienta para su producción, aquellas metodologías que surgen en los principales países productores de software, y que no necesariamente cumplen con las condiciones estructurales del país. Esto provoca, que gran parte de los proyectos dedicados a la elaboración de un producto de software en México, terminen a destiempo, con baja calidad y no cumplan las expectativas iniciales del cliente.

Debido a la cercanía de México con uno de los principales, si no es que el principal productor de software, cuenta con la ventaja de conocer y aplicar estos procesos antes que la mayoría de los países, estando a la vanguardia en el uso de procesos de producción. Esto da a las empresas mexicanas la oportunidad, en el mejor de los casos, de adaptar dichas metodologías a las circunstancia nacionales.

En la situación actual, la empresa que comience a poner los elementos necesarios para mejorar el proceso de producción de software tendrá mucha más ventaja competitiva frente a las demás.

Conclusiones

Como se observo en el apartado 1.1 de este capítulo, la tecnología informática en México esta creciendo en forma consistente, sin embargo, aun no se ha logrado alcanzar el crecimiento promedio que la economía mundial ha desarrollado para este sector (promedio mundial de 6.6%). México, imitando a otros países que son líderes en el sector, actualmente esta implementado un programa nacional para el desarrollo de la industria del software, con el objetivo de consolidarlo como el líder latinoamericano de desarrollo de software y contenidos digitales en español. En consecuencia este programa trata de establecer las bases estructurales necesarias para el desarrollo de la industria del software en México.

El crecimiento promedio mundial de la industria informática en la década de los noventas alcazo el 6.6%, mientras tanto en México para los años de: 1994-1.9%, 1995-2.1%, 1996-2.3%,1997-2.4%, 1998-2.6%, 1999-2.9%. 2000-3.2%, 2001-3.5%, 2002-3.7%, 2003-3.9% y hasta el tercer trimestre del 2004-4.5%.

Adicionalmente, como se observo en las secciones del apartado 1.1 de este capítulo, las perspectivas que se dan a conocer por parte del INEGI, las empresas encargadas del análisis del mercado para el sector de la producción de software (select) y las asociaciones dirigidas al mismo (AMITI), presentan un panorama alentador, en estos momentos, para la detonación del sector a corto y mediano plazo.

Una vez dadas estas condiciones, y como se menciono en el apartado 1.3 de este capitulo, el éxito de la producción de software, queda reducida a la eficiente aplicación y elección de la metodología mas apropiada para el caso particular. Es por esto que se debe remarcar que las metodologías son las encargadas de guiar el proceso de producción de software y la experiencia ha demostrado que la llave del éxito es la elección correcta de la metodología adecuada, lo que puede conducir al equipo de trabajo a desarrollar un producto de calidad. La elección de la metodología adecuada es más importante que utilizar las mejores y más potentes herramientas disponibles en el mercado.

CAPÍTULO 2

ANÁLISIS Y EVALUACIÓN DE METODOLOGÍAS DE PRODUCCIÓN DE SOFTWARE

El capítulo dos cubre cuatro de los aspectos fundamentales para el presente trabajo de investigación. En primer lugar, se realiza un análisis del problema enfocado a los principales factores que inciden sobre el proceso de producción de un software (Educación, Infraestructura tecnológica, Metodologías de producción de software, Inversiones y Legislación). A continuación, y una vez analizados estos factores, se describe la alternativa de solución al problema inicial planteado (La elección y uso de metodologías para el proceso de producción de software en México). Como tercer aspecto, se encuentra el análisis de las metodologías Rational Unified Process (RUP), Extreme Programming (XP) y Microsoft Soluction Framework (MSF), Scrum, Crystal Family, Open Source y Feature Driven Development (descripción de las características más relevantes de cada una de ellas). Por ultimo, y con base en una calificación cualitativa, se evalúan las metodologías para dar paso a lo que será la base de la propuesta de solución.

2.1 Análisis del problema.

Como se describió en el capítulo 1, para conseguir que el proceso de producción de software sea mas eficiente y sin fallas, es indispensable contar con la infraestructura necesaria que le de el soporte adecuado. Algunos de los factores que inciden directamente sobre el proceso de producción de software se mencionan a continuación:

- Educación.
- Infraestructura tecnológica (hardware, software, telecomunicaciones, etc.).
- Metodologías de producción de software.

Para la situación particular llamada México, se sumarian los siguientes factores:

- Inversiones
- Legislación

Educación. Es importante contar con una infraestructura educativa que permita generar profesionistas de calidad en el área de ingeniería de software, informática, etc. Actualmente México cuenta con alrededor de 80 diferentes programas de licenciaturas en informática y computación, lo cual no quiere decir que el numero de egresados de estas diferentes carreras sean los suficientes y los mejor capacitados para emprender en el ámbito laboral. Si realizamos una comparación con

nuestro país vecino (Estados Unidos de América) y uno de los principales productores de software en el mundo, se puede observar que el número de egresados en México, en estas carreras, es mucho menor en comparación a este. Esto implica que si México desea consolidarse como uno de los países importantes en la producción de software, deberá ejecutar una serie de medidas en educación que permita alcanzar índices decentes en la cantidad y calidad de los egresados de los programas de licenciatura en informática y computación.

Infraestructura tecnológica. Actualmente la globalización en el mundo ha provocado que las empresas busquen los mejores medios para llegar a sus objetivos. Las empresas mexicanas no se han quedado atrás y la mayoría, en estos momentos, cuenta con la suficiente infraestructura tecnológica para montar nuevos sistemas computacionales que les permita optimizar los tiempos de sus procesos. Aunado a esto, se puede decir, que México cuenta con una infraestructura en comunicaciones que permite a cualquier empresa tener acceso a servicios especializados de este sector.

Inversiones. Para el caso particular de México, es importante mencionar que se deben incentivar las inversiones en el área informática para el mercado interno. Además, para detonar esta industria tan importante como generadora de empleos, es necesario que el gobierno y empresarios emprendan una tarea conjunta a fin de promover la optimización de los procesos de negocio dentro de las empresas. Por otro lado, es necesario que México se gane el prestigio como un país productor de software de calidad, esto le dará más oportunidades a las empresas mexicanas de extender sus mercados y generar un número superior de empleos.

Legislación. Al igual que para muchas otras actividades, las leyes mexicanas que determinan las reglas a seguir para la inversión y el establecimiento de empresas dedicadas a la producción de software, son obsoletas. Es necesario legislar sobre estas para dar un marco más amplio de maniobra y las garantías necesarias tanto a inversionistas como a empresas establecidas.

Metodologías de producción de software. Como se menciono en el capitulo 1, la importancia del uso de una metodología acorde con las circunstancias del proyecto de producción del software, puede ser la diferencia entre el éxito o fracaso para el mismo. Generalmente, las metodologías de producción de software, únicamente contemplan el escenario para el cual fueron creadas, dejando vacíos e incertidumbre en escenarios distintos.

2.2 Estrategias de solución.

Actualmente en México se lleva a cabo, por parte del gobierno federal, un programa nacional para el desarrollo de la industria del software (PROSOFT). Este programa contempla siete estrategias las cuales fueron detalladas en el capitulo 1, y se listan a continuación:

1. Promover las exportaciones y la atracción de inversiones
2. Desarrollar capital humano en cantidad y calidad suficiente
3. Contar con un marco legal promotor de la industria
4. Desarrollar el mercado interno
5. Fortalecer a la industria local
6. Alcanzar niveles internacionales en capacidad de procesos
7. Promover la construcción de infraestructura física y de telecomunicaciones

Estas estrategias se están ejecutando en forma paralela a fin de lograr las metas propuesta por el gobierno federal a un mediano plazo, se esta hablando del año 2013 y las metas son:

- Lograr una producción anual de software de 5,000 millones de dólares.
- Alcanzar el promedio mundial de gasto en tecnologías de información.
- Convertir a México en el líder latinoamericano de desarrollo de software y contenidos digitales en español.

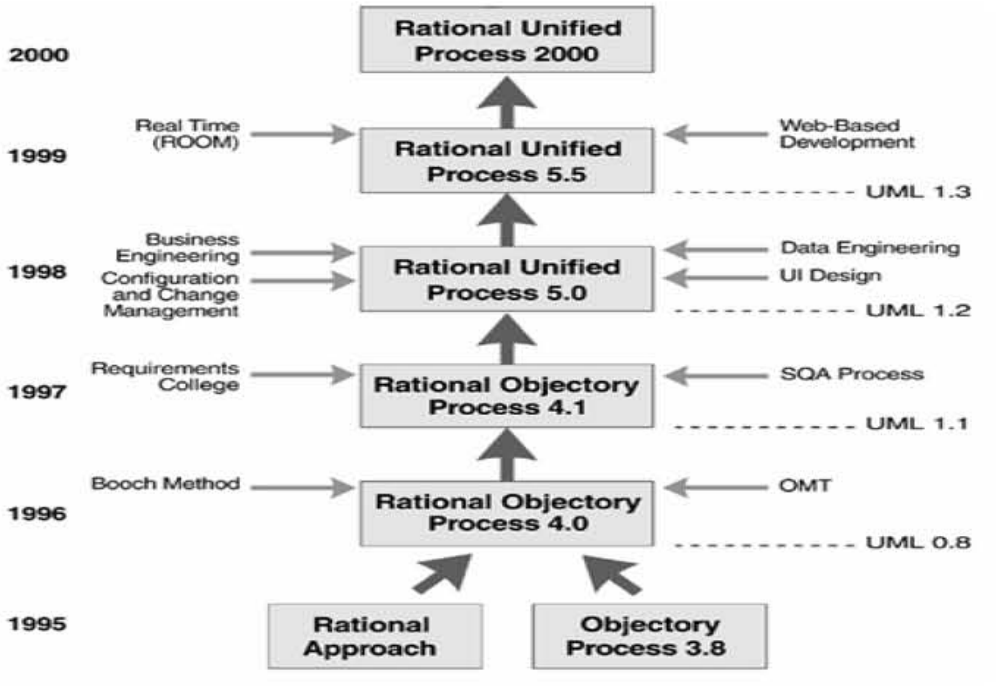
Como se puede observar, estas estrategias cubren cuatro de los cinco factores que inciden directamente sobre el proceso de producción de software, que se mencionaron en el apartado anterior. Dada esta circunstancia, en el presente trabajo de investigación se enfocara a analizar las tres principales metodologías de producción de software: Rational Unified Process (RUP), Extreme Programming (XP) y Microsoft Solution Framework (MSF). Con la finalidad de conocer sus principales característica, diferencias y similitudes a fin de generar una alternativa a seguir no importando las particularidades del proyecto de producción de software que se este llevando a cabo.

2.3 Rational Unified Process (RUP)

Rational Unified Process es una estructura de procesos para el desarrollo de software que provee de una técnica depurada para asignar tareas y responsabilidades dentro de una organización de desarrollo de software. Su objetivo es asegurar la producción de un software de alta calidad que satisfaga las necesidades de sus usuarios finales con un predecible tiempo y presupuesto[14].

A lo largo de los años, Racional Unified Process ha madurado en su propuesta, en ella se refleja la experiencia de mucha gente y compañías que al adoptarla como proceso de desarrollo, la enriquecieron y complementaron hasta llegar a ser lo que ahora es. En la figura 2.3.1 que a continuación se presenta, se ilustra la evolución del proceso a través de los años.

Genealogía del proceso unificado racional (figura 2.3.1)



Regresando en el tiempo, se puede ver que Racional Unified Process es el sucesor directo del Racional Objectory Process (versión 4). El proceso unificado racional incorpora más material en las áreas de la ingeniería de datos, del modelado de negocio, de la gerencia de proyecto, y de la

[14] Philippe Kruchten.The.rational unified process an introduction 3rd edition. Ed. Addison Wesley. Diciembre 2003, E.U.A. p. 33.

gerencia de la configuración. También trae una integración más cercana con la suite de herramientas de Racional Software.

El Racional Objectory Process era el resultado de la integración de " Racional Approach" y del Objectory Process (versión 3), después de la fusión de Rational Software Corporation y de Objectory AB en 1995. De Objectory, el proceso ha heredado su estructura de proceso y el concepto central del caso del uso. De Racional, ganó la formulación actual del desarrollo iterativo y la arquitectura. Esta versión también incorpora material sobre la administración de requerimientos, de Requisite Inc. y de un proceso detallado de pruebas heredado de SQA®, Inc., compañías que también se combinaron con Rational Software. Finalmente, este proceso fue el primero en utilizar el recién creado lenguaje de modelado unificado (UML 0.8).

El proceso de Objectory fue creado en Suecia en 1987 por Ivar Jacobson como resultado de su experiencia con Ericsson. Este proceso se convirtió en un producto en su compañía, Objectory AB. Centrado alrededor del concepto del caso del uso y de un método de diseño orientado al objeto, ganó rápidamente el reconocimiento en la industria del software y ha sido adoptado e integrado por muchas compañías por todo el mundo. Una versión simplificada del proceso de Objectory fue publicada como libro de texto en 1992.

El Racional Unified Process es un caso específico y detallado de un proceso más genérico descrito por Ivar Jacobson, Grady Booch, y James Rumbaugh en el libro, *The Unified Software Development Process*.

2.3.1 Prácticas del proceso unificado

RUP permite implementar 6 de las mejores prácticas de ingeniería de software:

- Desarrollo iterativo de software (Ciclo de Vida)
- Administración de Requerimientos
- Uso de arquitecturas basadas en componentes
- Modelado visual de software haciendo uso efectivo de UML[15]

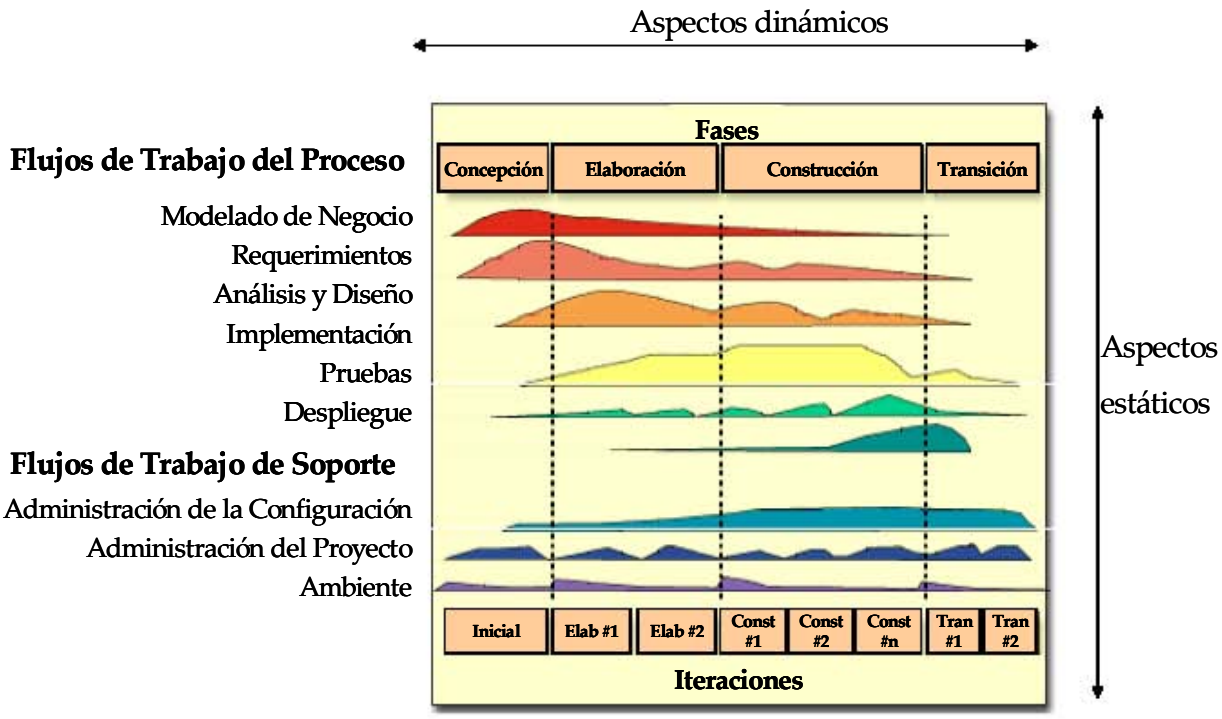
[15] **Unified Modeling Language,** el cuál es un lenguaje de modelado estándar de la industria que permite comunicar claramente los requerimientos, arquitecturas y diseño.

- Verificación de la calidad del software.
- Control de cambios de software.

El proceso de desarrollo de software que propone RUP, se representa en la figura 2.3.2, el cuál plantea dos dimensiones:

- La horizontal está asociada al tiempo y esta enfocada a los aspectos dinámicos del proceso, se expresa como ciclos, fases, Iteraciones e hitos.
- La vertical representa los aspectos estáticos del proceso como son: actividades, participantes, flujos de trabajos y artefactos (Documentos).

Proceso de desarrollo de Software (Figura 2.3.2)[16]



➢ *Desarrollo iterativo de software*

El desarrollo iterativo es un método de construcción de software cuyo ciclo de vida está compuesto por un conjunto de iteraciones. Cada iteración tiene como objetivo primordial entregar una versión del producto de software terminada. La iteración es concebida como un subproyecto que genera

[16] Philippe Kruchten.The.rational unified process an introduction 3rd edition. Ed. Addison Wesley. Diciembre 2003, E.U.A. p. 22.

productos de software, permitiendo al usuario tener puntos de verificación y control más rápidos y efectivos; Al mismo tiempo, esto provoca la aplicación de un proceso continuo de pruebas y de integración del código generado desde el principio de las iteraciones.

Las iteraciones están compuestas por el conjunto de disciplinas o actividades ya conocidas en el proceso de desarrollo de software. Estas son las especificaciones de requerimientos, el análisis y diseño, las pruebas, la administración de la configuración y el proceso de administración del proyecto.

- *Administración de Requerimientos*

Se utiliza un acercamiento sistemático para extraer y documentar los requerimientos del sistema, y entonces poder administrar cambios sobre estos requerimientos, incluyendo evaluaciones del impacto sobre el resto del sistema. La efectiva administración de requerimientos involucra mantener una clara declaración de estos, así como su disponibilidad para otros artefactos del proyecto.

- *Uso de arquitecturas basadas en componentes*

Uso de componentes para estructurar la arquitectura del software. Un desarrollo basado en componentes tiende a reducir la complejidad de la solución y da como resultado una arquitectura más robusta y adaptable que permite una efectiva reutilización.

- *Modelado visual de software*

Se produce un conjunto de modelos visuales del sistema, cada uno enfatiza en detalles específicos e ignora otros. Estos modelos permiten un mejor entendimiento del sistema que se esta desarrollando y provee de un mecanismo para una mejor comunicación entre los miembros del equipo.

- *Verificación de la calidad del software*

Continuamente se evalúa la calidad del sistema con respecto a los requerimientos funcionales y no funcionales. Se ejecutan pruebas como parte de cada iteración. Esto permite incrementar la detección y corrección de errores a temprana hora durante el ciclo de vida de desarrollo de software, en vez de corregir errores encontrados más tarde.

- *Control de cambios de software*

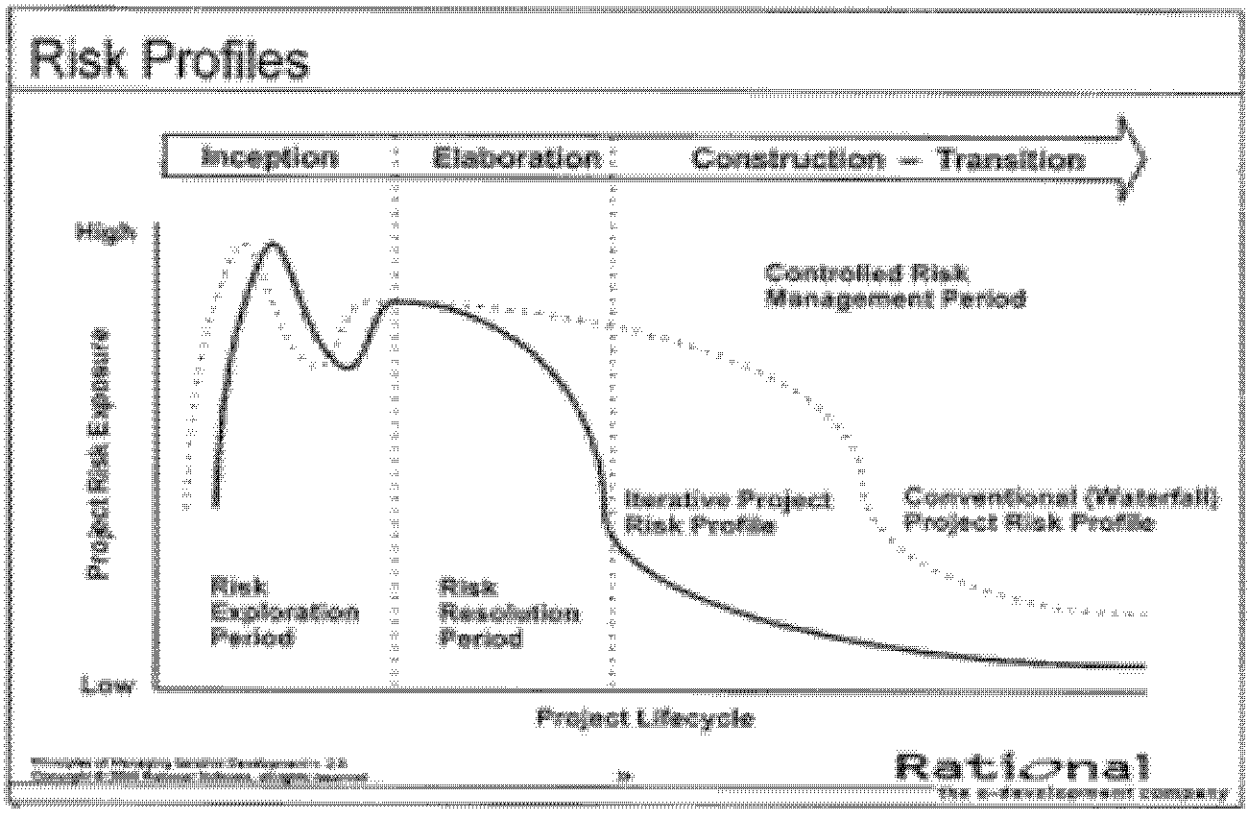
Establece un mecanismo disciplinado y controlado para la administración de cambios. Controla los cambios que son agregados en los artefactos del proyecto, quienes los introdujeron y cuando se realizaron.

Adicionalmente, RUP cuenta con algunas otras prácticas u orientaciones de gran importancia para el proceso de desarrollo de software, y las cuales describimos a continuación:

- *Orientación al manejo de riesgos*

La orientación al manejo del riesgo en RUP, básicamente se refiere a la identificación de los procesos con alto grado de riesgo, tanto en lo tecnológico como en la complejidad de negocio. Esta identificación se debe realizar durante la primera etapa de ciclo de vida del desarrollo de software. Por lo tanto, las prioridades del proyecto deben contemplar estos en primera instancia. Una vez detectado el conjunto de riegos, es necesario establecer un plan para su manejo de forma clara, documentada y con una implementación eficiente. Se pretende evitar posibles retrasos en los tiempos de entrega, problemas de calidad en el producto o en el peor de los casos, que puedan afectar la culminación del proyecto. Estos procesos pueden ser tan complejos y elaborados como la importancia del proyecto lo requiera. En las etapas iniciales se implementan las funcionalidades con mayor nivel de riesgo y las de mayor complejidad, mejorando la posibilidad de éxito del proyecto.

Perfiles de riesgo[17] (Figura 2.3.3)



En la fase inicial del proyecto, el nivel de exposición al riesgo es muy elevado, pero en las fases siguientes es completamente diferente (Ver Figura 2.3.3). Este comportamiento se debe al período de exploración de riesgos del modelo en espiral, donde se identifican los riesgos, se priorizan y se define un plan de manejo para mitigarlos. Se procede a la fase de elaboración donde se implementan aquellos casos de uso que atacan los riesgos de más alta prioridad, lo cual se denomina período de resolución de riesgos. Al final de esta fase se debe tener definida la arquitectura del sistema, así como la infraestructura en la que se soportará.

- *Orientación al cliente*

Cuando se inicia un proyecto de desarrollo de software se conoce la importancia de la participación del cliente para lograr su terminación exitosa, pero usualmente cometemos el error de olvidar esta norma básica, lo que implica que la participación del cliente se restringe al inicio y finalización del proyecto, lo que en la mayoría de los casos produce un alto grado de insatisfacción en el usuario,

[17] Principles of Managing Iterative Development V2.0 Rational Software Corporation. 2001. E.U.A.

al no obtener el producto con las especificaciones esperadas. El cliente es quien realmente conoce el valor que aportará el producto que está siendo desarrollado y puede definir las prioridades desde la perspectiva organizacional. Esto quiere decir que es necesario contar con su participación en el proceso de planificación de las fases y de las iteraciones. Posteriormente se requiere su participación en cada iteración para proveer retroalimentación temprana al equipo de desarrolladores, garantizando el cumplimiento de las expectativas que tiene, además de ofrecerle una visibilidad permanente del estado del proyecto, asegurando su compromiso para terminarlo exitosamente.

Se debe tener en cuenta que el cliente no se interesa por los aspectos técnicos de alta complejidad y riesgo, razón por la cual se debe combinar esta práctica con una orientación al manejo del riesgo.

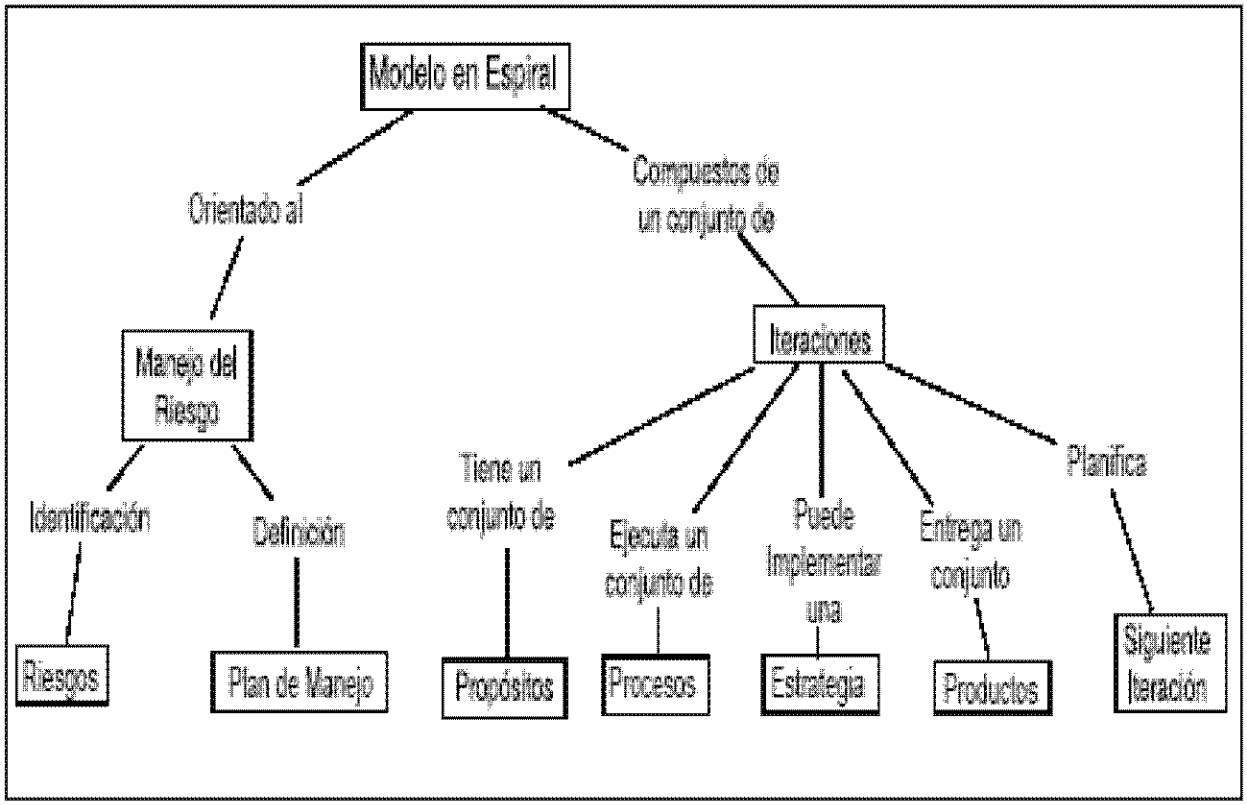
- *Desarrollo evolutivo*

Cuando se trabaja con una especificación de requerimientos monolítica (de gran cohesión), se cae en el error de creer que se comprende completamente el concepto del producto sin haberlo validado con el cliente. Este proceso inicia con un concepto poco claro del producto a construir, y sólo se tiene claridad en la medida que se vaya desarrollando y verificando el producto con el cliente. Este tipo de proyectos se asemejan más al patrón que siguen los proyectos de investigación y desarrollo de nuevos productos.

2.3.2 Modelo en espiral

El modelo en espiral se centra en algunas prácticas fundamentales del desarrollo de software, tales como la orientación al manejo de riesgos, la orientación al cliente y el desarrollo iterativo (Ver Figura 2.3.4). El modelo se organiza en un conjunto de iteraciones que pueden considerarse a sí mismas como pequeños proyectos que siguen el ciclo de vida completo. Las primeras iteraciones tienen como objetivo identificar los riesgos del proyecto para determinar su viabilidad, y en caso de seguir adelante, definir un plan de manejo para mitigarlos o eliminarlos. Adicionalmente, el usuario participa activamente en la priorización de los casos de uso a desarrollar y en el proceso de pruebas, con lo cual se logra obtener una funcionalidad estable y operativa desde las primeras iteraciones del proyecto.

Mapa conceptual del modelo en espiral[18] (Figura 2.3.4)

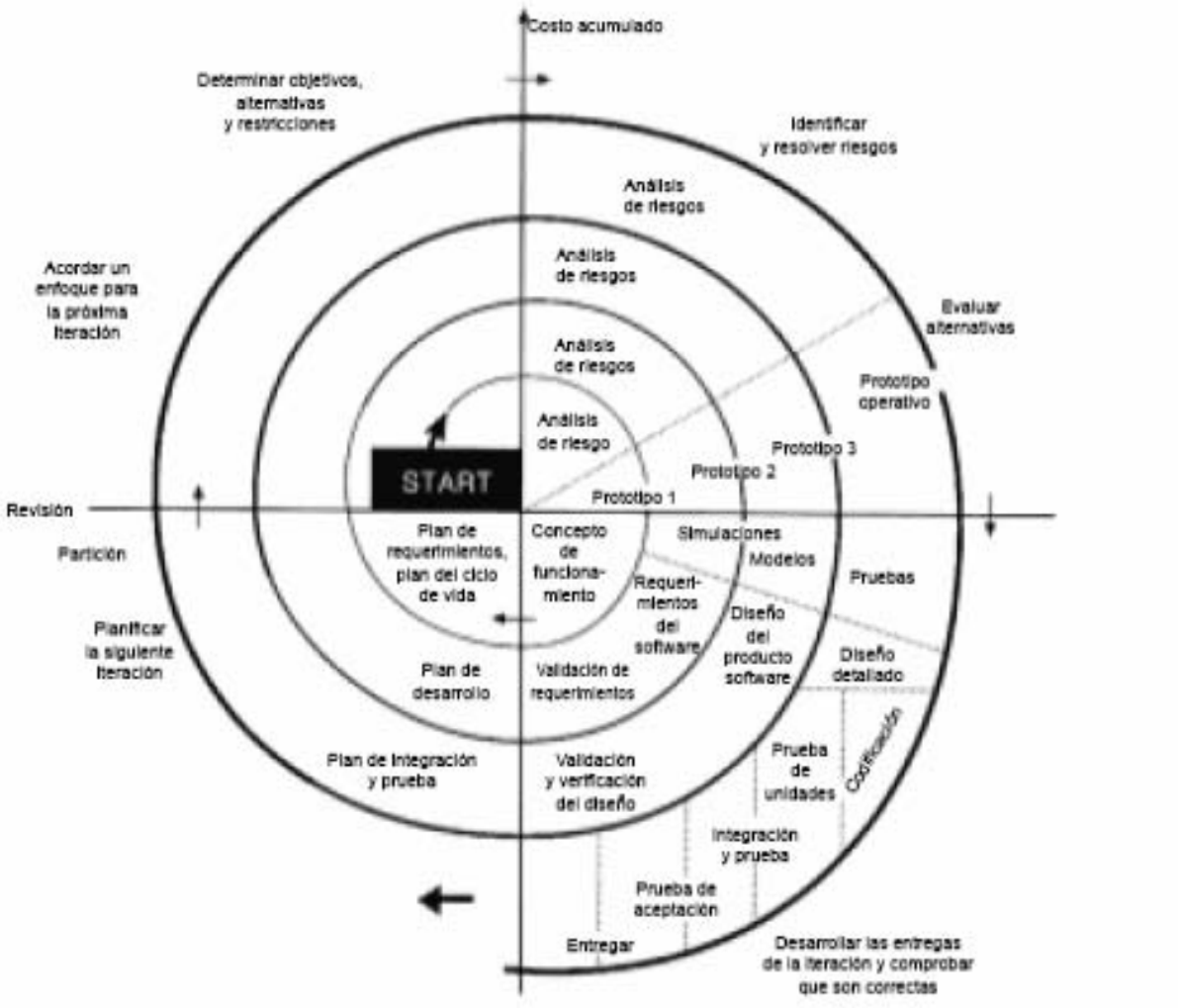


El modelo en espiral tiene muchas ventajas respecto a los modelos anteriores por su orientación a la resolución temprana de riesgos, por la definición de la arquitectura del sistema en sus fases iniciales y por su proceso continuo de verificación de la calidad. En términos generales este modelo tiene un nivel alto de complejidad y requiere mucha destreza administrativa y experiencia por parte del gerente del proyecto y su grupo de trabajo para manejarlo adecuadamente.

Este modelo es ideal para manejar proyectos que requieran la incorporación de nuevas tecnologías, o para desarrollar productos completamente nuevos o con un nivel alto de inestabilidad de los requerimientos. Típicamente se maneja una iteración inicial donde se define el alcance del proyecto, se identifican y priorizan los riesgos y se realiza el modelo de casos de uso inicial para determinar qué casos de uso definirán la arquitectura del sistema. Con la información anterior se procede a definir el plan de manejo de riesgos según su prioridad, así como los casos de uso que serán implementados en las siguientes iteraciones (Ver Figura 2.3.5).

[18] Barry Boehm. “A Spiral Model of Software Development and Enhancement”, ACM SIGSOFT Software Engineering Notes. Agosto 1986. E.U.A.

Modelo en espiral[19] (Figura 2.3.5)



2.3.3 Iteraciones

Una iteración es una secuencia de actividades dentro de un plan establecido, con criterios claros de evaluación, que se organizan con el propósito de entregar parte de la funcionalidad del producto. En las primeras iteraciones se desarrollan o implementan los casos de uso que tienen mayor complejidad y que llevan inherente un alto nivel de riesgo que puede afectar el éxito del proyecto. De esta forma, con cada iteración que se realiza, los riesgos del proyecto se reducen

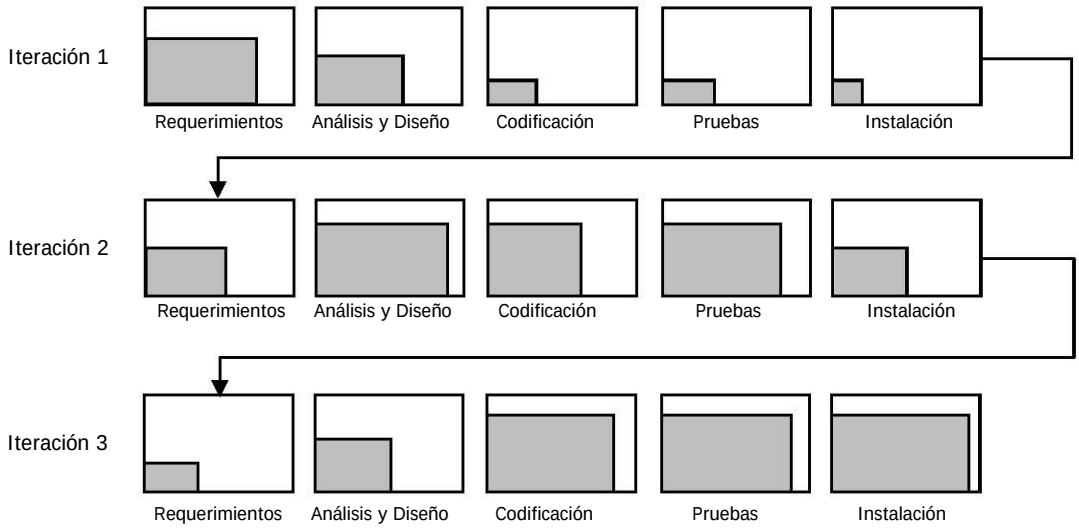
[19] Mc Connell, Steve. Desarrollo *y* gestión *de* proyectos informáticos. Ed. McGraw Hill, 1ª edición 1997. España.

acorde con el plan establecido, el cual está en permanente revisión para monitorear la posible aparición de nuevos riesgos y ajustarlo si es necesario.

La selección de los casos de uso, para cada iteración, se hace tomando en cuenta el mínimo conjunto de ellos que se requiere para implementar la funcionalidad que mayor riesgo tenga. Se debe realizar este proceso hasta que la lista de riesgos haya sido cubierta completamente.

Cada iteración puede tener uno o varios propósitos, lo cual determina su duración; a su vez se ejecutan varios procesos que van desde la especificación de requerimientos hasta las pruebas de unidad e integración. Adicionalmente, al final de cada iteración se evalúan los resultados del trabajo y se planea la siguiente iteración en forma detallada.

Ciclo de vida iterativo[20] (Figura 2.3.6)



2.3.4 Incorporación del proceso unificado en la producción de software

La concepción de un sistema de información va mucho más allá de levantar los requerimientos, elaborar un conjunto de modelos y comenzar a programar. Esta concepción limitada ha permitido que durante años no podamos hacer uso adecuado de los conceptos y las herramientas con los que contamos. En este punto podemos considerar que la definición de la arquitectura del software

[20] Peter Eeles, Kelly Houston, Wojtek Kozaczynski. Building J2EE Applications with the Rational Unified Process. Primera edición, 2002. Ed. Addison Wesley. E.U.A. p.45.

se convierte en el eje orientador que permite controlar el desarrollo iterativo e incremental del sistema, a través de su ciclo de vida. Esta arquitectura se define en las primeras fases del proyecto, principalmente en la de elaboración, y se refina a largo del mismo.

RUP se fundamenta en seis prácticas: el desarrollo iterativo, la administración de requerimientos, la arquitectura basada en componentes, el modelado visual, la verificación continua de la calidad y la administración de cambios. Estas seis prácticas orientan el modelo y con ellas se pretende solucionar muchos de los problemas asociados al software. Adicionalmente hay muchos aspectos de diseño que son bien conocidos, pero han sido muy poco implementados en los proyectos de software; estos son: facilidad de uso, modularidad, encapsulamiento y facilidad de mantenimiento. Es necesario entonces definir una arquitectura sólida basada en componentes, para construir mejores y más flexibles soluciones de software para las necesidades organizacionales.

Los cambios en un proyecto no pueden ser detenidos dado que la evolución del entorno de cada organización es continua, pero sí pueden ser administrados de manera que su impacto pueda ser estimado para determinar si dicho cambio se incluye o no y si el proyecto debe ser reajustado. Cada cambio en el proyecto debe tener especificado cuándo, cómo y quién lo va a realizar y qué productos se ven involucrados en el mismo. En ese punto es donde el control de cambios y la reutilización de los componentes a través de los diversos modelos adquieren una gran importancia.

Existen algunos aspectos que se deben tener en cuenta para desarrollar exitosamente un proyecto. A continuación se enumeran algunos de ellos:

- ✓ Se debe tener definida claramente la metodología de trabajo de cada fase del proceso del desarrollo de software, en especial las fases de administración de requerimientos y control de cambios, los cuales son los eslabones más débiles del proceso de desarrollo de software en nuestras organizaciones. La responsabilidad de definir, documentar y validar que se cumpla estrictamente la metodología de trabajo es del grupo de ingeniería de procesos.

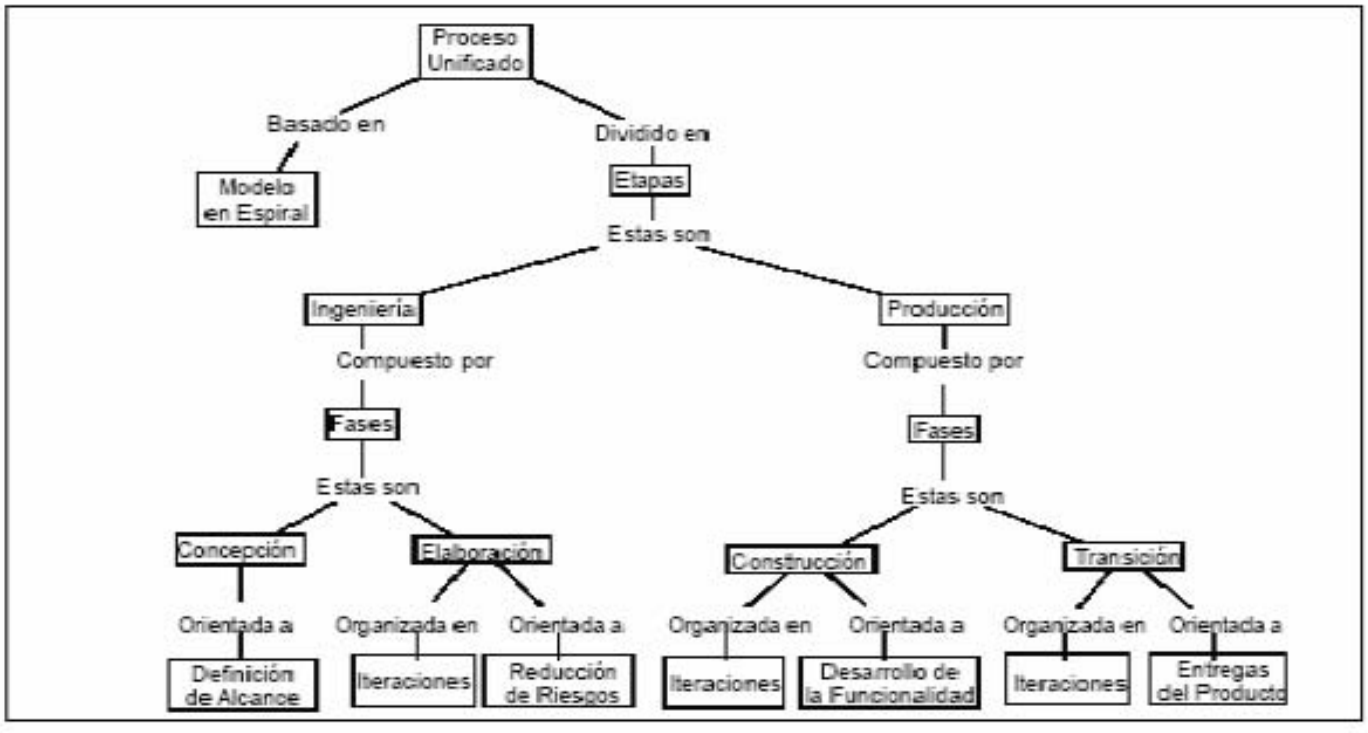
- ✓ La participación activa de los usuarios y los acuerdos en los tiempos pactados, tomando en cuenta los datos generados de los procesos de estimación y planificación, son responsabilidad del jefe del proyecto, pero deben ser elaborados con integrantes claves del equipo.

- ✓ El Grupo SQA3[21] debe definir, documentar y actualizar el proceso de aseguramiento de la calidad del software, gestionar los recursos necesarios para que sea operativo desde el comienzo del proyecto, entregar el plan de calidad y velar por su cumplimiento a lo largo del ciclo de vida del proyecto.

- ✓ El proceso de incorporación y utilización de nuevas tecnologías es uno de los aspectos más críticos dentro del proyecto. La definición de una metodología de administración del cambio tecnológico, clara y muy práctica, facilitaría considerablemente el trabajo realizado en la fase de elaboración, lo cual permitiría determinar la viabilidad de la incorporación de dicha tecnología en el proyecto.

Teniendo en cuenta los aspectos mencionados previamente, Rational elaboró un marco de referencia para el proceso de desarrollo de software basado en el modelo en espiral. Este método se conoce como RUP "Rational Unified Process". Para una mejor organización, el RUP agrupa las iteraciones en etapas y fases que facilitan la administración del proyecto (Ver Figura 2.3.7).

Mapa conceptual del proceso Unificado de Racional[22] (Figura 2.3.7)



[21] Grupo encargado del aseguramiento de la calidad del producto de software.
[22] Principles of Managing Iterative Development V2.0 Rational Software Corporation. 2001. E.U.A.

2.3.5 Etapas del proceso unificado

En RUP, el ciclo de vida del desarrollo de software esta descompuesto en cuatro etapas secuenciales: concepción, elaboración, construcción y transición.

✓ *concepción*

En esta etapa se realiza la conceptualización del sistema. Entre los objetivos principales de la misma se encuentran:

- Establecer la visión del proyecto, en la que se determinan los requerimientos funcionales y técnicos que debe cubrir la solución, así como los criterios de aceptación de los productos que se generen durante el ciclo de desarrollo.
- Identificar los principales riesgos del proyecto y establecer los mecanismos en caso de contingencias.
- Identificar los escenarios operativos críticos, así como los escenarios primarios, a fin de determinar prioridades y complejidad.
- Definir un bosquejo de la arquitectura de diseño de la solución, de acuerdo a los requerimientos y restricciones que se identifiquen en esta fase.
- Obtener una estimación más precisa, del tiempo y costo necesarios para el proyecto completo.
- Definir el ambiente de desarrollo, la forma en que se debe organizar el equipo de trabajo, los perfiles del personal requeridos y seleccionar las herramientas idóneas para el proceso.

Esta fase tiene como propósito definir y acordar el alcance del proyecto, identificar los riesgos asociados al proyecto, proponer una visión general de la arquitectura de software y definir el plan de trabajo para las fases e iteraciones.

A partir de los requerimientos funcionales y técnicos plasmados en los casos de uso; y la definición de los escenarios operativos críticos, se determina el plan de trabajo a seguir, de tal forma que inicialmente se mitigue la funcionalidad de mayor riesgo o complejidad, reduciendo así los puntos críticos que podrían llevar al fracaso del proyecto.

El plan de pruebas debe planearse en esta fase, ejecutarse desde la primera iteración de la fase de elaboración y refinarse sucesivamente durante el ciclo de vida del proyecto. Es importante establecer una relación clara y directa entre los casos de uso y los casos de prueba para facilitar que el proceso de aseguramiento de la calidad del software se ejecute adecuadamente.

✓ elaboración

Durante esta etapa se realiza el diseño inicial de la solución del problema, entre los objetivos principales de la misma se encuentran:

- Detallar los requerimientos funcionales y no funcionales del sistema, mediante los modelos correspondientes.
- Establecer la Arquitectura tecnológica definitiva.
- Establecer el ambiente de desarrollo.
- Evolucionar el Modelo de Datos a un modelo físico.
- Obtener el plan de trabajo definitivo de la construcción.

En esta fase, deben seleccionarse los casos de uso que permitan definir la arquitectura del sistema. Una vez realizada la especificación de los casos de uso y el primer análisis del dominio del problema, se debe diseñar la solución preliminar y comenzar la ejecución del plan de manejo de riesgos, según las prioridades definidas. Al final de esta fase se cuenta con los elementos necesarios para determinar la viabilidad del proyecto y poder definir los planes de trabajo de las etapas de construcción y transición.

✓ construcción

Durante esta etapa se realiza la construcción de la funcionalidad del sistema, entre los objetivos principales de la misma se encuentran:

- Desarrollar e incorporar todos los componentes al producto final de software.
- Llevar a cabo todo tipo de pruebas sobre el producto final.

Para llevar a buen término estos objetivos, se deben clarificar los requerimientos pendientes, administrar los cambios de los artefactos construidos, ejecutar el plan de administración de recursos y corregir (si es necesario) el proceso de desarrollo del proyecto.

✓ transición

Durante esta etapa se debe asegurar la aceptación y disponibilidad del producto de software terminado por los usuarios finales, entre los objetivos principales de la misma se encuentran:

- Complementar el ciclo de pruebas del producto final.
- Generar la documentación de usuario y técnica.
- Traspasar el software desarrollado a la comunidad de usuarios
- Obtener autosuficiencia de parte de los usuarios
- Lograr el consenso cuanto antes para liberar el producto

En otras palabras, se debe asegurar que el software esté disponible para los usuarios finales, ajustar los errores y defectos encontrados, capacitar a los usuarios y proveer el soporte técnico necesario. Se debe verificar que el producto cumpla con las especificaciones entregadas por las personas involucradas en el proyecto al inicio del mismo.

2.4 Extreme Programming (XP)

La programación extrema es una nueva disciplina para la producción de software desarrollada por Kent Beck. Esta basada en la simplicidad, la comunicación, la retroalimentación y la refactorización de código. Algunas de sus características generales se mencionan a continuación.

- ✓ Pertenece al grupo de las metodologías ágiles.
- ✓ Incentiva las relaciones interpersonales como clave para su éxito.
- ✓ Promueve el trabajo en equipo.
- ✓ Procura el aprendizaje de los desarrolladores.
- ✓ Propicia un buen clima de trabajo.
- ✓ Se basa en retroalimentación continua entre el cliente y el equipo de desarrollo.
- ✓ Comunicación fluida entre todos los participantes.
- ✓ Simplicidad en las soluciones implementadas.
- ✓ Coraje para enfrentar los cambios.

La Programación Extrema se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. Pretende que el desarrollo de

un proyecto de software sea ágil, disciplinado y con soluciones sencillas. Además, esta metodología cuenta con un enfoque adaptativo, en la que la planificación del proyecto progresa a medida que surgen cambios.

La programación extrema define un conjunto de prácticas y valores (tradicionales) que permiten que el programador se dedique a lo que realmente hace mejor, programar. Elimina la necesidad de dedicarse a tareas tediosas y burocráticas, tales como: Exhaustivos documentos de proyecto, diagramas de Gantt, enormes listas de requerimientos, reuniones de revisiones interminables, etc.

2.4.1 Las Historias de Usuario

A mediados de la década de 1980, Kent Beck y Ward Cunningham trabajaban en un grupo de investigación de Tektronix; allí idearon las tarjetas CRC y sentaron las bases de lo que después serían los patrones de diseño y la programación extrema.

De estas tarjetas CRC nacen lo que ahora se conoce en la programación extrema como las historias de usuario. Se trata de simples tarjetas de papel para fichado, de 4x6 pulgadas; donde CRC denota “Clase-Responsabilidad-Colaboración”, y se generó como una técnica que reemplaza a los diagramas en la representación de modelos. En las tarjetas se escriben las responsabilidades, una descripción de alto nivel del propósito de una clase y las dependencias primarias.

El cliente junto al equipo de desarrollo definen qué es lo va hacer el sistema. Para ello utilizan las *historias de usuario*.

Una historia de usuario es un texto de una o dos frases en las que se explica brevemente algo que debe hacer el sistema. Es más extensa que un requisito (que suele ser una frase corta) y menos que un caso de uso[23] (que puede ser de una o dos cuartillas).

Las historias de usuario es la técnica utilizada en XP para especificar los requerimientos del software. Tomando como principio las tarjetas CRC, las historias son tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe contener. El manejo

[23] Caso de uso, consiste en una definición detallada de un requerimiento. Concepto perteneciente a UML.

de estas tarjetas puede ser muy dinámico y flexible, ya que en cualquier momento se pueden complementar para especificar mayor funcionalidad o simplemente desecharse.

Las historias de usuario son lo suficientemente comprensibles y delimitadas para que los programadores puedan implementarlas en unas semanas. El intervalo de tiempo ideal para su implementación debería ser de entre una y tres semanas. Aquellas que requieran menos tiempo son agrupadas, mientras que aquéllas que necesiten más tiempo deben ser modificadas o divididas.

Respecto a la información contenida en la historia de usuario, no existe un estándar que determine con exactitud la información que debe contener, se sugieren varias plantillas pero no existe un consenso al respecto. Por ejemplo: en algunos casos se propone utilizar únicamente un nombre, en otros una descripción y un nombre; y en algunos otros agregar a estos una estimación en días, propuesta por el desarrollador.

Beck en su libro[24] presenta un ejemplo de ficha (*customer story and task card*) en la cual pueden reconocerse los siguientes contenidos:

- Fecha
- Tipo de actividad (nueva, corrección, mejora)
- Prueba funcional
- Número de historia
- Prioridad técnica y del cliente
- Referencia a otra historia previa
- Riesgo
- Estimación técnica
- Descripción
- Notas
- Lista de seguimiento con la fecha
- Estado (cosas por terminar)
- Comentarios

[24] Kent Benk. Extreme Programming Explained: Embrace Change. Ed. Addison Wesley. 1999. E.U.A.

Por otro lado, hay que mencionar que el nivel de atomicidad de una historia de usuario depende directamente de la complejidad del sistema, es importante recordar que debe haber al menos una historia por cada característica importante.

Con respecto a la planificación, el tiempo de programación de las historias no debe exceder el tamaño de una iteración, es decir, debe ir de una a tres semanas. Como en todos los proyectos de producción de software, no siempre se identifican todos los requerimientos que el usuario pretende solicitar, no hay que preocuparse porque al inicio de cada iteración se tendrán registrada la historia de cambios en las historias de usuario y con base en estas se podrá planificar la siguiente iteración.

Las historias de usuario son descompuestas en tareas de programación y asignadas a los programadores para ser implementadas durante una iteración.

2.4.2 Roles

Una de las partes importantes dentro de esta metodología es la identificación de los roles que formaran parte en el proyecto de software y sus respectivas responsabilidades. Según la propuesta original de Beck, a continuación se describe estos roles:

- ✓ **Programador.** El programador escribe las pruebas unitarias y produce el código del sistema. Debe existir una comunicación y coordinación adecuada entre los programadores y otros miembros del equipo.

- ✓ **Cliente.** El cliente escribe las historias de usuario y las pruebas funcionales para validar su implementación; asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar mayor valor al negocio. El cliente es sólo uno dentro del proyecto pero puede corresponder a un interlocutor que está representando a varias personas que se verán afectadas por el sistema.

- ✓ **Encargado de pruebas (*Tester*).** El encargado de pruebas ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.

- ✓ **Encargado de seguimiento (*Tracker*).** El encargado de seguimiento proporciona realimentación al equipo en el proceso XP. Su responsabilidad es verificar el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, comunicando los resultados para mejorar futuras estimaciones. También realiza el seguimiento del progreso de cada iteración y evalúa si los objetivos son alcanzables con las restricciones de tiempo y recursos presentes. Determina cuándo es necesario realizar algún cambio para lograr los objetivos de cada iteración.

- ✓ **Entrenador (*Coach*).** Es responsable del proceso global. Es necesario que conozca a fondo el proceso XP para proveer guías a los miembros del equipo de forma que se apliquen las prácticas XP y se siga el proceso correctamente.

- ✓ **Consultor.** Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto. Guía al equipo para resolver un problema específico.

- ✓ **Gestor (*Big boss*).** Es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es de coordinación.

En ocasiones estos roles pueden mezclarse con la finalidad de brindar mejores resultados. Por ejemplo: Programador-Tracker, Programador-Tester, Cliente-Programador. Aunque también existen algunas combinaciones que nunca se van a dar por la naturaleza del rol.

2.4.3 Ciclo de vida del proceso de programación extrema

El ciclo de desarrollo consiste en los siguientes pasos:

1. El cliente define el valor de negocio a implementar.
2. El programador estima el esfuerzo necesario para su implementación.
3. El cliente selecciona qué construir, de acuerdo con sus prioridades y las restricciones de tiempo.
4. El programador construye ese valor de negocio.
5. Vuelve al paso 1.

En todas las iteraciones de este ciclo tanto el cliente como el programador aprenden. No se debe presionar al programador a realizar más trabajo que el estimado, ya que se perderá calidad en el software o no se cumplirán los plazos. De la misma forma el cliente tiene la obligación de manejar el ámbito de entrega del producto, para asegurarse que el sistema tenga el mayor valor de negocio posible con cada iteración. El ciclo de vida ideal de XP consiste de seis fases: Exploración, Planificación de la Entrega (*Reléase*), Iteraciones, Producción, Mantenimiento y Muerte del Proyecto[25].

- *Exploración.* En esta fase, los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología.

- *Planificación de la Entrega.* En esta fase el cliente establece la prioridad de cada historia de usuario, y correspondientemente, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Una entrega debería obtenerse en no más de tres meses. Esta fase dura unos pocos días. Las estimaciones de esfuerzo asociado a la implementación de las historias la establecen los programadores utilizando como medida el punto. Un punto, equivale a una semana ideal de programación. Las historias generalmente valen de 1 a 3 puntos. Por otra parte, el equipo de desarrollo mantiene un registro de la "velocidad" de desarrollo, establecida en puntos por iteración, basándose principalmente en la suma de puntos correspondientes a las historias de usuario que fueron terminadas en la última iteración. La planificación[26] se puede realizar basándose en el tiempo o el alcance (los objetivos que se pretenden). La velocidad del proyecto es utilizada para establecer cuántas historias se pueden implementar antes de una fecha determinada o cuánto tiempo tomará implementar un conjunto de historias. Al planificar por

[25] Abrahamsson, Pekka, Salo, Outi, Ronkainen, Jussi & Warsta, Juhani. Agile software development methods. Review and analysis. Espoo 2002. VTT Publications 478. p.19.
[26] "La planeación es el proceso de determinar objetivos y definir la mejor manera de alcanzarlos. Se ocupa, pues, de los medios, (como se debe hacer) y de los fines (que es lo que tiene que hacer)". Stephen P. Robbins. Administración. Ed.Prentice-Hall. P 114.

tiempo, se multiplica el número de iteraciones por la velocidad del proyecto, determinándose cuántos puntos se pueden completar. Al planificar según alcance del sistema, se divide la suma de puntos de las historias de usuario seleccionadas entre la velocidad del proyecto, obteniendo el número de iteraciones necesarias para su implementación.

- *Iteraciones.* Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. El Plan de Entrega está compuesto por iteraciones de no más de tres semanas. En la primera iteración se debe intentar establecer una arquitectura[27] del sistema que pueda ser utilizada durante el resto del proyecto. Esto se logra escogiendo las historias que forcé la creación de esta, sin embargo, no siempre es posible ya que es el cliente quien decide qué historias se implementarán en cada iteración (para maximizar el valor de negocio). Al final de la última iteración el sistema estará listo para entrar en producción. Los elementos que deben tomarse en cuenta durante la elaboración del Plan de la Iteración son: historias de usuario no abordadas, velocidad del proyecto, pruebas de aceptación no superadas en la iteración anterior y tareas no terminadas en la iteración anterior. Todo el trabajo de la iteración es expresado en tareas de programación, cada una de ellas es asignada a un programador como responsable, pero llevadas a cabo por parejas de programadores.

- *Producción***.** La fase de producción requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente. Al mismo tiempo, se deben tomar decisiones sobre la inclusión de nuevas características a la versión actual, debido a cambios durante esta fase. Es posible que se rebaje el tiempo que toma cada iteración, de tres a una semana. Las ideas que han sido propuestas y las sugerencias son documentadas para su posterior implementación (por ejemplo, durante la fase de mantenimiento).

[27] ."La arquitectura de software abarca los siguientes puntos: 1.-Las decisiones significativas sobre la organización de un sistema de software. 2.- La selección de los elementos estructurales y de las interfaces por los cuales esta compuesto el sistema, junto con su comportamiento según lo especificado en la colaboración entre ellos. 3.- La composición de estos elementos en subsistemas progresivamente más grandes; el estilo arquitectónico que dirige esta organización, estos elementos, y sus interfaces, sus colaboraciones, y su composición". Philippe Kruchten.The.rational unified process an introduction 3rd edition. Ed. Addison Wesley. 2003. E.U.A. p. 84.

- *Mantenimiento.* Mientras la primera versión se encuentra en producción, el proyecto XP debe mantener el sistema en funcionamiento al mismo tiempo que desarrolla nuevas iteraciones. Para realizar esto se requiere de tareas de soporte para el cliente. De esta forma, la velocidad de desarrollo puede bajar después de la puesta del sistema en producción. La fase de mantenimiento puede requerir nuevo personal dentro del equipo y cambios en su estructura.

- *Muerte del Proyecto.* Es cuando el cliente no tiene más historias para ser incluidas en el sistema. Esto requiere que se satisfagan las necesidades del cliente en otros aspectos como rendimiento y confiabilidad del sistema. Se genera la documentación final del sistema y no se realizan más cambios en la arquitectura. La muerte del proyecto también ocurre al perder al cliente, es decir, cuando el sistema no genera los beneficios esperados por el mismo. Y en algunas ocasiones cuando no hay presupuesto para mantenerlo.

2.4.4 Prácticas

La principal suposición que se realiza en la programación extrema, es la posibilidad de disminuir la curva exponencial del costo del cambio a lo largo del proyecto, lo suficiente para que el diseño evolutivo funcione. XP apuesta por un crecimiento lento del costo del cambio y con un comportamiento asintótico. Esto se consigue gracias a las tecnologías disponibles para ayudar en el desarrollo de software y a la aplicación disciplinada de las prácticas que se describen a continuación:

- *El juego de la planificación (the planning game).* Es un espacio frecuente de comunicación entre el cliente y los programadores. El equipo técnico realiza una estimación del esfuerzo requerido para la implementación de las historias de usuario y los clientes deciden sobre el ámbito y tiempo de las entregas y de cada iteración. Esta práctica se puede ilustrar como un juego, donde existen dos tipos de jugadores: Cliente y Programador. El cliente establece la prioridad de cada historia de usuario, de acuerdo con el valor que aporta para el negocio. Los programadores estiman el esfuerzo asociado a cada historia de usuario. Se ordenan las historias de usuario según prioridad y esfuerzo, y se define el contenido de la entrega y/o iteración, apostando por enfrentar lo de más valor y riesgo cuanto antes. Este juego se

realiza durante la planificación de la entrega, en la planificación de cada iteración y cuando sea necesario reconducir el proyecto.

- *Pequeñas entregas (small releases).* La idea es producir rápidamente versiones del sistema que sean operativas, aunque obviamente no cuenten con toda la funcionalidad pretendida para el sistema pero si que constituyan un resultado de valor para el negocio. Una entrega debe estar entre un lapso de tiempo de un día y a más tardar dos meses[28].
- *Metáfora (metaphor).* En XP no se enfatiza la definición temprana de una arquitectura estable para el sistema. Dicha arquitectura se asume evolutiva y los posibles inconvenientes que se generarían por no contar con ella explícitamente en el comienzo del proyecto se solventan con la existencia de una metáfora. El sistema es definido mediante una metáfora o un conjunto de metáforas compartidas por el cliente y el equipo de desarrollo. Una metáfora es una historia compartida que describe cómo debería funcionar el sistema.
- *Diseño simple (simple design).* Se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto. La complejidad innecesaria y el código extra debe ser removido inmediatamente.
- *Pruebas (testing).* La producción de código está dirigida por las pruebas unitarias. Las pruebas unitarias son establecidas antes de escribir el código y son ejecutadas constantemente ante cada modificación del sistema. Los clientes escriben las pruebas funcionales para cada historia de usuario que deba validarse. En este contexto de desarrollo evolutivo y de énfasis en pruebas constantes, la automatización para apoyar esta actividad es crucial.
- *Refactorización (refactoring).* La refactorización es una actividad constante de reestructuración del código con el objetivo de remover duplicación de código, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar los posteriores cambios. La refactorización mejora la estructura interna del código sin alterar su comportamiento externo. No se puede imponer todo en un inicio, pero en el transcurso del tiempo este diseño evoluciona conforme cambia la funcionalidad del sistema.

[28] Los autores de la programación extrema piensan que la generación de versiones debe ser lo más cortas posibles. Consideran que mientras mas pequeñas, menor es el riesgo adquirido por lo trabajado. Ellos proponen la generación de versiones en periodos que van desde un día hasta dos meses. (Kent Benk. Extreme Programming Explained: Embrace Change. Ed. Addison Wesley. 1999. E.U.A. Extreme Programming Installed Ed. Addison Wesley. 2000. E.U.A.)

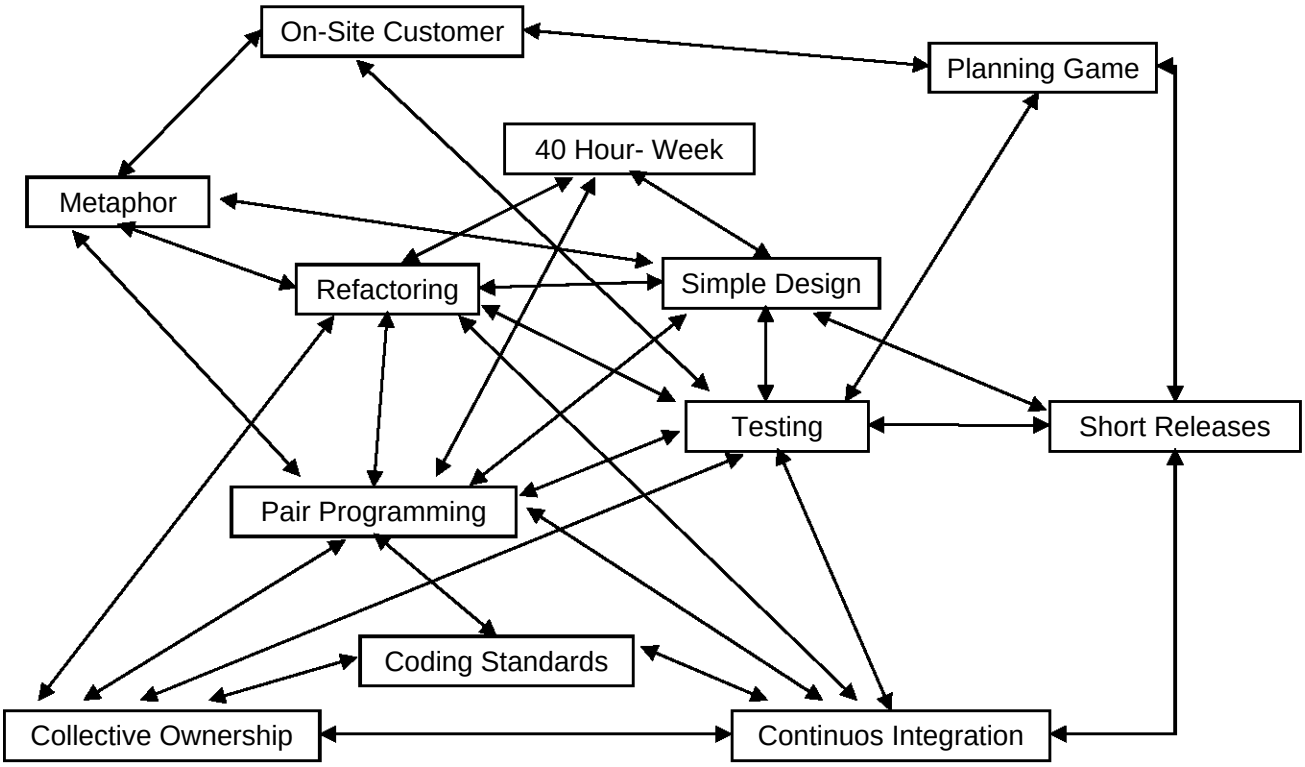
- *Programación por parejas (pair programming).* Toda la producción de código debe realizarse con trabajo en parejas de programadores. Esto provoca que halla retroalimentación entre ellos y la discusión de ideas y soluciones de cómo resultado un producto final de mayor calidad. Los diseños son mejores, el tamaño del código es menor, los problemas de programación se resuelven mas rápido, hay transferencia de conocimientos, se mejora la dinámica del equipo, etc.
- *Propiedad colectiva (collective ownership).* Todos los programadores tiene acceso para cambiar parte del código en cualquier momento. Esta práctica motiva a todos a contribuir con nuevas ideas en todos los segmentos del sistema, evitando a la vez que algún programador sea imprescindible para realizar cambios en alguna porción de código.
- *Integración continua (continous integration).* Cada pieza de código es integrada y probado en cortos tiempos, al final del día en la mayoría de las ocasiones o una vez que esté lista la funcionalidad. Así, el sistema puede llegar a ser integrado y construido varias veces en un mismo día. Todas las pruebas son ejecutadas y tienen que ser aprobadas para que el nuevo código sea incorporado definitivamente. La integración continua a menudo reduce la fragmentación de los esfuerzos de los desarrolladores por falta de comunicación sobre lo que puede ser reutilizado o compartido.
- *40 horas semanales (40-hour week).* Se debe trabajar un máximo de 40 horas por semana. No se trabajan horas extras en dos semanas seguidas. Esto es un síntoma de un serio problema en el proyecto. El trabajo extra desmotiva al equipo. Los proyectos que requieren trabajo extra para intentar cumplir con las metas suelen al final ser entregados con retraso. En lugar de esto se puede realizar el juego de la planificación para cambiar el ámbito del proyecto o la fecha de entrega.
- *Cliente en casa (on-site costumer).* El cliente tiene que estar presente y disponible todo el tiempo para el equipo. Gran parte del éxito del proyecto XP se debe a que es el cliente quien conduce constantemente el trabajo hacia lo que aportará mayor valor de negocio y los programadores pueden resolver de manera inmediata cualquier duda asociada. La comunicación oral es más efectiva que la escrita, ya que esta última toma mucho tiempo en generarse y puede tener más riesgo de ser mal interpretada. Según Kent Benk[29], un verdadero cliente debe sentarse con el equipo, disponible para contestar preguntas, resolver dudas, y asignar las prioridades para el sistema.

[29] Kent Benk. Extreme Programming Explained: Embrace Change. Ed. Addison Wesley. 1999. E.U.A.

- *Estándares de codificación (coding standards).* XP enfatiza la comunicación de los programadores a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación (del equipo, de la organización u otros estándares reconocidos para los lenguajes de programación utilizados). Los estándares de programación mantienen el código legible para los miembros del equipo, facilitando los cambios.

El mayor beneficio de las prácticas se consigue con su aplicación conjunta y equilibrada puesto que se apoyan unas en otras. A continuación se muestra la figura 2.4.1, en la cual se ejemplifica la relación entre las practicas propuesta por XP; donde una línea entre dos prácticas significa que las dos prácticas se refuerzan entre sí. La mayoría de las prácticas propuestas por XP no son novedosas sino que en alguna forma ya habían sido propuestas en ingeniería del software e incluso demostrado su valor en la práctica. El mérito de XP es integrarlas de una forma efectiva y complementarlas con otras ideas desde la perspectiva del negocio, los valores humanos y el trabajo en equipo.

Iteración de las prácticas de la programación extrema[30] (Figura 2.4.1)



[30] Kent Benk. Extreme Programming Explained: Embrace Change. Ed. Addison Wesley. 1999. E.U.A. p.59

La figura anterior muestra la relación entre las prácticas, la línea entre ellas significa que son complementarias.

2.4.5 Valores

XP se fundamenta en cuatro valores o principios que lo inspiran:

- ✓ *Comunicación.* XP pone en comunicación directa y continua a clientes y desarrolladores. El cliente se integra en el equipo para establecer prioridades y resolver dudas. De esta forma ve el avance día a día, y es posible ajustar el plan de trabajo y las funcionalidades de forma consecuente.

- ✓ *Feedback rápido y continuo.* Una metodología basada en el desarrollo incremental de pequeñas partes, con entregas y pruebas frecuentes y continuas, proporciona un flujo de retro-información valiosa para detectar los problemas o desviaciones. De esta forma fallos se localizan muy pronto. La planificación no puede evitar algunos errores, que sólo se evidencian al desarrollar el sistema. La retro-información es la herramienta que permite reajustar la agenda y los planes.

- ✓ *Simplicidad.* La simplicidad consiste en desarrollar sólo el sistema que realmente se necesita. Implica resolver en cada momento sólo las necesidades actuales. "*Los costes y la complejidad de predecir el futuro son muy elevados, y la mejor forma de acertar es esperar al futuro.*" Con este principio de simplicidad, junto con la comunicación y el feedback resulta más fácil conocer las necesidades reales.

- ✓ Coraje. El coraje implica saber tomar decisiones difíciles. Reparar un error cuando se detecta. Mejorar el código siempre que tras el feedback y las sucesivas iteraciones se manifieste susceptible de mejora. Tratar rápidamente con el cliente los desajustes de agendas para decidir qué partes y cuándo se van a entregar.

2.5 Microsoft Solution Framework (MSF)

Es un conjunto de modelos, conceptos y guías para la implementación de sistemas de información empresariales en un ambiente distribuido. MSF contribuye a alinear los objetivos de negocio y tecnológicos, reducir los costos de la utilización de nuevas tecnologías, y asegurar el éxito en la

implementación de las tecnologías Microsoft. MSF provee la guía para las fases de planeación, construcción y deploy del ciclo de vida del proyecto[31].

MSF es altamente parametrizable, escalable, completamente integrado a un conjunto de procesos de desarrollo de software, principios, y prácticas diseñadas para proporcionar el tipo de dirección deseada por el usuario cuando y donde esta sea necesaria. MSF también proporciona las prácticas probadas para la planeación, diseño, desarrollo, y liberaciones exitosas en soluciones empresariales.

MSF proporciona un conjunto de guías de desarrollo de software parametrizable y escalable para mejorar el desarrollo de aplicaciones. MSF soporta acercamiento de procesos múltiples, que permite que el usuario seleccione la trayectoria más conveniente. El framework flexible de MSF se puede adaptar para resolver las necesidades de cualquier proyecto, sin importar tamaño o complejidad. La filosofía de MSF sostiene que no hay estructura o proceso que se aplique de manera óptima a los requerimientos y a los ambientes para todos los proyectos. También reconoce que existe la necesidad de la dirección. MSF proporciona esta dirección de una forma no impositiva y permite que el usuario modifique el contenido para requerimientos particulares. Los componentes de MSF pueden ser aplicados individualmente o colectivamente para mejorar las tarifas del éxito en los diferentes tipos de proyectos.

La visión de MSF es proporcionar la guía de procesos desarrollada para los profesionales del software por los profesionales del software que es productiva, integrada, y extensible.

- Productivo: Una de las visiones dominantes de MSF es hacer gente más productiva. La productividad es soportada a través de la presentación de la guía de procesos de una forma aerodinámica y parametrica. Usando listas de comprobación y pautas en vez de contenido detallado, el usuario puede determinar rápidamente los requerimientos para terminar una tarea o una actividad.

- Integrado: Las soluciones y la guía se presentan con la herramienta, vía la integración completa del conjunto de herramientas, la integración de la ayuda y del contenido de MSF. Todos estos elementos son fácilmente actualizados con MSDN y a través de todos los

[31] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnvsent/html/vsts-msf.asp

aspectos del conjunto de herramientas. El contenido en sí mismo es organizado para el fácil mantenimiento.

- Extensible: La guía de procesos y la ayuda son completamente parametrizables dentro de MSF. Los usuarios eligen el acercamiento ágil o más estructurado, incorporan escenarios de desarrollo, y determinan su propia trayectoria a través del contenido.

El Visual Studio 2005 Team System incluye dos plantillas de la metodología de MSF, que se pueden utilizar "como está", parametrizarlo para sus requerimientos particulares, o utilizar como base para crear su propia plantilla de proceso:

- MSF for Agile Software Development
- MSF for CMMI® Process Improvement

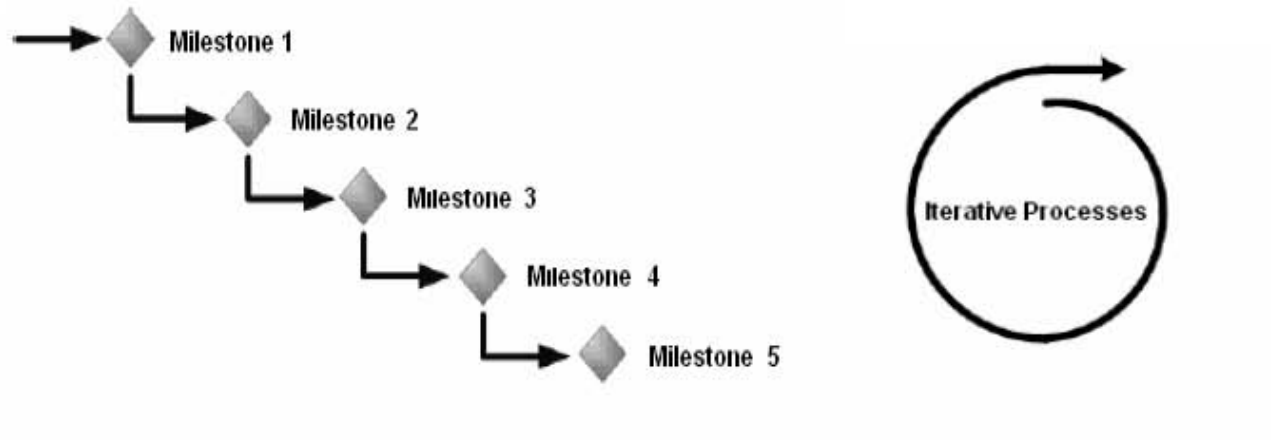
MSF consiste en siete modelos, que pueden ser usados individualmente o combinados entre si. Estos modelos son:

- Team Model (Modelo de Equipo)
- Process Model (Modelo de Procesos)
- Application Model (Modelo de Aplicación)
- Solution Design Model (Modelo de Diseño de Soluciones)
- Enterprise Architecture Model (Modelo de Arquitectura Empresarial)
- Infraestructure Model (Modelo de Infraestructura)
- Total Cost Ownership Model (Modelo de Costo Total de Propiedad)

Modelo de procesos

Un modelo del proceso guía el orden de las actividades del proyecto y representa el ciclo de vida de un proyecto. Históricamente, algunos modelos de proceso son estáticos y otros no permiten puntos de comprobación. Dos de tales modelos de proceso son el modelo de cascada y el modelo espiral. La figura 2.5.1 muestra los modelos en cascada y espiral.

Modelos en cascada y espiral[32] (Figura 2.5.1)



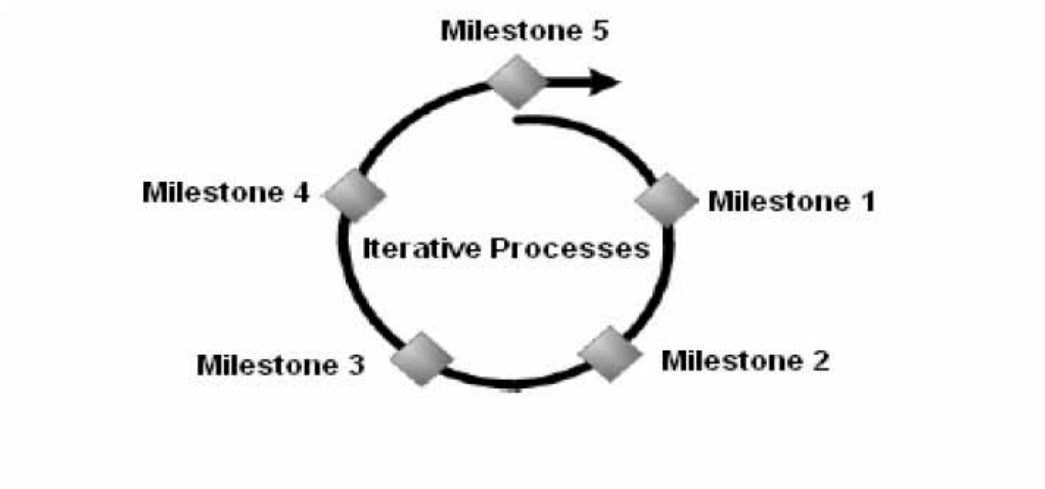
Modelo en cascada. Este modelo utiliza liberaciones significativas como puntos de transición y gravamen. Cuando se usa el modelo en cascada, se requiere concluir el conjunto de tareas de esta fase antes de moverse a la siguiente. Este modelo trabaja mejor para proyectos en el cual los requerimientos pueden ser claramente definidos y no son propensos a modificaciones en el futuro. Debido a que este modelo tiene puntos de transición fijos entre fases, se puede fácilmente monitorear el calendario de tareas y responsabilidades.

Modelo en espiral. Este modelo esta basado sobre la continua necesidad de refinar los requerimientos y los estimados del proyecto. El modelo de espiral es efectivo cuando se utiliza para el rápido desarrollo de aplicaciones en proyectos pequeños. Este acercamiento puede generar gran sinergia entre el equipo del desarrollo y el cliente porque el cliente es implicado en todas las etapas proporcionando la retroalimentación y la aprobación. Sin embargo, el modelo espiral no incorpora puntos de comprobación claros. Por lo tanto, el proceso del desarrollo puede llegar a ser caótico.

Modelo de Proceso MSF. Este modelo de proceso combina los mejores principios de los modelos en cascada y espiral. Este combina las liberaciones significativas del modelo en cascada, basado en la planeación y resultados predictivos, con los beneficios del modelo en espiral de regeneración y creatividad.

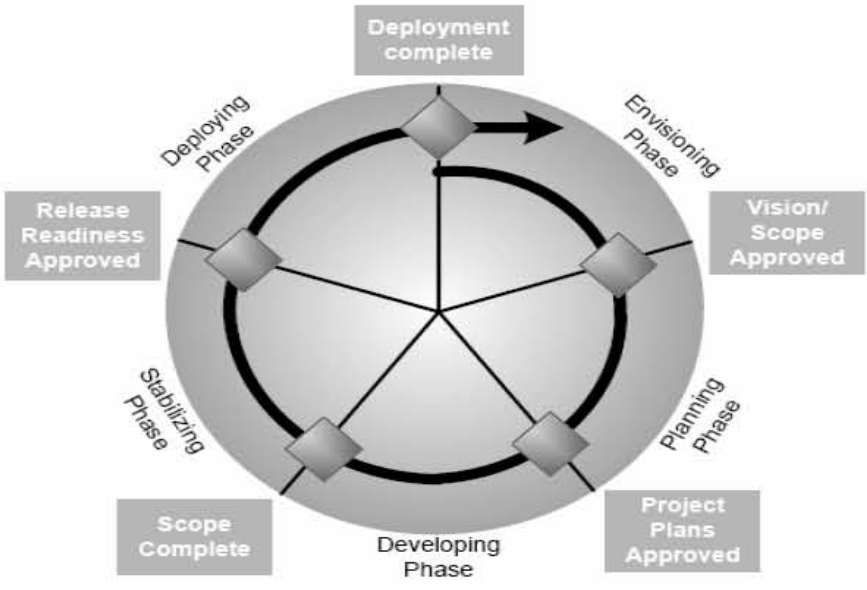
[32] MSF Team, Microsoft. Microsoft Solutions Framework. White Paper. MSF Process Model v.3.1. Publicado: junio 2002. pp. 5-6. http://www.microsoft.com/technet/itsolutions/msf/default.mspx

Modelo de proceso MSF[33] (Figura 2.5.2)



Fases MSF. El modelo de proceso de MSF es un modelo basado en fases, liberaciones significativas, y en un modelo iterativo que se puede aplicar al desarrollo y liberación en aplicaciones tradicionales, soluciones empresariales para el comercio electrónico (e-commerce), y en aplicaciones de Web distribuidas.

Fases MSF[34] (Figura 2.5.3)



[33] MSF Team, Microsoft. Microsoft Solutions Framework. White Paper. MSF Process Model v.3.1. Publicado: junio 2002. p. 6. http://www.microsoft.com/technet/itsolutions/msf/default.mspx
[34] MSF Team, Microsoft. Microsoft Solutions Framework. Op. cit. p. 23.

MSF contempla 5 distintas fases, cada una de las cuales termina en una liberación significativa.

Previsión. (Envisioning) La previsión puede ser definida como la creación de una amplia descripción de las metas y restricciones del proyecto. En esta fase, se identifica el equipo y lo que se debe lograr para el cliente. El propósito de la fase de previsión es construir una visión compartida del proyecto entre todos los integrantes del proyecto.

Durante la fase de la previsión, el equipo de la gerencia del programa identifica las tareas y los entregables que contienen los requisitos y las metas del proyecto. Esta fase culmina en una visión/alcance aprobada en una liberación significativa. Esta liberación indica que el cliente y el equipo están de acuerdo con el propósito y la dirección del proyecto.

Planeación. (Planning) Durante la fase del planeación, el equipo determina qué va a desarrollar y planea cómo crear la solución. El equipo prepara la especificación funcional, crea un diseño de la solución, y prepara planes de trabajo, valoraciones de costos, y horarios para entregables.

La fase del planeación implica el análisis de requerimientos. Estos requerimientos se pueden categorizar como del negocio, exigencias del consumidor, operacionales, y del sistema. Estos requerimientos se utilizan para diseñar la solución y sus características, como para validar la corrección del diseño.

Desarrollo. (Developing) Durante la fase que se desarrollo, el equipo de proyecto crea la solución. Este proceso incluye la generación del código que implementa la solución y la documentación del mismo. Además del código que se genera, el equipo también desarrolla la infraestructura para la solución.

Estabilización. (Stabilizing) Durante la fase de estabilización, el equipo realiza la integración, la carga, y las pruebas beta sobre la solución. Además, el equipo prueba los diferentes escenarios desarrollados para la solución. El equipo se centra en identificar, dar la prioridad, y resolver los errores para preparar la solución para la liberación. Durante esta fase, la solución debe ir cumpliendo con las características según lo definido en la especificación funcional para la versión, hasta cumplir con los niveles de calidad establecidos. Además, la solución debe estar lista para la liberación.

Liberación. (Deploying) Durante esta fase el equipo libera la solución tecnológica y el equipo de componentes, estabiliza la liberación, transfiere el proyecto al equipo de operaciones y soporte, y obtiene la aprobación final del proyecto por parte del cliente.

2.6 Otras metodologías importantes

Al igual que las metodologías presentadas en los apartados anteriores, existen algunas más que pueden servir como complemento de las mismas y enriquecer el funcionamiento del proceso de producción de software. A continuación se describen algunas de ellas con sus aspectos principales.

2.6.1 Crystal family

Fueron creadas por *Alistair Cockburn*. El término *cristal* es una referencia a las facetas de una gema, cada una de las cuales da una vista a la misma cosa (la gema), pero cada vista es diferente. Es una familia porque tipos diferentes de proyectos requieren tipos diferentes de metodologías crystal. Se mira esta variación a lo largo de dos ejes: el número de personas en el proyecto, y las consecuencias de los errores. La familia Crystal dispone un código de color para marcar la complejidad de una metodología: cuanto más oscuro un color, más *pesado* es el método. Cuanto más crítico es un sistema, más rigor se requiere. Se evalúa el proyecto según las pérdidas que puede ocasionar la falla de un sistema y el método requerido según este criterio. Los parámetros son Comodidad (C), Dinero Discrecional (D) (aumento de costo produciendo pérdidas salvables), Dinero Esencial (E) (pérdidas total de dinero) y Vidas (L).

2.6.1.1 Valores y propiedades

La metodología Crystal Family se compone de los siguientes valores y propiedades:

- ✓ *Entrega frecuente*. Entrega de software lo más pronto posible; diario, semanal o mensual dependiendo del proyecto.
- ✓ *Comunicación*. Todos juntos en el mismo cuarto. Una variante especial es disponer en la sala de un diseñador señior. Realizar reuniones en forma separada para que los concurrentes se concentren mejor.

- ✓ *Mejora reflexiva*. Tomarse un pequeño tiempo para analizar, reflexionar, discutir.
- ✓ *Seguridad personal*. Hablar cuando algo molesta, decirle amigablemente al manager que la agenda no es realista, o a un colega que su código necesita mejorarse.
- ✓ *Foco*. Saber lo que se está haciendo y tener la tranquilidad y el tiempo necesario para hacerlo.
- ✓ *Fácil acceso a usuarios expertos*. Siempre debe estar disponible un usuario experto, preparado para aclarar cualquier duda de negocio. Se propone un encuentro semanal, sin embargo esto depende de las características del proyecto.
- ✓ *Ambiente técnico con prueba automatizada, management de configuración e integración frecuente*. Muchos equipos ágiles compilan e integran varias veces al día.

2.6.1.2 Estrategias a seguir

Las estrategias a seguir para el proceso de producción de un software, por parte de la metodología Cystal Family se listan a continuación:

- *Exploración de 360°*. Verificar o tomar una muestra del valor de negocios del proyecto, los requerimientos, el modelo de dominio, la tecnología, el plan del proyecto y el proceso. La exploración es preliminar al desarrollo y esto debe consumir unos pocos días; a lo más dos semanas.

- *Victoria temprana*. Usualmente la primera victoria temprana consiste en la construcción de un esqueleto ambulante. La preferencia de Cockburn es “lo más fácil primero, lo más difícil segundo”.

- *Esqueleto ambulante*. Una transacción que debe ser simple pero completa. Podría ser una rutina de consulta y actualización en un sistema cliente-servidor, o la ejecución de una transacción en un sistema transaccional de negocios. Carece de parte de la funcionalidad, que se agregará incrementalmente. Debe producirse con buenos hábitos de producción y pruebas, y está destinado a crecer con el sistema.

- *Re-Arquitectura incremental*. La arquitectura debe evolucionar en etapas, manteniendo el sistema en ejecución mientras ella se modifica.

- *Radiadores de información*. Es una representación de las actividades a llevar a cabo en el proyecto y usualmente se representa en una lámina pegada en algún lugar visible para el conjunto del equipo. Esta podría mostrar el conjunto de la iteración actual, el número de pruebas pasadas o pendientes, el número de casos de uso o historias entregado, etc.

2.6.1.3 Técnicas utilizadas

- *Entrevistas de proyectos*. Se suele entrevistar a más de un responsable para tener visiones más ricas. La idea es averiguar cuales son las prioridades, obtener una lista de rasgos deseados, saber cuáles son los requerimientos más críticos y cuáles los más negociables. Si se trata de una actualización o corrección, saber cuáles son las cosas que se hicieron bien y merecen preservarse y los errores que no se quieren repetir.

- *Talleres de reflexión*. El equipo debe detenerse treinta minutos o una hora para reflexionar sobre sus convenciones de trabajo, discutir inconvenientes y mejoras a planear para el período siguiente.

- *Planeamiento Blitz*. Se ponen tarjetas indexadas en una mesa, con una función visible en cada una. El grupo finge que no hay dependencias entre tarjetas, y las alinea en secuencias de desarrollo preferidas. Los programadores escriben en cada tarjeta el tiempo estimado para desarrollar cada función. El patrocinador o embajador del usuario escribe la secuencia de prioridades, teniendo en cuenta los tiempos referidos y el valor de negocio de cada función. Las tarjetas se agrupan en períodos de tres semanas llamados iteraciones que se agrupan en entregas (reléase), usualmente no más largas de tres meses.

- *Estimaciones de pericia*. Se reúnen los expertos responsables y proceden como en una subasta para proponer el tamaño del sistema, su tiempo de ejecución, la fecha de las entregas según dependencias técnicas y de negocios, y para equilibrar las entregas en paquetes de igual tamaño.

- *Encuentros diarios de pie*. La palabra clave es *brevedad*, cinco a diez minutos como máximo. No se trata de discutir problemas, sino de identificarlos. Los problemas sólo se discuten en otros encuentros posteriores, con la gente que tiene que ver en ellos.

- *Miniatura de procesos*. Una forma de presentar Crystal Clear puede estar entre 90 minutos y un día. La idea es que la gente pueda entender la nueva metodología.

- *Gráficos de quemado*. Técnica para descubrir demoras y problemas tempranamente en el proceso, evitando que se descubra demasiado tarde, cuando aun no se sabe cuánto falta. Ilustran la velocidad del proceso, analizando la diferencia entre las líneas proyectadas y efectivas de cada entrega, como se ilustra en las figuras.

- *Programación lado a lado*. La versión de Crystal Clear establece proximidad entre programadores, pero cada quien se aboca a su trabajo asignado, prestando atención a lo que hace su compañero, quien tiene su propia máquina.

2.6.1.4 Roles

1) *Patrocinador*. Produce el plan que se debe desarrollar en base a prioridades de compromiso establecidas con el cliente. Consigue los recursos y define la totalidad del proyecto.
2) *Usuario Experto*. Junto con el experto en negocios produce la Lista de Actores-Objetivos y el Archivo de Casos de Uso y Requerimientos. Debe familiarizarse con el uso del sistema, sugerir atajos de teclado, modos de operación, información a visualizar simultáneamente, navegación, etc.
3) *Diseñador Principal*. Produce la descripción arquitectónica. Debe ser al menos un profesional que sea capaz de manejar con fluidez, mezclar e inventar procedimientos. El Diseñador Principal tiene roles de coordinador, arquitecto, mentor y programador más experto.
4) *Diseñador-Programador*. Produce, junto con el diseñador principal, los borradores de pantallas, el modelo común de dominio, las notas y diagramas de diseño, el código fuente, el código de migración, las pruebas y el sistema empaquetado. Cockburn no distingue entre diseñadores y programadores.
5) *Experto en Negocios*. Junto con el usuario experto produce la lista de Actores-Objetivos y el archivo de casos de uso y requerimientos. Debe conocer las reglas y políticas del negocio.

6) *Coordinador*. Con la ayuda del equipo, produce el mapa de proyecto, el plan de entrega, el estado del proyecto, la lista de riesgos, el plan y estado de la iteración y la agenda de visualización.
7) *Verificador*. Produce el reporte de errores (*bugs)*. Puede ser un programador en tiempo parcial, o un equipo de varias personas.
8) *Escritor*. Elabora documentación externa (Manuales técnicos y de usuario), apoyándose en las áreas involucradas en la producción del software.

2.6.1.5 Ciclo de vida

Crystal Clear enfatiza el proceso como un conjunto de ciclos anidados. En la mayoría de los proyectos se perciben siete ciclos:

1) El proyecto,
2) El ciclo de entrega de una unidad,
3) La iteración,
4) La semana laboral,
5) El período de integración, de 30 minutos a tres días,
6) El día de trabajo,
7) El episodio de desarrollo de una sección de código, de pocos minutos a pocas horas.

2.6.2 Código Abierto (Open Source)

El paradigma de la metodología del código abierto siguiere que el código fuente sea libremente disponible para realizarle modificaciones y redistribuirlo sin ningún cargo adicional.

El proceso se caracteriza por trabajar con equipos físicamente distribuidos, lo que es importante porque la mayoría de los procesos adaptables exigen equipos locales. Los proyectos de código abierto por lo general cuentan con uno o más mantenedores. El rol de mantenedor se define como la única persona que es la encargada de integrar cambios a un repositorio de código fuente.

Una característica particular del desarrollo de código abierto es que la depuración se puede realizar en paralelo. Muchas personas pueden involucrarse en el depurado. Cuando se encuentra

un error pueden enviar la corrección al mantenedor y este se encarga de integrarla. Esto es un buen papel para los no mantenedores ya que la mayor parte del tiempo se les consume en la búsqueda de errores. También es bueno para gente sin mucha destreza en programación, pues es quien por lo general toma este rol. Una de las herramientas más flexibles para llevar a cabo esta actividad es CVS (version control system), la cual administra el código fuente del proyecto permitiendo retornar a versiones anteriores.

2.6.3 Scrum

Fue desarrollado por *Ken Schwaber* y *Jeff Sutherland*. La palabra Scrum procede de la terminología del juego de rugby, donde designa al acto de preparar el avance del equipo en unidad pasando la pelota a uno y otro jugador. Igual que el juego, Scrum es adaptativo, ágil, auto-organizable y con pocos tiempos muertos.

Scrum no está concebido como método independiente, sino que se promueve como complemento de otras metodologías, incluyendo XP, MSF entre otras. Scrum esta orientada al manejo del proceso de desarrollo de los sistemas, enfatiza en valores y prácticas de gestión. No es una metodología especializada en requerimientos, implementación y demás cuestiones técnicas.

"Scrum se define como un proceso de administración y control que aplica las ideas de la teoría de control de procesos al desarrollo de sistemas, haciendo un acercamiento a las ideas de flexibilidad, adaptabilidad y productividad[35]". En conclusión se le puede considerar como un conjunto de patrones organizacionales que permiten llevar a cabo de una mejor forma la administración de un proyecto de software.

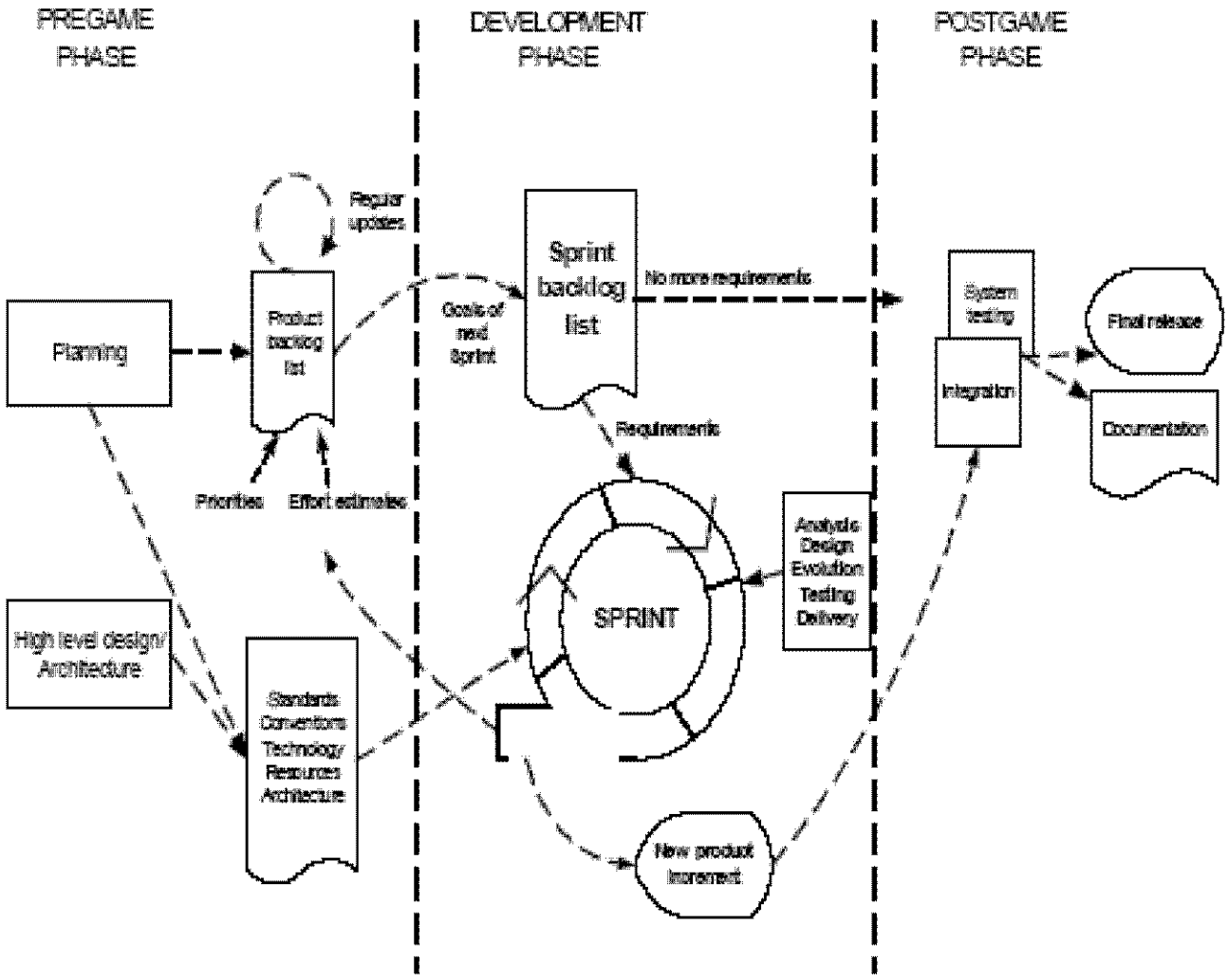
2.6.3.1 Ciclo de vida

Scrum trabaja en ciclos, en *sprints* de 30 días, los cuales son un conjunto de tres etapas donde se planea, construye y monitorea el desarrollo de un subconjunto del trabajo total. Las tres etapas son llamadas:

[35] Abrahamsson, Pekka, Salo, Outi, Ronkainen, Jussi & Warsta, Juhani. Agile software development methods. Review and analysis. Espoo 2002. VTT Publications 478. p.27

- *Fase de Pre-Game o también conocida como pre-Sprint.*
- *Fase de Desarrollo o también conocida como Sprint.*
- *Fase de Post-Game o también conocida como Post-Sprint*.

Ciclo de vida de Scrum (Figura 2.6.1)



Pre-Game. Como se puede ver en la figura 2.6.2 esta fase esta compuesta de dos actividades, la planeación y el diseño de la arquitectura a alto nivel.

- La planeación incluye la definición del sistema a ser desarrollado, el equipo del proyecto, herramientas y otros recursos, identificación de riesgos, control de incidencias, necesidades de capacitación y verificación de aprobación de la gerencia. Un *Product Backlog* (lista de retraso o acumulación) es la fuente principal de planeación. Esta "lista de retraso" es creada en esta fase con todos los requerimientos que son actualmente conocidos. Estos requerimientos pueden originarse de varios puntos, tales como los clientes, los departamentos de ventas y mercadotecnia, el área de ayuda a clientes o por los desarrolladores. La lista puede contener funciones del usuario y elementos de tecnología y arquitectura. Los requerimientos son priorizados y es estimado el esfuerzo para su implementación. La lista de *Product Backlog* es constantemente actualizada con nuevos y mas detalles de los requerimientos.

  El Product Owner es el responsable del Product backlog, este puede ser una persona o un equipo representando por varios clientes. El Product Owner revisa y da prioridad al conjunto de características en el *Product Backlog* para crear el *Reléase Backlog*. Este identifica el trabajo que el equipo realizará por el siguiente ciclo de 30 días. Esta lista es usada para planear las tareas para el equipo de desarrollo. Al mismo tiempo que el *Reléase Backlog* es creado, el equipo crea un *Sprint Goal,* el cual es una lista con los puntos mínimos a cubrir en el ciclo y que están contemplados dentro del *Reléase Backlog*. Al adoptar el *Sprint Goal* se evita que el equipo se enfoque exclusivamente en tareas detalladas. Esto implica que halla flexibilidad en el plan de trabajo, "no cumplimos con todas las tareas pero cumplimos con el *Sprint Goal*". Es tan importante ya que mantiene al equipo enfocado en entregar algo que tenga valor al cliente.

Desarrollo (Sprint). Los miembros de equipo se hacen responsables por las tareas del *Sprint* de 30 días, y se espera que el equipo se maneje a sí mismo. La principal característica administrativa del *Sprint* es la reunión diaria. Esta es una reunión informal de planeación y reporte de progreso que se lleva a cabo todos los días a la misma hora. Debe ser corta y concreta. Todos los involucrados, desarrolladores y usuarios, deben presentarse. La reunión es dirigida por el Scrum Master, quien esta a cargo del *Sprint* y su función es, al menos parcialmente, es controlar los cambios. En general el *Sprint* está cerrado a cambios una vez que se acuerda el *Reléase Backlog*. Este es un intento para balancear la necesidad de cambio continuo en un ambiente continuamente cambiante con la necesidad del equipo de lograr estabilidad durante el desarrollo.

Post-Game. Al final de cada ciclo de 30 días, la reunión Post-Sprint revisa los logros y fracasos. También es una reunión donde la funcionalidad ya completa puede ser presentada. El final de esta reunión da paso al principio del siguiente ciclo de 30 días.

2.6.3.2 Valores

- Equipos auto-dirigidos y auto-organizados. No hay manager que decida, ni otros títulos que "miembros del equipo"; la excepción es el Scrum Master que debe ser 50% programador y que resuelve problemas, pero no manda.
- Una vez elegida una tarea, no se agrega trabajo extra. En caso que se agregue algo, se recomienda quitar alguna otra cosa.
- Encuentros diarios con las tres preguntas:
  ¿Qué has hecho desde el último encuentro?
  ¿Qué obstáculos hay para cumplir la meta?
  ¿Qué harás antes del próximo encuentro?
- Iteraciones de treinta días; se admite que sean más frecuentes.
- Demostración a participantes externos al fin de cada iteración.
- Al principio de cada iteración, planeamiento adaptativo guiado por el cliente.

2.6.3.3 Roles

Existen cinco roles con sus respectivas responsabilidades en Scrum y a continuación se describen:

Scrum Master. Interactúa con el cliente y el equipo. Es responsable de asegurarse que el proyecto se lleve a cabo de acuerdo con las prácticas, valores y reglas de Scrum y que progrese según lo previsto. Se encarga de eliminar eventuales obstáculos que impidan la producción del equipo de trabajo.
Product Owner. Es el responsable oficial de proyectar, administrar, controlar y publicar la lista de acumulación del producto (*product backlog*). Es elegido por el Scrum Master, el cliente y los ejecutivos a cargo. Toma las decisiones finales de las tareas asignadas al registro y realiza las estimaciones de tiempo de desarrollo por recursos.
Scrum Team. Se refiere al equipo de trabajo del proyecto y tiene la autoridad para organizar y definir las acciones necesarias para alcanzar las metas de cada Sprint.

Client. Participa en las tareas relacionadas con el registro de la lista de acumulación del producto (backlog).
Manager. Está a cargo de las decisiones fundamentales y participa en la definición de los objetivos y requerimientos. Por ejemplo, selecciona al Project Owner, evalúa el progreso y reduce el registro de acumulación junto con el Scrum Master.

2.6.3.4 Prácticas

Producto Backlog (Lista de retraso o acumulación). El producto backlog es una lista donde se describen las necesidades o características que debe cubrir el producto final, es decir se define el trabajo a ser realizado en el proyecto. Esta lista es priorizada y actualizada de forma constante con los requerimientos técnicos y de negocio necesarios para la construcción del producto a lo largo de cada Sprint. Por ejemplo, contiene todas las características, funciones, lista de errores, defectos y mejoras tecnológicas. La lista es originada a partir de varias fuentes, tales como, el cliente, el equipo del proyecto y el área de mercadotecnia.

Esta práctica incluye la tarea de crear la lista, y controlarla constantemente durante todo el proceso, agregando, quitando, especificando, actualizando, y priorizando sus elementos.

Estimación de esfuerzo. Es un proceso iterativo en el cual las estimaciones iniciales de los elementos del producto Backlog son enfocadas en un nivel más exacto, gracias a la disposición de más información sobre un elemento determinado. El Product Owner junto con el Scrum Team son los responsables de realizar la valoración del esfuerzo.

Sprint. El Sprint es el procedimiento de adaptarse a los cambios de las variables de ambiente (requisitos, tiempo, recursos, conocimiento, tecnología etc.). El equipo de Scrum se organiza para producir un nuevo incremento del producto ejecutable en un Sprint que dura aproximadamente treinta días. Las herramientas de trabajo del equipo son las reuniones de planeacion del Sprint, el Sprint Backlog y las reuniones diarias de Scrum.

Reuniones de planeacion del Sprint. Es una reunión de dos fases organizada por el Scrum Master. Los clientes, usuarios, los gerentes, el Product Owner y el Scrum Team participan el la primera fase para decidir sobre las metas y la funcionalidad del siguiente Sprint. La segunda fase de la reunión es llevada a cabo por el Scrum Master y el Scrum Team enfocándose sobre como el

incremento del producto es implementado durante el Sprint.

Sprint Backlog. Es el punto de inicio para cada Sprint. Esto es una lista de los elementos del producto Backlog a ser implementados en el siguiente Sprint. Los elementos son seleccionados por el Scrum Team junto con el Scrum Master y el Product Owner en la reunión de planeacion del Sprint.

Reuniones diarias de Scrum. Diariamente son organizadas reuniones a fin de conocer el progreso del equipo de trabajo y estas también sirven para planear las reuniones siguientes, entonces las preguntas obligadas son: ¿Qué has hecho desde la última reunión? y ¿Qué harás antes de la próxima reunión?; también son discutidos los problemas encontrados y alguna otra variable importante en pequeñas reuniones de aproximadamente quince minutos. El Scrum Master es el encargado de dirigir estas reuniones.

Reuniones de revisión del Sprint. Al final del día del Sprint, el Scrum Team y el Scrum Master presentan el resultado del Sprint al gerente, clientes, usuarios y al Product Owner en una reunión informal. Los participantes tienen acceso al incremento del producto y toman las decisiones sobre las siguientes actividades.

2.6.4 Feature Driven Development (Desarrollo guiado por características)

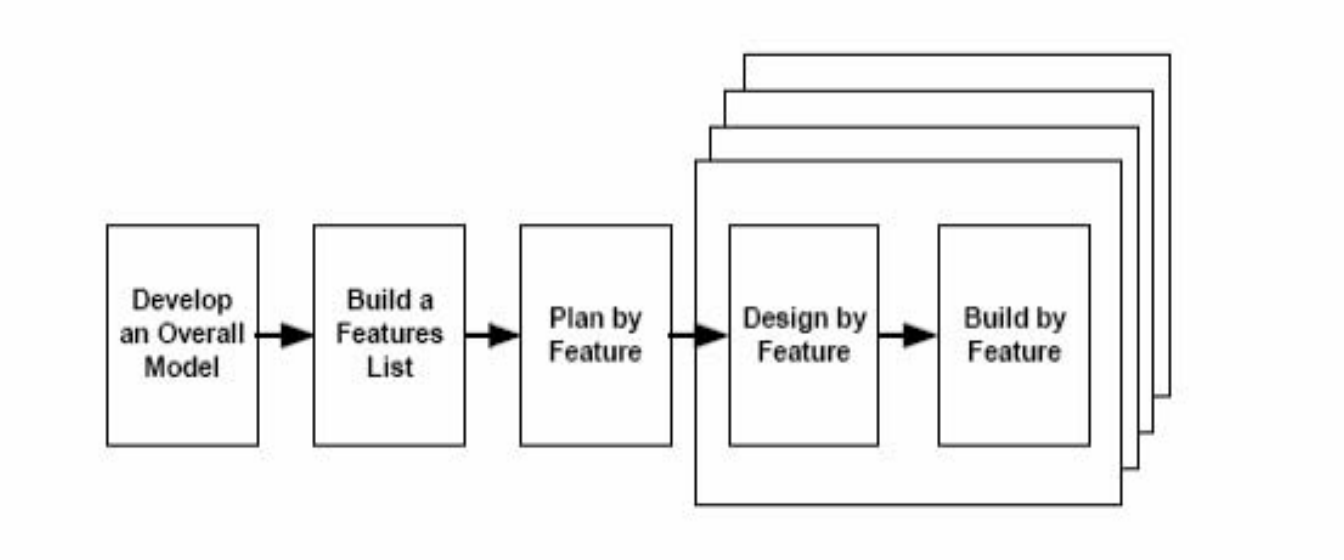
Es un ágil y adaptativo acercamiento para el desarrollo de sistemas. A diferencia de otras metodologías, no cubre todo el ciclo de vida del desarrollo del software, solamente se enfoca a las fases de diseño y construcción. Es considerada más adecuada para proyectos mayores y de misión crítica. Feature Driven Development (FDD) no requiere un modelo específico de proceso y se complementa con otras metodologías. Dentro de las principales prácticas que incorpora se encuentra el desarrollo iterativo, las entregas frecuentes y una supervisión constante del proceso del proyecto.

2.6.4.1 Proceso

Feature Driven Development (FDD) consiste en cinco procesos secuenciales durante los cuales el diseño y la construcción del sistema se realizan. El uso de un proceso iterativo proporciona mayor

flexibilidad para realizar adaptaciones a los cambios solicitados sobre los requisitos y las necesidades de negocio. Una iteración en Feature Driven Development (FDD) implica un periodo de una a tres semanas aproximadamente. En la figura 2.6.2 se muestra el proceso completo de Feature Driven Development (FDD) con su secuencia de cinco pasos.

Proceso de Feature Driven Development (FDD[36]) (Figura 2.6.2)



- *Desarrollo del Modelo general.* Al comienzo de esta fase ya debe conocerse el alcance, el contexto y los requisitos del sistema a ser construidos. En esta etapa la forma en que será constituido el sistema es modelada. Feature Driven Development (FDD) espera que los detalles del negocio sean descritos en los ciclos de desarrollo posteriores. El propósito de este modelo es proveer una guía en la planeación de las *features* y la integración del código desarrollado. "Los expertos de dominio presentan un ensayo en el que los miembros del equipo y el arquitecto principal se informan de la descripción de alto nivel del sistema. El dominio general se subdivide en áreas más específicas y se define un ensayo más detallado para cada uno de los miembros del dominio. Luego de cada ensayo, un equipo de desarrollo trabaja en pequeños grupos para producir modelos de objeto de cada área de dominio. Simultáneamente, se construye un gran modelo general para todo el sistema[37]."

[36] Abrahamsson, Pekka, Salo, Outi, Ronkainen, Jussi & Warsta, Juhani. Agile software development methods. Review and analysis. Espoo 2002. VTT Publications 478. p.48
[37] Idem. P.48.

- *Construir Lista de Features (características)*: El modelo de objetos y la documentación existente de los requerimientos proporcionan la base para la construcción de la Lista de *Features*. En la lista se presentan las funciones solicitadas por el cliente y que deben ser incluidas en el sistema. Se escriben en un lenguaje que todas las partes puedan entender El alcance de las *features* es intencionalmente diseñado para que se requieran dos semanas para resolverlo. Si las *features* parecen ser mas grandes que esto, son descompuestas en partes mas pequeñas. La lista de features es revisada por los usuarios y los patrocinadores del sistema para validarla y complementarla.

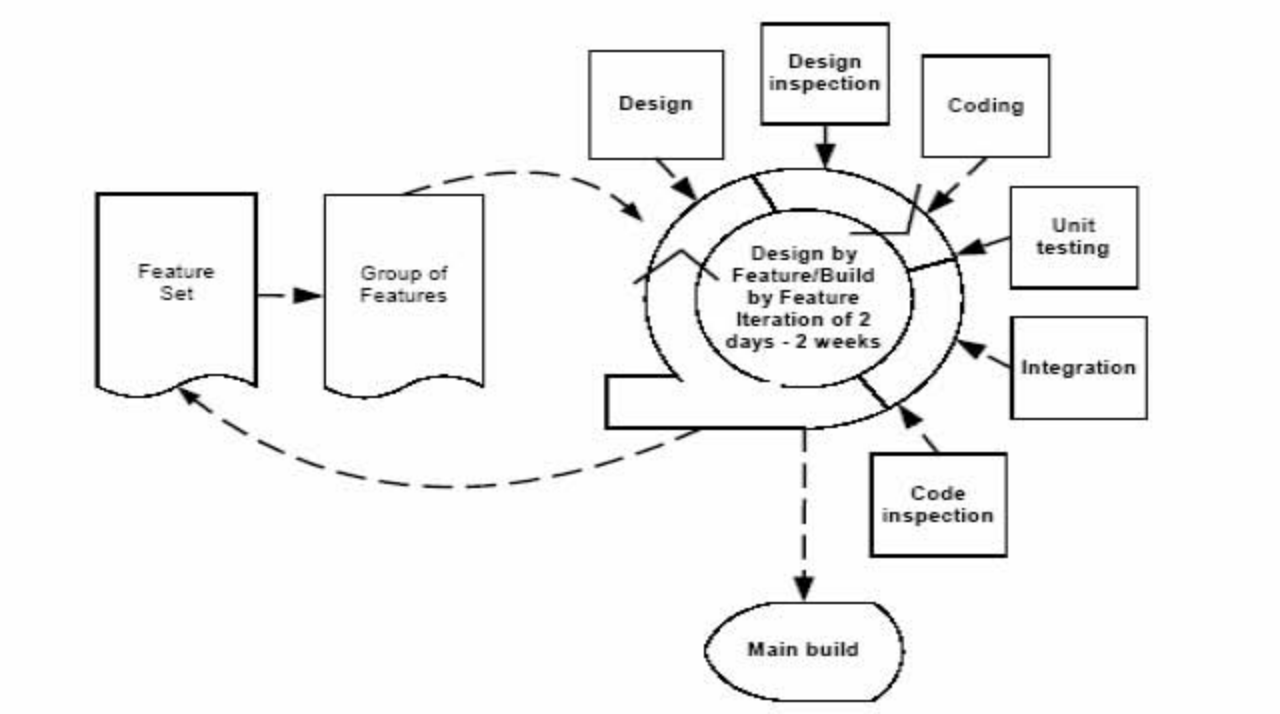
- *Planear por Features (características)*. El planear por feature incluye la elaboración de un plan de alto nivel, en el cual los features se ordenan según su prioridad y dependencia, y se asignan a los programadores jefes. El conjunto de *features* es usado por el equipo de planeación para crear el orden y los ciclos de liberación de las *features*. Los programadores principales manejan equipos pequeños. La responsabilidad de crear las clases y métodos necesarios para el funcionamiento es dada a los desarrolladores.

- *Diseño por Features*. En esta fase el equipo de trabajo se encarga de seleccionar un pequeño grupo de features de la lista original para ser diseñados. Se generan paquetes de diseño, los cuales consisten en componentes de modelamiento que describirán los objetos al punto de guiar las actividades de desarrollo a la forma de código. Secuencias y características a nivel de negocio son desarrolladas cooperativamente a nivel de equipo y las clases y los métodos son hechos por el responsable de éstos.

- *Construcción por Features*. Esta fase del proceso iterativo incluye las tareas de inspección del diseño, la codificación, la realización de pruebas unitarias, la integración y la inspección del código. Después de una iteración exitosa los features terminados se integran a la estructura principal y la fase de diseñar y construir comienza nuevamente con un subconjunto de features nuevos.

A continuación se presenta la figura 2.6.3, en la cual se puede observar el proceso iterativo de diseño y construcción de features, así como los pasos que se involucran en este ciclo.

El proceso de diseño y contracción por feature de Feature Driven Development (FDD[38]) (Figura 2.6.3)



2.6.4.2 Roles

Los roles en Feature Driven Development (FDD) se clasifican en tres categorías: roles claves, roles de soporte y roles adicionales. Estas categorías se describen a continuación:

Roles claves

- *Administrador del proyecto*: Es el líder administrativo y financiero del proyecto. Tiene la última palabra en el alcance o visión, los tiempos y la asignación del personal. Debe proteger al equipo de trabajo de distracciones exteriores y promover las condiciones adecuadas de trabajo.
- *Arquitecto jefe*: Es el responsable del diseño del sistema y la capacitación del equipo de trabajo para la realización del diseño. Toma las decisiones finales para todas las partes del diseño. En caso necesario, este rol puede dividirse en arquitecto de dominio y arquitecto técnico.
- *Administrador de desarrollo.* Tiene como responsabilidad conducir las actividades diarias de

[38] Abrahamsson, Pekka, Salo, Outi, Ronkainen, Jussi & Warsta, Juhani. Agile software development methods. Review and analysis. Espoo 2002. VTT Publications 478. P.50

desarrollo y solucionar cualquier conflicto que pueda ocurrir dentro del equipo. Las tareas de este rol se pueden combinar con el rol de arquitecto jefe o administrador del proyecto.

- *Programador jefe*: Es un desarrollador experimentado que participa en el análisis y diseño de proyectos. Es responsable de conducir a equipos pequeños en análisis, diseño y desarrollo de los features. Es el encargado de seleccionar los *features* a desarrollar en la siguiente iteración.
- *Propietarios de clases*: Trabajan bajo el mando del programador jefe en las tareas de diseño, codificación, prueba y documentación. Es el encargado de codificar las clases asignadas a su propiedad.
- *Experto de dominio*: Puede ser un usuario, un cliente, un patrocinador, un analista de negocios o una mezcla de todo eso. Deben poseer el conocimiento de los diferentes requerimientos del sistema y transmitírselo a los desarrolladores para que se genere un sistema eficiente.

Roles de soporte

- *Administrador de entrega*: Controla el progreso del proceso revisando los reportes del programador jefe y manteniendo reuniones breves con él; reporta al administrador del proyecto.
- *Experto del lenguaje*: Es un miembro del equipo, responsable de poseer un conocimiento especifico de una tecnología o lenguaje.
- *Ingeniero de construcción*: Es la persona responsable de la creación, mantenimiento y ejecución de los procesos de construcción de la aplicación (builds), incluyendo las tareas de administración de versiones y publicación de la documentación.
- *Herramientista*: Tiene la responsabilidad de construir pequeñas herramientas de ayuda al desarrollo, pruebas y conversión de datos en el proyecto. Por ejemplo: la creación y mantenimiento de bases de datos y sitios Web para específicos proyectos.
- *Administrador del sistema*: Su tarea es la configuración, administración y resolución de problemas de los servidores, redes o estaciones de trabajo. Desarrolla y prueba ambientes de trabajo para el equipo. También puede ayudar en la instalación del sistema en el ambiente de producción.

Roles adicionales

- *Probadores.* Se encarga de verificar que el sistema construido cumpla con los requerimientos que el cliente específico.

- *Deployer.* Su tarea es la migración de los datos existentes al formato requerido por el nuevo sistema, además de participar en la puesta en marcha de nuevas versiones.
- *Escritores Técnicos.* Tiene la responsabilidad de generar la documentación para el usuario.

Un miembro de un equipo puede tener otros roles a cargo, y un solo rol puede ser compartido por varias personas.

2.6.4.3 Características (Features)

Un *features* en Feature Driven Development (FDD) es una función pequeña expresada en la forma *<acción> <resultado> <por | para | de | a> <objeto>* con los operadores adecuados entre los términos.

La jerarquía de las *features* utiliza los siguientes formatos:

- *Para feature*: <acción> el <resultado> <de | para | sobre | por> un <objeto>
- *Para un conjunto de features* : <acción><…endo> un <objeto>
- *Para un conjunto mayor de features*: administración de <acción>

- Ejemplos:
- Calcular el total de la facturación de Noviembre (*feature*)
- Modificar el estado de las facturas de producción (*feature*)
- Administrar el perfil de los proveedores (*feature*)
- Haciendo una venta a un cliente (Conjunto de *features*)
- Cargando la facturación de los proveedores (Conjunto de *features*)
- Administración de Bancos (Conjunto mayor de *features*)

2.6.4.4 Practicas

Feature Driven Development (FDD) se compone de un conjunto de prácticas, que si bien, no son nuevas, la mezcla de ellas hace que los cinco procesos de Feature Driven Development (FDD) sean únicos para cada caso. Además sugiere que todas las prácticas disponibles deberían ser

usadas para obtener el mayor beneficio del método. A continuación se listan las prácticas propuestas por Feature Driven Development (FDD):

- *Modelado del Objeto de Dominio.* Exploración y explicación del dominio del problema. Esto da como resultado un framework donde los features son agregados.
- *Desarrollo por Features.* Desarrollo y seguimiento del progreso a través de una lista de pequeña funcionalidad descompuesta y cliente-valores de funciones.
- *Propiedad Individual de clase (Código).* Cada clase tiene una sola persona como único responsable de su consistencia, funcionamiento y de la integridad conceptual de la clase.
- *Equipos de Feature.* Se refiere a pequeños equipos, dinámicamente formados.
- *Inspección.* Se refiere al uso de los mecanismos mejor-conocido defecto-detección.
- *Builds regulares.* Se refiere a asegurarse que siempre hay un sistema disponible demostrable ejecutándose. Los builds regulares forman la línea base para la cual se agregan los nuevos features.
- *Administrador de la configuración.* Permite la identificación y seguimiento histórico de las últimas versiones de cada archivo de código de fuente terminado.
- *Reporte del progreso.* El progreso es reportado con base en el total del trabajo de todos los niveles necesarios de la organización.

2.7 Evaluación de metodologías

La eficiencia de un proceso de software se mide de una forma indirecta. Se evalúan un conjunto de métricas según los resultados que provienen del proceso. Entre estos resultados se incluyen medidas de errores detectados antes de la entrega del software, defectos detectados e informados por el usuario final, el esfuerzo humano, el tiempo consumido, ajustes a la planeación, entre otros.

Las métricas de software se refieren a un conjunto de medidas para el software de computadora, la medición se puede aplicar al proceso de software con el intento de mejorarlo sobre una base continua.

La finalidad de la utilización de métricas para la evaluación de la eficiencia del proceso de software es la creación de un producto de calidad, lo que ocasiona que también aumente:

- La eficiencia en costos y tiempo.
- La posibilidad de repetir éxitos en proyectos.
- El control de los riesgos en los procesos.
- Confianza y satisfacción del cliente.

Las evaluaciones de proceso de software se han establecido como un instrumento objetivo para la determinación de la capacidad de los procesos de una organización. Durante una evaluación se examinan, con base en pruebas objetivas, las exigencias de los procesos y la capacidad de llevar estas exigencias a la práctica. Por medio de entrevistas individuales o de grupo, talleres y el estudio de la documentación existente se determina el estado actual de la organización.

Existen algunas normas internacionales que permiten la evaluación de estos procesos, entre las cuales se encuentra CMM (Capability Maturity Model).

La meta principal de una evaluación es el hecho de comprobar en qué modo los diferentes elementos que componen el proceso, se estimen convenientes para su mejora y si estas estimaciones pueden entregar como resultado un producto con la calidad esperada sin exceder el tiempo y los costes definidos.

La evaluación del proceso de software es importante para el presente trabajo de investigación, debido a que con base en esta podemos desarrollar los puntos mencionados a continuación:

- Verificar los procesos actuales, carencias y aciertos.
- Identificar los elementos principales del proceso y su potencial de mejoramiento.
- Planear acciones de mejoramiento concretas con base en las diferentes circunstancias.

A continuación se muestra una tabla comparativa de las tres metodologías de mayor interés para este trabajo de investigación. Esta tabla, exclusivamente permite ver las diferencias y similitudes entre las metodologías en referencia a su modelo de proceso, las fases que involucran, el modelo de equipo de trabajo (roles) y las disciplinas que contemplan para el control de un proyecto.

Comparación de metodologías RUP, MSF y XP[39]. (Tabla 2.7)

	MSF	RUP	XP
Modelo de proceso	Iterativo	Iterativo	Iterativo
Fases	▪ Prevision ▪ Planeacion ▪ Desarrollo ▪ Estabilización ▪ Liberación (Deploy)	▪ Concepción ▪ Elaboración ▪ Construcción ▪ Transición	▪ Exploración ▪ Planificación de la Entrega ▪ Iteraciones ▪ Producción ▪ Mantenimiento ▪ Muerte del Proyecto
Modelo de equipo	▪ Gerente de producto ▪ Desarrollador ▪ Probador ▪ Experiencia del usuario ▪ Gerente de Versiones ▪ Miembros adicionales del equipo	▪ Analista ▪ Desarrollador ▪ Probador ▪ Gerente ▪ Conjunto adicional de roles	▪ Programador ▪ Cliente ▪ Encargado de pruebas ▪ Encargado de seguimiento ▪ Entrenador ▪ Consultor ▪ Gestor
Disciplinas	▪ Gerencia de proyecto ▪ Gerencia de riesgos ▪ Gerencia de configuración	▪ Modelado de Negocio ▪ Requerimientos ▪ Análisis y Diseño ▪ Implementación ▪ Pruebas ▪ Despliegue (Deploy) ▪ Ambiente ▪ Gerencia de proyecto ▪ Gerencia de configuración y cambios	▪ Requerimientos ▪ Planeacion ▪ Análisis y Diseño ▪ Implementación ▪ Pruebas ▪ Refactorización ▪ Integración continua

Como se puede observar en la tabla anterior, la similitud en los procesos abarcados por las tres metodologías es muy grande, sin embargo durante la descripción de las mismas en las paginas anteriores, se nota la importancia que se le da a cada proceso, identificado por el nivel de detalle y las diferentes alternativas de solución que las metodologías presentan para cada uno de ellos.

Dentro de este trabajo de investigación realizaremos la evaluación de las metodologías descritas con anterioridad bajo las siguientes variables:

- Definición de requerimientos.
- Iteración con el cliente.
- Tiempos de codificación.
- Número de errores detectados en el producto.
- Tiempo consumido y esfuerzo planeado.

[39] Elaboración propia.

- Planificación del proyecto.
- Capacitación de Recursos Humanos.
- Administración de cambios.

Debido al limitado tiempo que se tiene para la realización de este trabajo de investigación, la evaluación de las metodologías no se llevara a cabo en un caso práctico. La forma a realizarse, es mediante la calificación de las prácticas (de estas metodologías), que contemplen el control y solución a las variables antes descritas. Es decir, cada metodología tendrá una calificación con base al nivel en que contemple los mecanismos suficientes para el control de la variable. Estas calificaciones se asignaran con base a lo analizado en los apartados anteriores.

A continuación se muestra la tabla con las calificaciones de cada metodología, sobre el nivel de mecanismos contemplados, para el control de las variables antes descritas.

Calificación de variables por metodología[40]. (Tabla 2.8)

	METODOLOGÍA		
VARIABLES	RUP	XP	MSF
Definición de Requerimientos	5	5	3
Iteración con el cliente	4	3	0
Tiempos de codificación	0	5	0
Numero de errores	4	5	1
Planificación del proyecto	4	4	3
Capacitación de recursos humanos	0	3	0
Administración de cambios	5	0	0
Tiempo consumido y esfuerzo planeado	4	0	3

La escala de calificación para las variables elegidas es la siguiente:

0 – La metodología no contempla estrategias para controlar la variable.

5 – La metodología plantea las estrategias necesarias para controlar la variable.

[40] Elaboración propia.

Hay que tomar en cuenta que existen cuatro metodologías complementarias que fueron descritas en el presente capítulo, y de las cuales no se realizo una calificación sobre las variables propuestas. Esto no quiere decir que no se tomaran en cuenta para la generación de la propuesta.

Las calificaciones asignadas en la tabla anterior, se apoyaron en los argumentos que describen las metodologías al inicio de este capitulo, los cuales se explican a continuación:

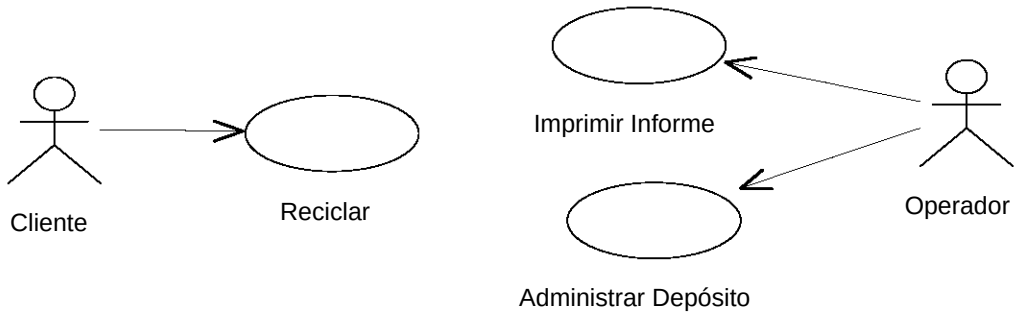
Definición de Requerimientos.

Para la variable “definición de requerimiento”, se puede ver que RUP utiliza un acercamiento sistemático para extraer y documentar los requerimientos del sistema, y entonces poder administrar cambios sobre estos requerimientos, incluyendo evaluaciones del impacto de los cambios, sobre el resto del sistema. La efectiva administración de requerimientos involucra mantener una clara declaración de estos, así como mantenerlos disponibles para otros artefactos del proyecto.

Los desarrolladores y el cliente deben acordar que es lo que el sistema debe hacer:

- Levantar requerimientos.
- Documentar funcionalidad y restricciones.
- Documentar decisiones.
- Identificar actores.
- Identificar casos de uso.
- Los casos de uso describen la funcionalidad.
- Los requerimientos no funcionales se incluyen en una especificación complementaria.

Representación de casos de uso (Figura 2.7.1)



Con base en las prácticas descritas, podemos observar que RUP, propone la ejecución de una serie de controles con el fin de asegurarse de una correcta definición de requerimientos.

Por otro lado, XP considera que es necesario tener una estrecha relación de comunicación entre el cliente y el equipo de desarrollo, con el fin de definir de una forma correcta lo que debe hacer el sistema. Esto lo apoya en las *historias de usuario*, donde definen junto con el usuario los escenarios que contemplara el sistema.

Una historia de usuario es un texto de una o dos frases en las que se explica brevemente algo que debe hacer el sistema. Es más extensa que un requisito (que suele ser una frase corta) y menos que un caso de uso[41] (que puede ser de una o dos cuartillas).

Las historias de usuario son lo suficientemente comprensibles y delimitadas para que los programadores puedan implementarlas en unas semanas

Por su parte en MSF, la fase del planeación implica el análisis de requerimientos. Estos requerimientos se pueden categorizar como de negocio, exigencias del consumidor, operacionales, y del sistema. Estos requerimientos se utilizan para diseñar la solución y sus características, así como para validar la corrección del diseño.

Del análisis de las tres metodologías en el manejo de esta variable, podemos concluir que tanto RUP, como XP contemplan una serie de medidas de control para disminuir el riesgo en la definición de los requerimientos del sistema, por lo tanto el nivel de calificación sobre estas es

[41] Caso de uso: consiste en una definición detallada de un requerimiento. Concepto perteneciente a UML.

bastante bueno. Se puede decir, que por si solas estas metodologías controlan este aspecto, pero una mezcla de ellas podría asegurarnos mucho más control, el cual es parte fundamental del proceso de producción de un software.

Iteración con el cliente.

Para la variable “Iteración con el cliente”, RUP reconoce la importancia de la participación del cliente durante el inicio de un proyecto de desarrollo de software para lograr su terminación exitosa, pero usualmente cometemos el error de olvidar esta norma básica, lo que implica que la participación del cliente se restringe al inicio y finalización del proyecto, lo que en la mayoría de los casos produce un alto grado de insatisfacción en el usuario, al no obtener el producto con las especificaciones esperadas.

Dos de las ventajas importantes de involucrar al cliente durante el proceso de la producción de software es el contar con su participación en la tarea de planificación de las fases e iteraciones, y tener una fuente para la retroalimentación en cada iteración. Estas dos consideraciones permiten que se pueda identificar los riesgos más rápidamente y resolverlos con base al conocimiento del experto en el negocio, “el cliente”.

Las consideraciones mencionadas anteriormente dan una idea de la importancia que RUP le otorga a la participación del cliente durante todo el proceso de producción, con lo cual la calificación otorga en la tabla 2.7 esta justificada.

Por otro lado, con XP, el cliente tiene que estar presente y disponible todo el tiempo para el equipo. Gran parte del éxito del proyecto XP se debe a que es el cliente quien conduce constantemente el trabajo hacia lo que aportará mayor valor al negocio y los programadores pueden resolver de manera inmediata cualquier duda asociada. La comunicación oral es más efectiva que la escrita, ya que esta última toma mucho tiempo en generarse y puede tener más riesgo de ser mal interpretada. Según Kent Benk[42], un verdadero cliente debe sentarse con el equipo, disponible para contestar preguntas, resolver dudas, y asignar las prioridades para el sistema.

Es por esto que las practicas XP al respecto, también son recomendadas como una buena opción

[42] Kent Benk. Extreme Programming Explained: Embrace Change. Ed. Addison Wesley. 1999. E.U.A.

para tener bajo mayor control esta variable, y por consiguiente mitigar riesgos durante el proceso de producción de software.

Por su parte MSF, no presenta mucha información sobre este punto, lo cual indica que existe un posible hueco (riesgo), en el caso de aplicar por completo esta metodología en un proceso de producción de software.

Nuevamente se puede afirmar que una buena conjunción de las prácticas de RUP y XP, para el manejo de esta variable, nos aseguran un mejor control sobre el proceso y por lo mismo, mayores posibilidades de tener éxito.

Tiempos de codificación.

Para la variable "Tiempos de codificación", RUP y MSF no realizan alguna mención sobre las practicas a ejecutar para controlar y agilizar este proceso. En el proceso de producción de software es importante destacar la importancia de los recursos humanos como parte primordial del éxito de un proyecto. Podemos afirmar que los tiempos de codificación son dependientes directos de los recursos humanos que lo ejecutan.

Por su parte XP, contempla algunas prácticas que impactan directamente en los tiempos de codificación.

- ✓ Incentiva las relaciones interpersonales como clave para su éxito.
- ✓ Promueve el trabajo en equipo.
- ✓ Procura el aprendizaje de los desarrolladores.
- ✓ Propicia un buen clima de trabajo.
- ✓ Se basa en retroalimentación continua entre el cliente y el equipo de desarrollo.
- ✓ Comunicación fluida entre todos los participantes.
- ✓ Simplicidad en las soluciones implementadas.
- ✓ Programación por pareja.

Estas prácticas ayudan a mantener un buen ambiente laboral y al mismo tiempo propician el crecimiento profesional de los involucrados en el proyecto. La implantación de estas prácticas

durante el proceso de producción de software puede ser parte importante para tener éxito.

Número de errores.

Para la variable “numero de errores”, RUP continuamente evalúa la calidad del sistema con respecto a los requerimientos funcionales y no funcionales. Se ejecutan pruebas como parte de cada iteración. Esto permite incrementar la detección y corrección de errores a temprana hora durante el ciclo de vida de desarrollo de software, en vez de corregir errores encontrados más tarde. Esta practica de RUP, es muy importante, ya que permite tener una idea mas clara del avance real del proyecto y realizar las correcciones durante el periodo que dura una iteración.

Por su parte en XP, la producción de código está dirigida por las pruebas unitarias. Las pruebas unitarias son establecidas antes de escribir el código y son ejecutadas constantemente ante cada modificación del sistema. Los clientes escriben las pruebas funcionales para cada historia de usuario que deba validarse. En este contexto de desarrollo evolutivo y de énfasis en pruebas constantes, la automatización para apoyar esta actividad es crucial.

MSF, no contempla práctica alguna para el control de esta variable, por lo tanto la calificación sobre esta mostrada en el cuadro 2.7, no debe ser muy buena. No proporciona ventajas y si desventajas con respecto a XP y RUP.

Como se puede observar, tanto RUP como XP, contemplan mecanismos similares para la detección y corrección temprana de errores. Quizás la complementación de estas prácticas proporcione un mejor control sobre la calidad de los productos generados, por lo cual es necesario contemplarlo.

Planificación del proyecto.

Para la variable “Planificación del proyecto”, RUP define que la fase de concepción, tiene como propósito definir y acordar el alcance del proyecto, identificar los riesgos asociados al proyecto, proponer una visión general de la arquitectura de software y definir el plan de trabajo para las fases e iteraciones.

En otras palabras, la fase de concepción del proyecto es para RUP, la fase de planeación en la cual se definirá el alcance del mismo y la forma en que se llevara a cabo su ejecución para las siguientes iteraciones.

Por su parte XP, pretende que el desarrollo de un proyecto de software sea ágil, disciplinado y con soluciones sencillas. Además, esta metodología cuenta con un enfoque de adaptación, en la que la planificación del proyecto progresa a medida que surgen cambios. En la práctica, el juego de planeación es un espacio frecuente de comunicación entre el cliente y los programadores. El equipo técnico realiza una estimación del esfuerzo requerido para la implementación de las historias de usuario y los clientes deciden sobre el ámbito y tiempo de las entregas y de cada iteración.

MSF, propone que durante la fase del planeación, el equipo determine qué va a desarrollar y planee cómo crear la solución. El equipo prepara la especificación funcional, crea un diseño de la solución, y prepara planes de trabajo, valoraciones de costos, y horarios para entregables**.**

Como se puede observar las tres metodologías contemplan varias prácticas encaminadas a dar mayores ventajas sobre el proceso de planeación, estas proporcionan mayor información sobre los riesgos, lo que permite establecer planes de trabajo más acordes a la realidad y en una secuencia mas precisa.

Administración de cambios.

Para la variable “administración de cambios”, RUP es la única de las tres que establece un mecanismo disciplinado y controlado para esta tarea. Define las reglas para administrar que cambios son agregados en los artefactos del proyecto, quienes solicitan e introducen estos y la planificación de su ejecución. Esta administración es importante ya que da la pauta para conocer los impactos reales que se realizan en el producto y los tiempos exactos cuando se deben aplicar.

Esta es una de las variables que mas hay que vigilar, debido a que la aplicación de un cambio en un tiempo no propicio o sin conocer las implicaciones que tiene con respecto al conjunto de componentes del producto, puede dar como resultado el fracaso del proyecto.

Conclusiones

Como se puede observar, los factores que determinar el éxito o fracaso del proceso de producción de software son muy variados. Las metodologías aquí evaluadas no contemplan todos estos puntos, y en algunas de ellas no los toman en cuenta durante todo su proceso.

Es evidente que cada metodología esta concebida con características particulares a ser aplicadas en escenarios bien establecidos y que pueden variar según la complejidad del producto de software a desarrollar, el número de requerimientos y la importancia que este tenga con respecto al negocio que pertenece.

Las prácticas rescatables de las tres metodologías sirven como herramientas para llevar a cabo un mejor control sobre el proceso de producción de un producto de software.

Las metodologías analizadas anteriormente cuentan con prácticas muy semejantes y que en algunos casos se complementan entre si. Esto ayudara al diseño de la guía metodológica a fin de lograr el mejor resultado posible del proceso de producción de software.

Las características más relevantes de las metodologías analizadas con base en las variables en estudio (Definición de requerimientos, Iteración con el cliente, Tiempos de codificación, Numero de errores, Planificación del proyecto, Capacitación de recursos humanos, Administración de cambios, Tiempo consumido y esfuerzo planeado) y descritas en el apartado 2.7 del capitulo dos, son:

Rational Unified Process (RUP)	Del apartado 2.3 : • Desarrollo iterativo de software (Ciclo de Vida). • Administración de Requerimientos. • Uso de arquitecturas basadas en componentes. • Modelado visual de software haciendo uso efectivo de UML (Unified Modeling Language). • Verificación de la calidad del software. • Control de cambios.

	• Orientación al manejo de riesgos. • Orientación al cliente. • Desarrollo evolutivo.
Extreme Programming (XP).	Del apartado 2.4: • Desarrollo iterativo. • El juego de la planificación (the planning game). • Pequeñas entregas (small releases). • Metáfora (metaphor). • Diseño simple (simple design). • Pruebas (testing). • Refactorización (refactoring). • Programación por parejas (pair programming). • Propiedad colectiva (collective ownership). • Integración continua (continous integration). • 40 horas semanales (40-hour week). • Cliente en casa (on-site costumer). • Estándares de codificación (coding standards). Valores : • Comunicación. • Feedback rápido y continuo • Simplicidad • Coraje
Microsoft Solution Framework (MSF).	Del apartado 2.5 • Desarrollo iterativo. • Análisis de requerimientos. • Administración de cambios.

Scrum	Del apartado 2.6 • Desarrollo iterativo. • producto BackLog (Lista de retraso o acumulación). • Estimación de esfuerzos. • Sprint. • Reuniones de planeación de Sprint. • Sprint BackLog. • Reuniones diarias de SCrum. • Reuniones de revisión de Sprint. Valores: • Equipos auto-dirigidos y auto-organizados. • Una vez elegida una tarea, no se agrega trabajo extra. En caso que se agregue algo, se recomienda quitar alguna otra cosa. • Reuniones diarias de trabajo • Iteraciones de treinta días; se admite que sean más frecuentes. • Demostración a participantes externos al final de cada iteración. • Al inicio de cada iteración, planeamiento adaptativo guiado por el cliente.
Crystal Family	Del apartado 2.6 • Exploración de 360. • Victoria temprana. • Esqueleto ambulante. • Re-Arquitectura incremental.

	• Radiadores de información. • Entrevistas de proyectos. • Talleres de reflexión. • Planeamiento Blitz. • Estimaciones de pericia. • Encuentros diarios de pie. • Miniatura de procesos. • Gráficos de quemado. • Programación lado a lado. Valores: • Entrega frecuente. • Comunicación. • Mejora reflexiva. • Seguridad personal. • Foco (enfoque sobre el trabajo). • Fácil acceso a usuarios expertos. • Ambiente técnico con prueba automatizada, administrador de configuración e integración frecuente.
Open Source	Del apartado 2.6 • Trabajo en equipos físicamente distribuidos. • Mantenedores de código. (Integrador) • Control de versiones (uso de herramientas como cvs-version control system, para la administración de código)
Feature Driven Development	Del apartado 2.6 Modelado del Objeto de Dominio.

	Desarrollo basado en Features. Propiedad Individual de clase (Código). Equipos de Feature. Inspección. Builds regulares. Administrador de la configuración. Reporte del progreso.

CAPÍTULO 3

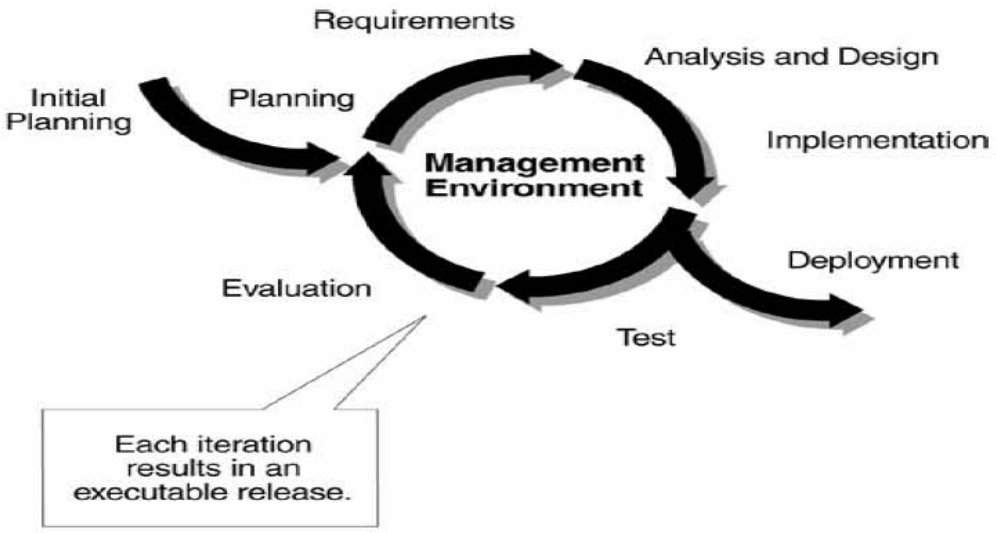
PLANEACIÓN DE LA GUÍA METODOLÓGICA PARA LA PRODUCCIÓN DE SOFTWARE

Como se menciona en el capitulo uno, del presente trabajo de investigación, regularmente las metodologías para la producción de software cubren los siguientes puntos:

- ✓ Planificación: ámbito del proyecto, estudio de viabilidad, análisis de riesgos, planificación temporal, asignación de recursos.
- ✓ Análisis (¿qué?): levantamiento de requerimientos.
- ✓ Diseño (¿cómo?): estudio de alternativas, diseño arquitectónico.
- ✓ Implementación/Construcción/Desarrollo: adquisición, creación e integración de los recursos necesarios para que el sistema funcione.
- ✓ Pruebas: pruebas de unidad, pruebas de integración, pruebas alfa, pruebas beta, pruebas de aceptación.
- ✓ Mantenimiento (correctivo y adaptativo).

Estos puntos se desarrollaran en el presente capítulo a fin de proporcionar la guía metodológica que permita ayudar en la mejora del proceso de producción de software. Lo que se pretende con ello es proporcionar a las organizaciones dedicadas a la producción de software, la información necesaria sobre las mejores practicas combinadas de las metodologías en estudio. Además sirve como referencia para las personas que estudian las carreras relacionadas con la ingeniería de software, permitiéndoles tener una guía de ayuda para el proceso de producción de software.

Proceso de desarrollo de software incremental e iterativo[43] (Figura 3.1)



[43] Philippe Kruchten.The.rational unified process an introduction 3rd edition. Ed. Addison Wesley. Diciembre 2003, E.U.A. p. 7.

En la figura 3-1 se pueden observar los pasos que componen el proceso de desarrollo de software incremental e iterativo. Este modelo de desarrollo lo utilizan varias de las metodologías estudiadas en el capitulo 2 y el cual se retoma como parte de la propuesta de mejora en el presente capitulo. La característica principal de este proceso consiste en la identificación temprana de riesgos en el proyecto, lo cual permite atacar y reaccionar a ellos de una manera oportuna y eficiente. Además, en cada iteración se obliga al equipo de desarrollo a cerrar una versión estable de los artefactos del producto.

"El desarrollo iterativo de software ofrece un número de soluciones a las causas de raíz de los problemas del desarrollo del software:

- Los malentendidos serios se hacen evidentes a una hora temprana en el ciclo de vida, cuando es posible reaccionar a ellos.
- La aproximación permite y anima al usuario a reanalizar para sacar los requerimientos verdaderos del sistema.
- El equipo de desarrollo es forzado a centrarse en estas partes que son las más críticas del proyecto y se blindan de las partes que los distraigan de los riesgos verdaderos.
- La continúa prueba iterativa permite una visión objetiva del estado del proyecto.
- Las inconsistencias entre los requerimientos, diseños, e implementación se detectan temprano.
- La carga de trabajo del equipo, especialmente el equipo de prueba, se separa más uniformemente a través del ciclo de vida del proyecto.
- El equipo puede asimilar las lecciones aprendidas y por lo tanto puede mejorar continuamente el proceso.
- En el proyecto pueden ser dados a los Stakeholders la evidencia concreta del estado del proyecto a través de su ciclo de vida.[44]"

De esta forma, el desarrollo en iteraciones, se caracteriza por la entrega de pequeños incrementos de funcionalidad en cortos lapsos de tiempo; lo que permite tener una respuesta más rápida y acertada a cambios en los requerimientos. Dependiendo de la magnitud del software a producir, las iteraciones pueden variar entre dos semanas y tres meses de ejecución.

[44] Philippe Kruchten.The.rational unified process an introduction 3rd edition. Ed. Addison Wesley. Diciembre 2003, E.U.A. p. 8.

Las características descritas anteriormente dan una clara idea de la importancia que juega esta práctica en el proceso de producción de un software. Esta practica es una constante en cinco de las siete metodologías analizadas en el capitulo anterior, entre ellas se encuentra RUP, MSF, XP, FDD y Scrum.

Con base en el análisis comparativo realizado en el capitulo anterior, esta guía metodológica estará compuesta en su mayoría de dos de las tres principales metodologías estudiadas, Rational Unified Process (RUP) y extreme programming (XP). Y aunque la división de las etapas del proceso de producción de software presentado en este trabajo de investigación varía con respecto a estas, se retomaran de tal forma que se distribuyan en sus apartados correspondientes. Además como parte de la propuesta del presente se describe a continuación las etapas que debieran estar presentes en el proceso de producción de un software.

3.1 Planeación

Antes de entrar en materia sobre la forma de planificación propuesta, con base en el análisis del capítulo anterior, se definirá el termino planeación. En su libro "Administración", Stephen P. Robbins define la planeación de la siguiente forma:

"La planeación establece el esfuerzo coordinado. Da dirección a los administradores como a los no administradores. Cuando todos saben a donde de dirige la organización y lo que espera que contribuyan para alcanzar los objetivos, debe existir mayor coordinación, cooperación y trabajo en equipo. La carencia de planeación puede fomentar el zigzagueo y por tanto evitar que la organización se dirija de forma eficiente hacia sus objetivos.[45]"

"La planeación es el proceso de determinar objetivos y definir la mejor manera de alcanzarlos. Se ocupa, pues, de los medios, (como se debe hacer) y de los fines (que es lo que tiene que hacer)[46]."

"La planeación es una manera de reducir la incertidumbre a través de la previsión del cambio.[47]"

"La planeación establece los objetivos o estándares que deben usarse para facilitar el control. Si

[45] Stephen P. Robbins. Administración. Ed.Prentice-Hall. p. 114.
[46] Idem. p. 114.
[47] Idem. p. 115.

no estamos seguros de lo que tratamos de alcanzar, ¿cómo podremos determinar si lo hemos alcanzado? En la planeación, desarrollamos lo objetivos. En la función de control comparamos el desempeño real con los objetivos, identificando cualquier desviación significativa y aplicando las acciones correctivas necesarias. Sin planeación, no puede haber control[48]."

En las definiciones citadas anteriormente se puede observar la importancia que tiene la planeación para cualquier organización. En términos del presente trabajo de investigación, se puede decir que cualquier organización dedicada a la producción de software debe tomar como tarea principal la planeación de sus diferentes procesos.

En otras palabras, y enfocado a la producción de software, se puede decir que el principal proceso de esta actividad es determinar los objetivos (el producto de software) y los medios con los cuales se va a llevar a cabo su producción. Es un esfuerzo coordinado de un equipo de trabajo, con la finalidad de alcanzar un fin común, evitar la incertidumbre y dar control sobre el proceso. A esto le podemos llamar planeación.

La propuesta para la fase de planeación, como el primer y más importante paso de un proceso de producción de software, se describe a continuación.

Con base en la metodología RUP, la cual se enfoca en los aspectos específicos de un proceso de desarrollo iterativo, se puede definir la fase de planeación de la siguiente forma:

Un plan general o también llamado de la fase, y una serie de planes de iteración. Regularmente, en los diferentes proyectos de producción de software se cae en el error de realizar una planeación de comienzo a fin en forma detallada, esto es riesgoso, debido a que para que tales planes sean realistas, se debe tener una comprensión muy buena de que será construido, con requerimientos fijos y una infraestructura estable, esto implica, que el planeador tenga la suficiente experiencia en proyectos similares para poder asumir muchas de las consideraciones existentes.

Como se menciono anteriormente, la fase de planeación debe dividirse en dos partes:

El **plan general**, el cual debe contemplar el ciclo de vida del proyecto, por lo tanto debe ser único.

[48] Stephen P. Robbins. Administración. Ed.Prentice-Hall. p. 115.

Debe contener el ciclo completo del proyecto y puede ser resumido por:

- Fechas de los principales puntos de control
    - Objetivo del ciclo de vida.
    - Arquitectura del ciclo de vida.
    - Capacidad operacional inicial.
    - Lanzamiento del producto.
- Proveer del personal con el perfil específico.
    - ¿Qué recursos se requieren, especialidad y en que momento?
- Fechas de los puntos de control de menor importancia.
    - Final de cada iteración y su objetivo primario, si se conoce.

Es necesario recordar que la elaboración del plan general se realiza como primer paso del proceso de producción de un producto de software y que esta basado en estimados y supuesto, lo cual provoca que se este actualizando durante el tiempo que dure el proyecto.

El plan general no es, ni debe ser estático, siempre debe haber la posibilidad de cambios durante el proceso de su ejecución.

El **plan de iteración**, debe contemplar el lapso de tiempo que dura la iteración, debe ser uno por iteración y varios en el proyecto. Un proyecto tiene generalmente dos planes de la iteración activos simultáneamente:

- El plan actual de la iteración que se utiliza para seguir el progreso.
- El plan siguiente de la iteración (el que esta para la iteración pendiente), que se construye hacia la segunda mitad de la iteración actual y esta listo al final de la iteración actual.

La construcción del plan de iteración debe realizarse usando las técnicas y herramientas tradicionales (los diagramas de Gantt[49], etcétera) de planeación para definir las tareas, asignaciones por individuo y por equipos. El plan debe contener las fechas más importantes, tales

[49] Este fue desarrollado por Henry L. Gantt en 1917 y es una sencilla herramienta de gráficos de tiempos, ya que son fáciles de aprender, leer y escribir. Estos resultan bastante eficaces para la planificación y la evaluación del avance de los proyectos

como: llegada de componentes de otras organizaciones, definiciones de arquitectura y revisiones importantes.

La elaboración del plan de iteración se debe realizar por los integrantes del equipo de desarrollo y de negocio (aquellas personas que definen los requerimientos y toman decisiones sobre el negocio) en forma conjunta. Esto implicara que los tiempos asignados a las diferentes tareas sean más apegados a la realidad, debido a que los diferentes equipos (análisis, diseño, desarrollo, pruebas, etc.) conocen con mayor certeza los tiempos requeridos para sus actividades, basados en su experiencia.

Adicionalmente a las prácticas de RUP, para esta fase de planeación, existe en XP una que ayuda a mejorar la definición de actividades en un proceso de desarrollo iterativo y que es parte importante de esta propuesta. Así pues, debe existir una constante iteración entre los clientes (conocedores del negocio) y los programadores. Estos últimos deben estimar el tiempo necesario para la codificación y los conocedores del negocio decidir sobre el alcance de los requerimientos. Esta práctica es conocida como el juego de la planeación, y su ejecución debe realizarse durante la elaboración de los planes de iteración en todo el ciclo.

La entrada inicial para la generación de estos planes, es la lista de requerimientos o características que debe cumplir el producto de software a ser construido. Este motivo implica que el plan inicial de un proyecto de producción de un software considere diversos supuestos y experimente un constante cambio durante las siguientes fases del proceso, ajustándose a un entorno más real una vez analizados los requerimientos.

En resumen, las prácticas propuestas para esta fase se listan a continuación con su respectivo origen:

- Desarrollo Iterativo – XP, RUP, Scrum, FDD, MSF.
- Juego de la planeación - XP.
- Realización de un Plan general y varios de iteración para el proceso - RUP, XP.

Las prácticas mencionadas forman parte de las metodologías de RUP y XP, para mayor detalle de lo aquí descrito se pueden consultar:

- Capítulo 7, The Project Management Discipline, del libro “The rational unified process an introduction 3rd edición”, de la editorial Addison Wesley.

- Capítulo 15. Planning Strategy, Del libro “Extreme Programming Explained: Embrace Change”, de Kent Beck, editorial Addison Wesley, 1999.

3.2 Análisis

El insumo principal de la fase de análisis son los requerimientos, estos indican la funcionalidad y características del producto de software a generar. El propósito del análisis es transformar los requerimientos del sistema en documentos detallados con la funcionalidad solicitada; lo que servirá como base para que el equipo de diseño modele un conjunto de subsistemas, componentes y clases que conformaran el producto final.

Las metodologías manejan diversas formas para la administración de los requerimientos, XP utiliza las historias del usuario, Scrum utiliza el Backlog o listas de retraso, RUP utiliza los casos de uso; Y aunque en esencia, la finalidad de los tres es el levantamiento de requerimientos de una forma ordenada y eficiente para su buena administración, la practica propuesta para este proceso es la de RUP, es decir, la utilización de los casos de uso para la documentación y administración de requerimientos.

Durante esta fase se debe realizar el análisis de los requerimientos funcionales (reglas de negocio) y no funcionales (componentes de arquitectura), con la finalidad de definir los componentes necesarios para el conjunto de la aplicación. A esta práctica se le conoce como desarrollo basada en componentes, y permite una gran flexibilidad para la realización de cambios, desarrollos rápido y mantenimiento.

Es muy importante la interacción continua y cercana de los analistas y el cliente (conocedores del negocio) durante el proceso de análisis. Para asegurar el hecho de que se de esta situación, se propone la integración al equipo de trabajo a los conocedores de negocio, estando disponibles en cualquier momento para resolver dudas. Esta integración le permite al cliente conocer mas sobre la forma en que se lleva a cabo el proceso, aprende de el y no depender de personas externas para la realización de cambios. Esta práctica se aplica en XP y se le conoce como cliente en sitio.

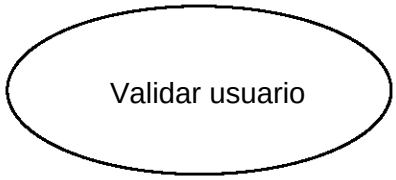
Por otro lado, y como parte de la presente guía se propone la utilización del lenguaje UML[50] para la definición de requerimientos, en la fase de análisis, y de componentes en la fase de diseño. La utilización de esta herramienta es parte de las buenas prácticas que RUP utiliza en su proceso.

UML es una herramienta que nos proporciona una notación y un meta-modelo contenido en una serie de diagramas. Y aunque existen diferentes diagramas para las fases de producción de un software, aquí solo se mencionan el mínimo necesario para poder llevar a cabo un análisis de forma consistente. Para la fase de análisis se debe utilizar el caso de uso.

La importancia de utilizar UML como lenguaje de modelado radica, básicamente, en la utilidad que cada vez más gente dedicada a la producción de software le da como un estándar para el modelado de sistemas, lo que implica su consolidación como un lenguaje universal de modelado para sistemas computacionales y con ello, contar con un mismo significado sin depender del país o región donde se elabore.

Un caso del uso es la descripción de un conjunto de secuencias de acciones, incluyendo las variantes, que un sistema realiza para ofrecer un resultado observable o tangible a un determinado usuario. Gráficamente, el caso del uso se representa como una elipse[51]. Existen algunos términos más que complementan la definición de un caso de uso. En la figura 3.2 se muestra la representación de un caso de uso.

Representación del Caso de Uso (Figura 3.2)

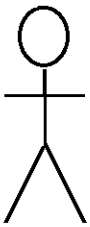


[50] El Lenguaje de Modelamiento Unificado (UML - Unified Modeling Language) es un lenguaje estándar para escribir modelos de software. Puede ser utilizado para visualizar, especificar, construir y documentar artefactos de un sistema de software. El UML es solamente una lengua y así es solamente una parte de un método del desarrollo del software. El UML es independiente del proceso, aunque óptimamente este debe ser utilizado en un proceso conducido por casos del uso, arquitectura-céntrica, iterativo e incremental. Grady Booch, James Rumbaugh, Ivar Jacobson. Unified Modeling Language User Guide. Publisher: Addison Wesley. Chapter 2 Introducing the UML.

[51] Grady Booch, James Rumbaugh, Ivar Jacobson. Unified Modeling Language User Guide. Publisher: Addison Wesley. Capítulo 16. Use Cases. Terms and Concepts.

Actor, representa los posibles roles que puede jugar un usuario cuando interactúa con un caso de uso. Un actor puede representar a un humano, un dispositivo de un equipo de cómputo u otros sistemas o subsistemas.

Representación de un Actor (Figura 3.3)



Cliente

Flujos y Eventos. Un caso del uso describe lo que hace un sistema (o un subsistema, una clase, o un interfaz) pero no especifica cómo lo hace. A la hora de modelar, es importante que se vea claramente la separación entre las operaciones que están dentro y fuera del alcance. Se debe especificar el comportamiento del caso de uso describiendo un flujo de acontecimientos en un texto. Cuando se describe este flujo, se debe incluir como y cuando el caso de uso comienza, describir las relaciones entre los diferentes agentes y los objetos que intercambian, así como el flujo básico y los alternativos dependiendo el comportamiento.

Escenario. Representa uno de los posibles flujos de una operación dependiendo de sus variantes.

A grandes rasgos estos son los elementos que componen un diagrama de casos de uso, para conocer mayor detalle sobre las diferentes técnicas de elaboración de estos diagramas es necesario consultar las bibliografías:

- Grady Booch, James Rumbaugh, Ivar Jacobson. Unified Modeling Language User Guide. Publisher: Addison Wesley. Capítulo 16. Use Cases y Capitulo 17. The Use Case Diagrams.

- Martin Fowler, Kendall Scott. “UML Distilled Second Edition A Brief Guide to the Standard Object Modeling Language”. Segunda Edición Agosto 18, 1999. Editorial Addison Wesley. Capítulo 3. Use Case.

La generación de la documentación de los casos de uso y sus respectivos diagramas debe realizarse por un equipo que contemple personal con conocimientos amplios del negocio y experiencia en la generación de casos de uso, como práctica de RUP.

Uno de los objetivos de la presente guía es mostrar una forma eficiente de administrar los requerimientos de un proyecto de software. Para la administración de requerimientos se propone la disciplina de requerimientos que utiliza RUP, y la cual tiene como metas:

- Establecer y mantener el acuerdo con los clientes y otros stakeholders en lo que debe hacer el sistema y porqué.
- Proveer a los desarrolladores del sistema con una mejor comprensión de los requerimientos.
- Definir los límites del sistema (delimitar los alcances).
- Proporcionar una base para planear el contenido técnico de las iteraciones.
- Proporcionar una base para estimar tiempo y costos del desarrollo del sistema.
- Definir un interfaz de usuario del sistema, centrándose en las necesidades y las metas de los usuarios.

Estas metas se alcanzan definiendo una visión de sistema, para traducir esta a su vez a un modelo de casos de uso[52], con especificaciones suplementarias y definiendo a mayor detalle los requerimientos del software. Además, debe existir una administración adecuada de los atributos de los requerimientos para ayudar a manejar el alcance y los cambios dentro del sistema.

Requerimientos

Se ha hablado de los requerimientos, pero ¿Cuál es la definición de un requerimiento? O ¿Que es un requerimiento? A continuación se define el término requerimiento:

“Definimos un requerimiento como una condición o capacidad con la cual un sistema debe

[52] Un caso del uso es una descripción de un conjunto de secuencias de acciones, incluyendo las variantes, que un sistema realiza para ofrecer un resultado observable o tangible a un determinado usuario. Gráficamente, el caso del uso se representa como una elipse. Grady Booch, James Rumbaugh, Ivar Jacobson. Unified Modeling Language User Guide. Publisher: Addison Wesley. Capítulo 16. Use Cases. Terms and Concepts.

conformarse[53]". Con base en esta definición, podemos decir que un requerimiento es un conjunto de características que debe cumplir el sistema, o bien, especifica las acciones que debe poder realizar el sistema. Existen dos tipos de requerimientos, los funcionales y no funcionales.

"Los requerimientos funcionales son utilizados para expresar el comportamiento de un sistema especificando las condiciones de la entrada y de la salida que se espera que resulten.[54]"

Es decir, son todas aquellas acciones que debe realizar el sistema y que son requeridas por el usuario, especificando para ello los datos de entrada y de salida en cada acción.

Adicionalmente, un sistema debe contar con una variedad de cualidades de calidad que no estén descritas específicamente por los requerimientos funcionales del sistema. A estos se les denomina requerimientos no funcionales. Y tienen que ver con características que de una u otra forma puedan limitar el funcionamiento del sistema, como por ejemplo, el rendimiento (en tiempo y espacio), interfaces de usuario, fiabilidad (robustez del sistema, disponibilidad de equipo), mantenimiento, seguridad, portabilidad, estándares, etc.

- *Utilidad*. Los requerimientos de utilidad tratan del factor estético humano (interfaces de usuario, la facilidad de aprender y utilizar el sistema), la consistencia en la interfase, la documentación del usuario y los materiales de capacitación.

- *Confiabilidad*. Los requerimientos de confiabilidad tratan de la frecuencia y severidad de fallas, de la forma de recuperación, prevención, y de la exactitud.

- *Funcionamiento*. Los requerimientos de funcionamiento imponen condiciones sobre requerimientos funcionales. ejemplo, un requerimiento que especifique la tarifa de la transacción, la velocidad, la disponibilidad, la exactitud, el tiempo de reacción, el tiempo de la recuperación, o el uso de la memoria con el cual una acción dada debe ser realizada.

- *Soportabilidad*. Los requerimientos de Soportabilidad tratan la capacidad de probar, de

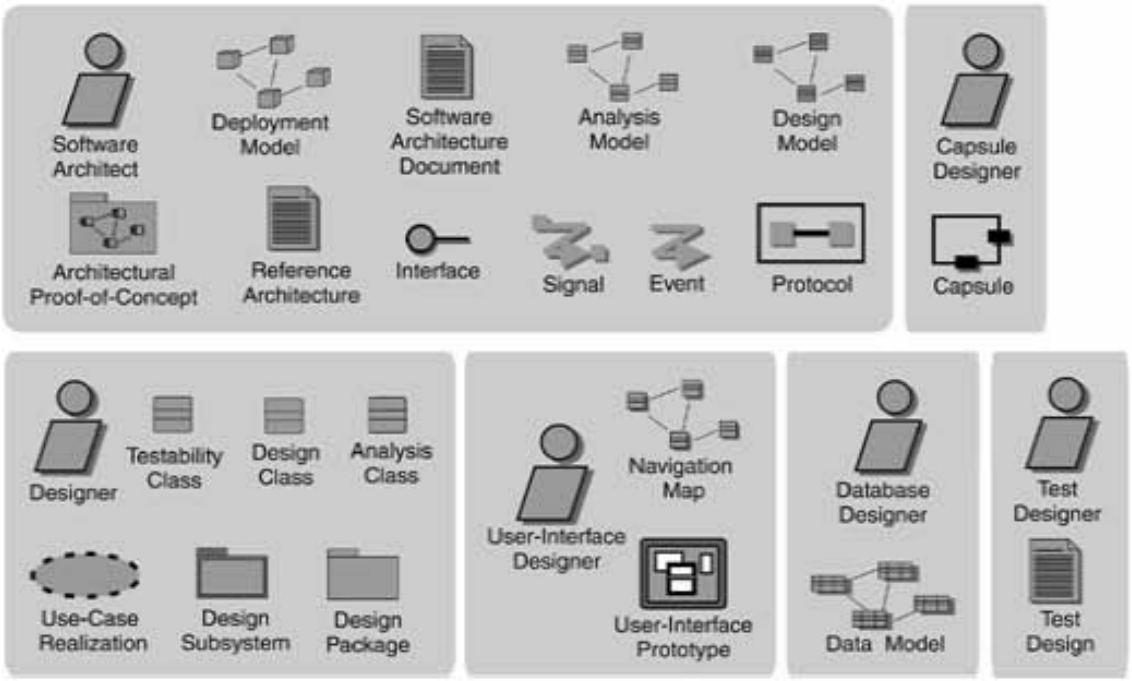
[53] Philippe Kruchten.The.rational unified process an introduction 3rd edition. Ed. Addison Wesley. Diciembre 2003, E.U.A.
[54] Philippe Kruchten.The.rational unified process an introduction 3rd edition. Ed. Addison Wesley. Diciembre 2003, E.U.A. p. 158.

mantenimiento, y las otras cualidades requeridas para mantener el sistema actualizado después de su lanzamiento. Los requerimientos de soportabilidad no están impuestos necesariamente ante el sistema, sino que por el contrario refieren a menudo al proceso usado para crear el sistema o los varios artefactos del proceso del desarrollo del sistema. Un ejemplo es el uso de un estándar específico de la codificación de C++.

Para tener un mejor detalle de los requerimientos a analizar y construir, se propone la descomposición de estos en features (características o rasgos), donde un feature es la unidad mínima de medida para un requerimiento. Un caso de uso, entonces esta compuesto de un conjunto de features. El número de casos de uso determinaría el tamaño del software.

RUP expresa los procesos de análisis y diseño en términos de roles, artefactos, actividades, y workflow (flujos de trabajo).

Roles y artefactos en el análisis y diseño[55]. (Figura 3.4)



Los principales roles para los procesos de análisis y diseño son:

[55] Philippe Kruchten.The.rational unified process an introduction 3rd edition. Ed. Addison Wesley. Diciembre 2003, E.U.A. p. 175.

- Arquitecto de Software. El arquitecto del software conduce y coordina actividades y los artefactos técnicos a través del proyecto
- Diseñador. El diseñador define las responsabilidades, las operaciones, las cualidades, y las relaciones de una o varias clases y determina cómo deben ser ajustadas al ambiente de implementación. Además, el diseñador puede tener responsabilidad de unos o más paquetes del diseño o subsistemas del diseño, incluyendo cualquiera de las clases poseídas por los paquetes o los subsistemas

Adicionalmente puede incluir los siguientes roles:

- Diseñador de la base de datos: El diseñador de la base de datos es necesario cuando el sistema que se esta diseñado incluye una base de datos.
- Diseñador de la cápsula (para los sistemas en tiempo real): El diseñador de la cápsula es una clase de diseñador que se centre en asegurarse de que el sistema puede responder a los acontecimientos de una manera oportuna con el uso apropiado de las técnicas de diseño concurrentes.
- Revisor de la arquitectura y revisor del diseño: Estos especialistas repasan los artefactos dominantes producidos con este workflow.

Como complemento a estos roles y sus actividades, cabe mencionar que los responsables de la realización del análisis, deben ser los mismos que se encargaran del diseño. Estos deben supervisar el uso del lenguaje de modelado en las actividades de análisis y diseño, tanto para los clientes como para las personas a su cargo.

Los artefactos que considera RUP principales como parte de los entregables para las fases de análisis y diseño son:

- El modelo de diseño. Que es el modelo principal para el sistema bajo construcción.
- El documento de la arquitectura del software. Que captura las varias vistas arquitectónicas del sistema

En este apartado se han propuesta la utilización de las siguientes prácticas:

- Cliente en Sitio – XP.
- Utilización de Casos de Uso para documentación de requerimientos – RUP.
- Utilización de UML como lenguaje de modelado – RUP.
- Desarrollo basado en componentes (definición de componentes) – RUP, MSF.
- Uso de Feature como unidad mínima de medida pera los requerimientos – FDD**.**
- Análisis y diseño en términos de roles, artefactos, actividades, y workflow – RUP.
- Disciplina de requerimientos. – RUP.

3.3 Diseño

El propósito de la fase de diseño es adaptar los resultados del análisis a las restricciones impuestas por los requerimientos no funcionales, el ambiente de implementación, requerimientos de funcionamiento, etc. El diseño es un refinamiento del análisis. Se centra en la optimización de los procesos del sistema mientras se asegura de la cobertura completa de los requerimientos.

"El diseñar es crear una estructura que organice la lógica en el sistema. Un buen diseño ordena la lógica de modo que un cambio en una parte del sistema no siempre requiera un cambio en otra parte del mismo. Un buen diseño se asegura de que cada parte de la lógica en el sistema tenga un y solamente un lugar. Un buen diseño coloca la lógica cerca de los datos sobre los que opera. Un buen diseño permite la extensión del sistema con cambios en solamente un lugar[56]."

De lo mencionado por Kent Beck en su libro, se puede concluir que el diseño de un producto de software debe contemplar las siguientes características:

- *Escalabilidad.* El diseño se debe realizar basado en componentes, de tal forma que el agregar nueva funcionalidad no implique la reconstrucción del sistema, es decir, solo debe bastar agregar la nueva funcionalidad en la parte adecuada.

- *Simplicidad.* Se deben diseñar soluciones simples en lo posible, es decir: *Diseños apropiados para el nivel tecnológico de la gente a codificarlo*. El diseño debe ser lo suficientemente comprensible para la gente que va a trabajar en el. De que sirve tener un diseño "elegante" si la gente que va codificarlo no lo entiende. *Comunicación.* Todas las

[56] Kent Beck, Extreme Programming Explained – Embrace Change. Ed. Addison Wesley. 1999. pp. 43-44.

ideas que necesitan ser comunicadas se deben representar en el diseño. Todos los elementos del sistema se comunican a los futuros lectores. *Diseñar con los menos elementos posibles, e*ntre menos componentes diseñados, será menor el tiempo y recursos a utilizar para probar y documentar.

- *Mantenimiento.* Al igual que para la escalabilidad, el diseño basado en componentes permite tener una mayor control sobre los diferentes elementos que componen la aplicación, y por tal motivo es mas fácil ubicar y corregir los errores.

- *Reutilización / No duplicidad.* Diseñar con base en factores (factorización). Siempre hay que diseñar teniendo presente los diseños anteriores para evitar la duplicidad de funcionalidad. Esto ayudara a la reutilización y diseño de rutinas genéricas de uso común. La duplicidad de la lógica o de la estructura hace código difícil de entender y de modificarse.

Al igual que para el análisis, el diseño se debe realizar utilizando como herramienta de modelado UML (Unified Modeling Language). Para esta fase se contemplan la elaboración de los siguientes diagramas:

- *Diagramas de interacción.* El diagrama de interacción describe una interacción. Consiste en un sistema de objetos y sus relaciones, incluyendo los mensajes que se pueden intercambiar entre ellos. Representa la forma en como un Cliente (Actor) u Objetos (Clases) se comunican entre si en petición a un evento. Esto implica recorrer toda la secuencia de llamadas, de donde se obtienen las responsabilidades claramente.

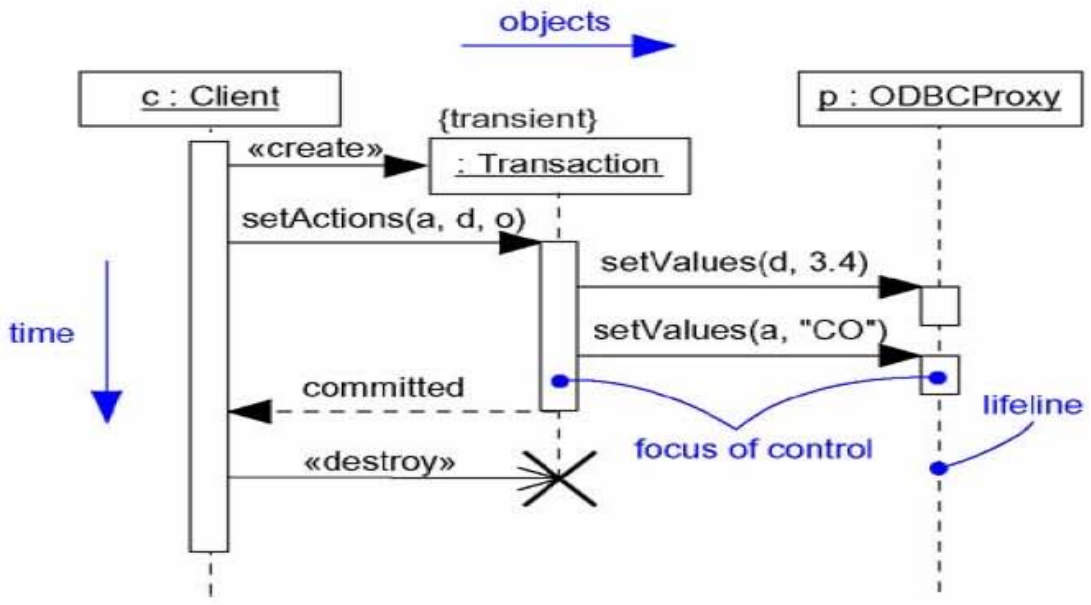
  Los diagramas de la interacción se utilizan para modelar los aspectos dinámicos de un sistema. Esto implica modelar casos concretos o prototípicos de las clases, interfaces, componentes, y nodos, junto con los mensajes que se envían entre ellos, todos en el contexto de un panorama que ilustre un comportamiento.

  Con los diagramas de interacción se pueden modelar los aspectos dinámicos de un sistema. Este proceso es difícil, especialmente si se habla de sistemas distribuidos con eventos concurrentes en varias direcciones. La mejor forma de modelar estos aspectos es construyendo tablas con historias de los procesos con sus diferentes alternativas,

implicando la interacción de los objetos y los mensajes que se pueden enviar entre ellos. Este proceso de modelado se llama en UML diagramas de interacción.

- *Diagrama de Secuencia.* Un diagrama de secuencia es un diagrama de interacción que hace énfasis en el orden y tiempos de los mensajes. Gráficamente, un diagrama de secuencia es una tabla que muestra los objetos y mensajes disponibles lo largo del eje X, y las peticiones en los distintos tiempos, a lo largo del eje del eje Y.

Ejemplo de un diagrama de secuencia. [57]. (Figura 3.5)



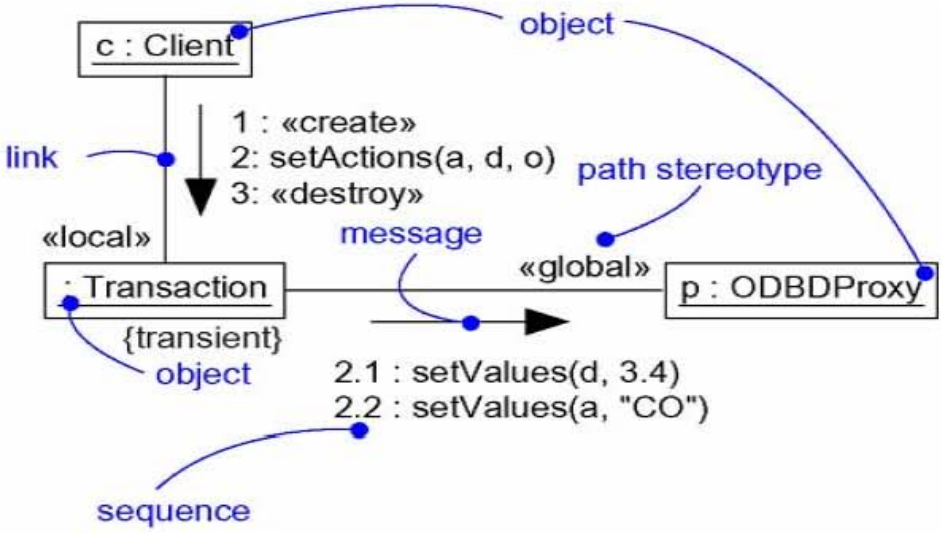
Como se puede observar en la figura 3.5, y pensando que se esta en un plano cartesiano, en el eje X se colocan las clases que tendrán participación en el proceso descrito, así mismo se dibujan los mensajes que estarán intercambiando las distintas clases. En el eje Y se indica el orden y momento en el cual se realizaran estos intercambios de mensajes. En el ejemplo observamos que el proceso inicia de una instancia **c** de tipo Cliente, la cual instancia un objeto de tipo Transacción y le asigna valores mediante la operación setActions(a, d, o), esta instancia de Transacción a su vez hace uso de un objeto **p** de tipo ODBCProxy al cual se le asignan valores con la operación setValues() en dos ocasiones. Una vez realizada esta operación retorna

[57] Grady Booch, James Rumbaugh, Ivar Jacobson. Unified Modeling Language User Guide. Publisher: Addison Wesley. Octubre 1998. E.U.A.

un mensaje de persistido al cliente c, este destruye la instancia del objeto Transacción.

- *Diagrama de Colaboración.* un diagrama de la colaboración es un diagrama de la interacción que hace énfasis en la organización estructural de los objetos que envían y reciben mensajes. Gráficamente, un diagrama de la colaboración es una colección de vértices y arcos.

Ejemplo Diagrama de colaboración[58]. (Figura 3.6)



Como se puede observar en la figura 3.6, en esta se definen los objetos a participar en el proceso, se pintan las relaciones, la dirección del flujo y de forma muy sencilla se enumera los pasos a seguir durante este proceso.

Ambos diagramas describen a buen nivel de detalle el proceso que se esta diseñando, por lo tanto, la generación de cualquiera de ellos durante esta fase, es suficiente para tener un buen diseño.

[58] Grady Booch, James Rumbaugh, Ivar Jacobson. Unified Modeling Language User Guide. Publisher: Addison Wesley. Octubre 1998. E.U.A.

En el libro Unified Modeling Language User Guide. Publisher: Addison Wesley. Capitulo 18. Interaction Diagrams, de Grady Booch, James Rumbaugh, Ivar Jacobson. Se puede encontrar más referencia sobre la forma de crear estos diagramas.

- *Diagrama de Clases.* Es un diagrama que muestra un conjunto de clases, interfases, y colaboraciones y sus relaciones. Gráficamente, un diagrama de clases es una colección de vértices y arcos[59]. Los diagramas de clases son los más comunes diagramas utilizados para el modelado de objetos orientados a sistemas. Se utilizan para modelar una vista de un sistema.

  Los diagramas de clases comúnmente se componen de:

  - *Clases y Objetos.* Un objeto es una representación de una entidad, ya sea real o conceptual, con límites bien definidos y con significado dentro de un modelo. Cada objeto en un modelo se caracteriza por su estado, su comportamiento y su identidad. El estado de un objeto es una de las posibles condiciones bajo las que el objeto puede existir. El estado de un objeto cambia con el tiempo y está definido por un conjunto de propiedades (atributos), por los valores de esas propiedades y por las relaciones que dicho objeto puede tener con otros objetos. El comportamiento de un objeto determina la forma en que responde ante peticiones de otros objetos, y tipifica todo lo que el objeto puede hacer. El comportamiento de un objeto se materializa en el conjunto de operaciones definidas para dicho objeto. La identidad implica que cada objeto es único, incluso si su estado es idéntico al de otro objeto.

    En UML una clase es una descripción de un grupo de objetos con propiedades comunes (atributos), comportamiento común (operaciones), relaciones comunes y semántica común. Por tanto, una clase es una plantilla (*témplate*) para la creación de objetos, donde cada objeto de dicha clase será una instancia de ella (no pudiendo ser instancia de más de una clase).

    Las clases se documentan con una descripción de lo que hacen, sus métodos y sus atributos. Las relaciones entre clases se documentan con una descripción de su

[59] Grady Booch, James Rumbaugh, Ivar Jacobson. Unified Modeling Language User Guide. Publisher: Addison Wesley. Octubre 1998. E.U.A.
.

propósito, su cardinalidad (cuantos objetos intervienen en la relación) y su opcionalidad (cuando un objeto es opcional el que intervenga en una relación).

- *Interfaces*. Una interfaz (*interface*) especifica un conjunto de operaciones en un elemento del modelo que describe el comportamiento visible de dicho elemento fuera del elemento. La representación gráfica es una línea terminada en un círculo.

- *Colaboraciones*. Una colaboración es una sociedad de clases, interfaces, y otros elementos que trabajan juntos para proporcionar un cierto comportamiento cooperativo que sea más grande que la suma de todas sus piezas. Una colaboración es también la especificación de cómo un elemento, tal como un clasificador (clase, interfaz, componente, nodo, o caso del uso) o una operación, es observada por un sistema de clasificadores y de asociaciones que desempeñan los papeles específicos usados en una manera específica.

- *Relaciones de dependencias, de generalización y de asociación*. Una relación en un diagrama de clases, se representa mediante una línea que une dos o más clases. La línea puede ir acompañada de diferentes tipos de adornos que definen su semántica y características. Los elementos adicionales que pueden aparecer en la representación de una relación son:

  - Rol: Identifica con nombres a los elementos que aparecen en los extremos de la línea que denota la relación, dicho nombre describe la semántica que tiene la relación en el sentido indicado. Por ejemplo, la asociación de composición entre *Maquina* e *Ingrediente* recibe el nombre de *existencias* como rol en ese sentido.
  - Multiplicidad: Indica la cardinalidad de la relación. Por ejemplo: *1*, *1..**, *5*, *, como indicadores de multiplicidad.

  Una asociación binaria se representa mediante una línea sólida que une dos clases, se trata de una relación entre las dos clases no muy fuerte, es decir, no se exige dependencia existencial.

La agregación (*aggregation*) es una forma especial de asociación que especifica una relación todo-parte entre el agregado (todo) y una parte que lo compone. Una agregación se representa mediante un rombo en el extremo “todo'' de la relación.

Una agregación de composición o simplemente composición (*composite aggregation*) es una agregación más fuerte que implica:

- Dependencia existencial: El elemento dependiente desaparece al destruirse el que lo contiene y, si es de cardinalidad 1, es creado al mismo tiempo.
- Pertenencia fuerte: Se puede decir que el objeto contenido es parte constitutiva y vital del que lo contiene.
- No compartición: Los objetos contenidos no son compartidos, esto es, no forman parte del estado de otro objeto.

La composición se representa mediante un rombo relleno del lado de la clase que contiene a la otra en la agregación.

El concepto de herencia define otro tipo de relación entre clases (generalización) donde una clase comparte estructura y/o comportamiento con una o más clases. El término superclase se refiere a la clase que guarda la información común, mientras que el término subclase se refiere a cada uno de los descendientes de la superclase.

A partir de las relaciones de generalización se crea una jerarquía de abstracción en la cual una subclase puede heredar de una o más superclases. Es por esto que a la herencia también se le suele denominar jerarquía es “una” o “clase de”.

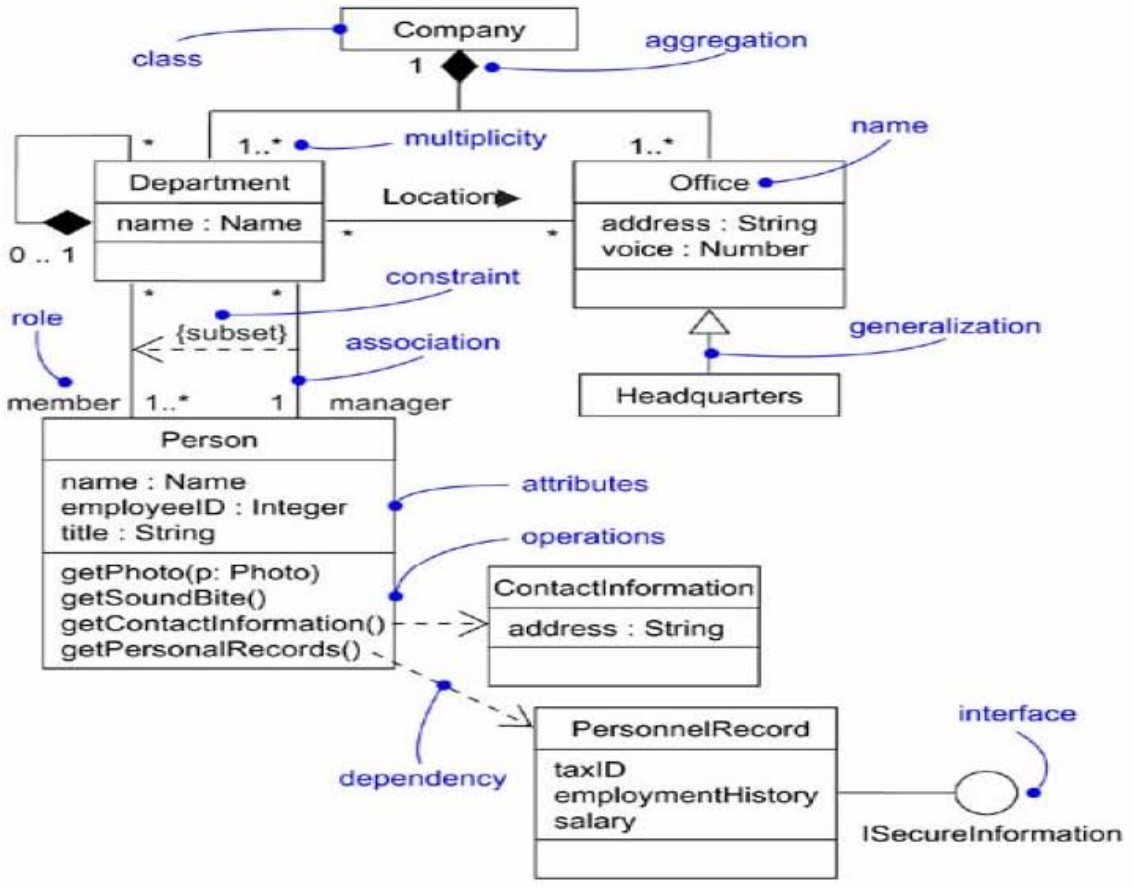
Existen dos formas básicas de identificar herencia en un modelo, por generalización o por especialización:

- *Generalización*: Proporciona la capacidad de crear superclases que encapsulan la estructura y el comportamiento común a varias clases.
- *Especialización*: Proporciona la capacidad de crear subclases que representan refinamientos de una superclase, generalmente añadiendo estructura y comportamiento a las nuevas subclases. Se añaden subclases

para especializar el comportamiento de clases ya existentes. Las operaciones heredadas pueden ser re-escritas por una subclase, lo que conocemos como polimorfismo, pero por el contrario, la subclase no debería proporcionar una estructura o comportamiento menor que el de su superclase.

La dependencia indica que una clase utiliza a otra como argumento en la definición de una operación.

Ejemplo de diagrama de Clases (Figura 3.7)



La tarea principal que se debe llevar a cabo durante esta fase, y siempre en las primeras iteraciones del proceso, es la definición y diseño del framework o arquitectura[60] que va a soportar

[60] El framework o arquitectura para el desarrollo de un producto de software, se puede definir como el conjunto de componentes genéricos que sirven de base para la construcción e intercomunicación de las diferentes capas de una aplicación.

los procesos de negocio. Esta actividad la debe realizar el área de arquitectura, describiendo de forma detallada su funcionamiento y la forma de uso para el diseño y codificación de los componentes de negocio.

Uno de los problemas mas comunes durante las diferentes fases del proceso de producción de software es la falta de comunicación entre el equipo de trabajo, este problema se resuelve mediante la realización de reuniones diarias, así se conocen los avances y problemas encontrados durante su día laboral. Estas reuniones deben ser cortas en tiempo, no más de una hora, y enfocarse básicamente a recopilar la información sobre los problemas que afectan el avance del proceso. Las reuniones deben realizarse al final del día laboral.

Otra de las prácticas que debe adoptarse como parte para la mejora en el proceso de producción de un software, es el uso de patrones de diseño, que a continuación se define:

"Los patrones hacen referencia a la comunicación de problemas y soluciones. Simplemente, los patrones permiten conocer y documentar problemas recurrentes, y su solución en un contexto particular, y así comunicar este conocimiento a otros. Uno de los elementos mas importantes de la definición anterior es la *recurrencia*, puesto que la meta del patrón es fomentar la reutilización conceptual en otro momento diferente[61]."

La documentación existente de los problemas y soluciones, relacionados a la producción de software, ayudan a disminuir en gran medida el riesgo de fracaso durante este proceso. Al reutilizar soluciones probadas a problemas resueltos se disminuye la incertidumbre y se acelera la producción. Esta guía, propone la utilización de los patrones de diseño como parte de esta fase. Para conocer los diferentes patrones que se pueden aplicar a la producción de software se recomienda la siguiente bibliografía:

- Deepak_Alur, John_Crupi, Dan_Malks. Core J2EE™ Patterns: Best Practices and Design Strategies. Ed. Pearson Education; 1ra edición (Junio 26, 2001).

[61] Deepak_Alur, John_Crupi, Dan_Malks. Core J2EE™ Patterns: Best Practices and Design Strategies. Ed. Pearson Education; 1ra edición (Junio 26, 2001), p. 17.

A continuación se listan las prácticas propuestas para esta fase, las cuales fueron descritas en detalle anteriormente:

- Diseño escalable. – XP.
- Diseño simple. – XP.
- Diseño basado en componentes. - RUP, XP, MSF.
- Reutilización de diseño. – XP.
- Uso de UML como lenguaje de modelado. – RUP.
- Generación de Diagramas de iteración y clases. – RUP.
- Revisiones diarias de trabajo. - Scrum.
- Utilización de Patrones de Diseño y Desarrollo. - RUP, XP, MSF

3.4 Implementación

La fase de implementación consiste en la codificación de los componentes de negocio analizados y diseñados en las fases anteriores.

La disciplina de implementación debe cubrir algunos puntos primordiales, los cuales se nombran a continuación:

- *Definir la organización del código en términos de los subsistemas a implementar y organizados en capas.* Se deben definir de forma clara los paquetes que conformara la aplicación, así como sus posibles capas. Esta tarea de definición la realiza el área de arquitectura y allí especifica los paquetes para los diferentes módulos a implementar y las capas por las cuales estará constituida. Regularmente se dividen en tres capas, presentación, negocio y datos. La capa de presentación es la interfase con la que el usuario final interactuará, la capa de negocio esta compuesta de toda aquella lógica del negocio que se esta trabajando y la capa de datos es aquella que contiene la lógica para la persistencia de la información en la base de datos.

- *Integrar el código producido individualmente o por equipo en un único sistema ejecutable.* La tarea de integración es una de las mas importantes de esta fase de implementación, en ella se junta el código elaborado por los diferentes módulos y/o personas del equipo. Esta

integración permite encontrar con anticipación problemas de comunicación entre los diferentes módulos y/o personas que integran el equipo. Más adelante se dará una definición mas detallada de este proceso.

- *Incorporar al cliente como parte del equipo de implementación.* Esta práctica permite tener a una persona que conozca el negocio y que pueda agilizar la aclaración de dudas y la tomar decisiones en caso de indefiniciones. Se trata de incorporar en el equipo de implementación a personas que tengan un conocimiento avanzado del negocio y que cuenten con la jerarquía suficiente para la toma de decisiones (autorizar cambios).

- *Manejar estándares de codificación.* Es importante que antes de iniciar la codificación de los componentes de negocio se definan los lineamientos de codificación. La definición de estos debe estar a cargo del área de arquitectura del proyecto. En estos se deben definir las reglas a seguir para la codificación, tales como:

    - nomenclatura a usar para dar nombre a las variables, clases, métodos, funciones, constantes, etc.
    - El tamaño en líneas de los métodos, clases, funciones.
    - El nivel de detalle de los comentarios en el código.

  El manejo de estándares permite una lectura fácil y clara del código para cualquier integrante del equipo de desarrollo. Ayuda a no tener dependencias de código sobre una o varias personas en especial.

- *Generar los casos de prueba antes de codificar y probar los componentes desarrollados como unidades.* Es muy sencilla esta práctica, y básicamente trata de realizar la codificación de los métodos o clases a construir antes de construirlos. Esto obliga al programador a conocer mejor el contexto de la va a construir. El programador debe generar un programa de prueba donde se haga uso del método a construir, al realizar la prueba esta debe arrojar un error. A continuación y con base en el diseño, debe realizar la codificación del método y ejecutar el programa de prueba. Se realiza la revisión del código en caso de error y nuevamente se ejecuta la prueba, esto se hará hasta que la prueba pase con éxito.

- *Codificación por capas.* Dependiendo del número de capas que tendrá la aplicación, se debe dividir el equipo de desarrollo en sub-equipos especializados por capa y designar a una persona como responsable de codificación para esta capa. Esto provoca que exista mayor especialización de los programadores en determinada herramienta y por consecuencia mayor rapidez y tiempos más cortos de codificación. La persona designada como responsable debe tener un mayor nivel tecnológico (en programación dependiendo la capa) a fin de poder apoyar y resolver los problemas más complejos que se presenten.

- *Realizar integraciones de código en forma frecuente.* Esta práctica se encarga de mantener las diferentes partes de código libres de problemas de contratos o de comunicación. Entre mas frecuentes sean estas integraciones mas acoplado se tendrá el código de la aplicación. Permite identificar y resolver en forma temprana los posibles problemas que se puedan presentar por indefiniciones de contratos. Además mantiene el código actualizado, de tal forma que se pueden generar versiones con mayor rapidez.

- *Propiedad colectiva de código.* Cualquier integrante del equipo tiene acceso a todo el código de la aplicación, esto permite no tener dependencia de una persona en particular para un proceso específico. Y aun que la asignación de tareas es personal, se debe involucrar a más de una, en los procesos importantes.

- *40 horas a la Semana.* Es importante no permitir que el equipo de trabajo labore más de 40 horas a la semana, de ocurrir así, se generan circunstancias que afectan gravemente el proceso de codificación. Entre algunas de estas se encuentran: estrés, cansancio, actitudes de rebeldía o pasivas, y conflictos personales en el grupo, lo que provoca una baja en el redimiendo general.

- *Refactorizacion (Refactoring).* La práctica de refactorización trata sobre la reestructura del sistema removiendo el código duplicado, mejorando la comunicación, simplificando y agregando flexibilidad a los procesos.

Adicionalmente existen dos actividades frecuentes y muy importantes durante esta fase: la generación de *builds* y las *integraciones*.

Un buid es una versión operacional del sistema o parte de un sistema que muestra un subconjunto de funcionalidades a ser contenidas en el producto final. Durante un proceso iterativo de producción de software se realizan muchos *builds*. La constante generación de builds permite tener puntos de revisión y control de código, y ayuda a detectar problemas de integración en los componentes.

La generación de builds debe ser controlada dependiendo del tamaño de funcionalidad a liberar y las prioridades en incidencias. Se propone la generación de un *build* por semana hasta el término de la iteración. Esta forma de generar los avances de funcionalidad permite mantener un eficiente control de versiones, de tal modo que se pueda regresar a una versión anterior en caso de que sea necesario.

La integración es la actividad donde se realiza la combinación, unión de los diferentes componentes de una aplicación. Esta se da en dos niveles:

- Para integrar el trabajo de un equipo especifico. Este tipo de integración se debe realizar diariamente al final del día. El código que es integrado en este nivel debe estar libre de errores de compilación, no es necesario que cumpla por completo con los requisitos del negocio y debe contener su caso de prueba.

- Para integrar subsistemas en un sistema completo. Regularmente los proyectos de producción de software, dependiendo del tamaño de su funcionalidad, se dividen en varios módulos. Estos módulos muchas veces trabajan en forma paralela y es inevitable que llegué un momento en que se integren para interactuar entre si. Este tipo de integración se debe realizar, por lo menos, una vez por semana o dependiendo del avance alcanzado en un periodo delimitado de tiempo. El código integrado debe estar libre de errores de compilación, es importante integrar funcionalidad completa de los diferentes requisitos, ya que a partir de esta se realiza un build, el cual se instalara en un ambiente para realizarle pruebas.

Debe existir un repositorio único de código donde se administre la historia de los cambios realizados en cada integración y/o build. Existen diversas herramientas para la administración de código, entre las más populares se encuentran Visual Source Safe (VSS) y Concurrent Versions

System (CVS); Estas permiten la administración de código fuente y documentación de una forma eficiente y sencilla.

La persona encargada de realizar estos build es conocida como administrador de la configuración, y tiene como responsabilidad la generación de builds y la integración de código. Regularmente esta persona forma parte del equipo conocido como de "arquitectura[62]".

Como parte de la propuesta para esta fase, las prácticas descritas se listan a continuación:

- Definir la organización del código en términos de los subsistemas a implementar y organizados en capas – RUP
- Integrar el código producido individualmente o por equipo en un único sistema ejecutable. – RUP
- Incorporar al cliente como parte del equipo de implementación. – XP, RUP.
- Manejar estándares de codificación. – XP.
- Generar los casos de prueba antes de codificar y probar los componentes desarrollados como unidades. – XP.
- Codificación por capas. – XP.
- Realizar integraciones de código en forma frecuente. – XP.
- Propiedad colectiva de código. – XP
- 40 horas a la Semana. – XP.
- Generación de Builds. – Open Source.
- Uso de un repositorio único de código. – Open Source.
- Revisión continúa de código. – XP.
- Refactorizacion. – XP.

3.5 Implantación

Es la último fase del proceso de producción de software, y su tarea principal es la de poner en un ambiente operacional la aplicación desarrollada. Esto implica considerar todos los elementos necesarios para su uso exitoso. Generalmente, la gente dedicada a la producción de software

[62] El equipo de arquitectura es el encargado de definir los componentes genéricos que van a soportar la aplicación en su conjunto. Se encarga de proveer los medios necesarios para el diseño, implementación, prueba y puesta en producción del producto de software terminado.

subestima la importancia de esta fase, piensan que una vez terminada la codificación de la aplicación, esta va a trabajar por si sola, sin embargo no es así, la fase implica tareas críticas, y dependiendo de su adecuada ejecución, permitirá en forma exitosa la puesta en marcha del sistema computacional. A continuación se listan las actividades a realizar como parte de esta fase:

- Prueba del software en su ambiente operacional final (prueba beta).
- Empaquetado del software para la entrega.
- Distribución del software.
- Instalación del software.
- Capacitación de los usuarios y de la fuerza de ventas.
- Migración del Software existente o de la bases de datos.

La realización de estas actividades varía extensamente a través de la industria del software, dependiendo del tamaño del proyecto, del modo de la entrega, y del contexto del negocio.

La mayoría de las actividades mencionadas anteriormente forman parte de las prácticas que se proponen en las diferentes metodologías aquí analizadas y que se retoman como parte de la propuesta para la mejora del proceso.

3.6 Mantenimiento

La fase de mantenimiento involucra cambios al software con el fin de corregir defectos y dependencias encontradas durante su uso, así como la adición de nueva funcionalidad para mejorarlo.

El proceso de producción de software que se esta proponiendo como parte de esta guía metodológica, es de tipo iterativo. Este proceso esta enfocado a la programación orientada a objetos, y para el cual no se hace una mención explicita de la fase de mantenimiento. Sin embargo y conociendo de la importancia del mismo, se dará una descripción y posible justificación de la omisión de esta fase dentro del proceso.

Como se pudo observar en los capítulos anteriores, la mayoría de las prácticas descritas en estas fases tienen la finalidad de reducir al mínimo el mantenimiento del software. A continuación se

listan los tipos de mantenimiento a un software:

- *Perfectivo*: Mejora del software (rendimiento, flexibilidad, reusabilidad, etc.) o implementación de nuevos requisitos.
- *Adaptativo*: Adaptación del software a cambios en su entorno tecnológico (nuevo hardware, otro sistema de gestión de bases de datos, otro sistema operativo, etc.)
- *Correctivo*: Corrección de fallos detectados durante la explotación.
- *Preventivo*: Facilitar el mantenimiento futuro del sistema (verificar precondiciones, mejorar legibilidad, etc.).

La mayoría de las prácticas descritas en las fases de análisis, diseño e implementación buscan la prevención de los posibles problemas a enfrentar en esta fase. Estas prácticas proponen diversas estrategias para facilitar la actividad de mantenimiento, entre ellas se tiene:

- Desarrollo basado en componentes (definición de componentes) – RUP, MSF.
- Análisis y diseño en términos de roles, artefactos, actividades, y workflow – RUP.
- Definir la organización del código en términos de los subsistemas a implementar y organizados en capas – RUP
- Disciplina de requerimientos. – RUP.
- Uso de un repositorio único de código. – Open Source.
- Manejar estándares de codificación. – XP.
- Utilización de Patrones de Diseño y Desarrollo. - RUP, XP, MSF
- Codificación por capas. – XP.
- Refactorizacion. – XP.

3.7 Evaluación de la propuesta

Para la evaluación de esta propuesta se retomara el criterio con el cual se realizo la calificación de las metodologías analizadas en el apartado 2.7 del capítulo 2. Como se recordara, el criterio utilizado se refiere a calificar a las metodologías con base en el nivel en que contemplen prácticas para el control de las variables:

- Definición de requerimientos.
- Iteración con el cliente.
- Tiempos de codificación.
- Número de errores detectados en el producto.
- Tiempo consumido y esfuerzo planeado.
- Planificación del proyecto.
- Capacitación de Recursos Humanos.
- Administración de cambios.

De lo anterior y retomando la tabla 2.8 del capítulo 2, se tiene:

Calificación de variables por metodología y propuesta[63]. (Tabla 3.1)

	METODOLOGIAS			
VARIABLES	RUP	XP	MSF	Propuesta Metodológica
Definición de Requerimientos	5	5	3	5
Iteración con el cliente	4	3	0	5
Tiempos de codificación	0	5	0	5
Numero de errores	4	5	1	5
Planificación del proyecto	4	4	3	5
Capacitación de recursos humanos	0	3	0	3
Administración de cambios	5	0	0	5
Tiempo consumido y esfuerzo planeado	4	0	3	4

[63] Elaboración propia.

Como se puede observar en la tabla anterior, la combinación de las tres metodologías proporciona una propuesta que contempla en mayor medida el control de las variables definidas. Hay que tener en cuenta que en esta tabla no se muestran las calificaciones para las metodologías Scrum, Open Source, Cristal Family y Feature Driven Development (FDD), ya que estas están consideradas como complementarias.

A continuación se listan las ventajas de la utilización de la propuesta metodológica:

- Contar con los elementos de control suficientes para todas las actividades del proceso de producción de software (En un 90% aproximadamente). Lo que las metodologías por separado no contemplan.

- Debido a que la propuesta esta basada en la metodología Extreme Programming (XP), la cual es de acceso público (licencia no requerida), y además no depende de alguna herramienta en particular, el costo de aplicación podría ser nulo, dependiente del costo de las herramientas a utilizar para su complementación (Existen herramientas de licencia gratuita para estos procesos). En contraste, las metodologías RUP y MSF, cuentan con un costo elevado de licenciamiento para la utilización y capacitación en el detalle de sus procesos, aunado al uso obligado de herramientas comerciales para funcionar correctamente.

- La utilización de UML como lenguaje de modelado para el análisis y diseño de los componentes, proporciona una gran flexibilidad en la codificación, incluso en fábricas de software situadas en lugares distantes a la elaboración del diseño.

Para finalizar esta evaluación, se muestra a continuación el análisis FODA de la propuesta resultante de la presente investigación:

FORTALEZAS	**OPORTUNIDADES**
▪ La propuesta contempla las prácticas más importantes de las principales metodologías de la actualidad. ▪ La propuesta cubre cada una de las etapas de la producción de software, con base en las mejores prácticas. ▪ Bajo costo de implementación de la propuesta metodológica. ▪ Bajo costo de capacitación, debido a la sencillez de su diseño.	▪ Captación de Mercados internacionales, principalmente Latinoamérica. Aunque también existen posibilidades en Estados Unidos y Europa. ▪ Consolidación de México como un país productor de software. ▪ Existencia de un déficit de profesionales en ingeniería de software en Estados Unidos de América, por lo tanto, posibles plazas de trabajo para México. ▪ Condiciones idóneas para la detonación de la industria del software en México.
DEBILIDADES	**AMENAZAS**
▪ Falta de una capacitación integral sobre procesos y herramientas de trabajo. (incluida la propuesta metodológica) ▪ Salarios incongruentes con las capacidades individuales, lo que provoca un constante movimiento de recursos humanos dentro del medio. ▪ Falta de políticas para el seguimiento profesional de los recursos humanos. (Ascensos, promociones, capacitación, etc.) ▪ Poca especialización de recursos humanos.	▪ Cancelación de apoyo gubernamental a la industria del software. (Tanto local como federal) ▪ Falta de capacitación sobre la propuesta metodológica. ▪ Falta de capacitación y certificación de los ingenieros en herramientas actuales. ▪ Déficit de profesionales en ingeniería de software. ▪ Falta de actualización en planes de estudio para las carreras relacionadas a la ingeniería de software.

3.8 Escenarios de la propuesta

Uno de los puntos más importantes del presente trabajo de investigación, es sin duda, el análisis de los escenarios futuros para la industria dedicada a la producción de software en México. Estos estarán divididos en intervalos de 5 años y abarcarán un lapso de 15 años como máximo.

Para el análisis de los escenarios se tomaran en cuenta el comportamiento de las dos siguientes variables principales:

- *La adopción de nuevas tecnologías.* Como se pudo visualizar a lo largo de estos tres capítulos, el avance tecnológico en la industria del software evoluciona rápidamente, por lo que los procesos para su producción (metodologías) diariamente son corregidos y ajustados. Así mismo se construyen y mejoran herramientas (lenguajes de programación) encaminadas en este mismo sentido. Hace apenas algunas décadas no se concebía un proceso sistemático para la producción de un software, se realizaban programas independientes de gran tamaño (cientos de líneas) que permitían la realización de una sola operación. Con la aparición de la computadora personal, surgieron los primeros sistemas operativos[64] y junto con ellos herramientas cada vez más sofisticadas que permiten generar lógica de un proceso de negocio con el fin de optimizarlo en su tiempo de operación.

  Es por todo lo anterior, que las empresas mundiales que se encuentren capacitadas en las últimas tecnologías para la producción de un software, siempre tendrán mayor probabilidad de éxito frente a sus competidores. Por lo tanto, para el caso particular de México, que no cuenta con una base como productor de software, el primer paso es la actualización continua sobre las nuevas tecnologías (llámense lenguajes de programación, metodologías de producción, lenguajes de modelado, herramientas de apoyo, etc.) que le permitan el desarrollo, la madurez y establezcan las bases para una industria competitiva.

- *Los programas gubernamentales en apoyo a la industria del software en México.* En la actualidad, el papel que juegan los gobiernos en los diferentes rubros de la economía es fundamental para su desarrollo. Hace algunos años, varios países emprendieron esfuerzos

[64] Sistema Operativo. Programa de software encargado de administrar los recursos de una computadora.

para detonar la industria del software e incorporarla a su economía como una de sus principales actividades. Los ejemplos mas concretos de la implementación de estas estrategias se pueden ver en los países de la India e Irlanda, que a raíz de esto se han consolidado como parte de los principales productores a nivel mundial. Es por esto, que la importancia para México de implementar una estrategia similar para detonar la industria de software. El nivel de apoyo que se le inyecte a esta industria será un factor condicionante para su crecimiento en el país.

Las variables secundarias a revisar con base al comportamiento de las variables principales antes mencionadas, se describen a continuación:

Los programas gubernamentales de apoyo a la industria del software y la capacidad de las empresas para mantener a su equipo de trabajo actualizado en las tendencias tecnológicas a nivel mundial, propiciara la utilización mas completa de las metodologías:

1. Rational Unified Process (RUP). Esta metodología permite implementar seis de las mejores prácticas de la ingeniería de software: desarrollo iterativo (Ciclo de Vida), administración de requerimientos, uso de arquitecturas basadas en componentes, modelado visual a través de UML, verificación de la calidad y control de cambios.

2. Extreme Programming (XP). Esta metodología, es de las más exitosas por sus prácticas: el juego de la planeacion, cliente como parte del equipo, programación en pareja, pequeñas entregas, diseño simple, pruebas y la orientación muy marcada por el crecimiento profesional del recurso humano.

3. Microsoft Solution Framework (MSF). Por su parte, la implementación de esta metodología permite hacer uso de una serie de modelos, principios y guías para el diseño y desarrollo de soluciones.

4. Otras metodologías. Existen algunas otras metodologías que mediante su implementación aportan certidumbre al proceso de producción de un software, entre ellas se encuentran: Open Source, crystal family, Srum y feature driven development. Es importante considerarlas dentro de los escenarios aquí propuestos.

Y como consecuencia se tendrá mayor certidumbre sobre el éxito del proceso.

Escenario "Te quiero para siempre".

Este escenario cubre la relación más optimista entre las variables primarias. El gobierno de México se encarga de aplicar programas para el fortalecimiento de la industria del software, mientras tanto las empresas mexicanas crean programas de capacitación para mantener al día a su personal en las tendencias tecnológicas. Algunas de las condiciones para cumplir este escenario son:

- Existencia de apoyos por parte de los gobiernos federal y estatal para incentivar la industria del software, estos son los encargados de generar los mecanismos necesarios para la detonación de la industria en México.
- Inversión en infraestructura tecnológica, informática como de comunicaciones.
- Revisión y actualización de los planes de estudio para las carreras universitarias afines a la tecnología informática.
- Exploración de los mercados internacionales (nichos de mercado en el mundo).
- inversión en el sector.
- Modifican de leyes para proveer una legislación adecuada.
- Las empresas mexicanas emprenden medidas para mantener actualizados sus recursos humanos en las tendencias tecnológicas. Existe una adopción rápida de las nuevas tecnologías.
- Las empresas mexicanas apoyan a sus recursos para certificarse en las herramientas de trabajo.
- Existen remuneraciones acordes a los niveles internacionales, lo que impide la fuga de cerebros y fortalece la investigación en el sector.

Escenario "Última llamada".

Como su nombre lo dice, es la ultima llamada para lograr una unión de fuerzas entre el sector publico y privado. No existe una planeacion común para la aplicación de medidas que fortalezcan el sector. Existe una continuidad en las políticas de apoyo al sector a través de los sexenios, sin embargo las empresas no asimilan este apoyo para mejorar y actualizar sus procesos. A continuación se listan algunas de las condiciones que harían cumplir este escenario:

- Existencia de apoyos por parte de los gobiernos federal y estatal para incentivar la industria del software, estos son los encargados de generar los mecanismos necesarios para la detonación de la industria en México.
- Inversión en infraestructura tecnológica, informática como de comunicaciones.
- Revisión y actualización de los planes de estudio de las carreras universitarias afines a la tecnología informática.
- Exploración de los mercados internacionales. (nichos de mercado en el mundo)
- inversión en el sector.
- Modifican de leyes para proveer una legislación adecuada.
- Las empresas mexicanas no emprenden medidas para mantener actualizados a sus recursos humanos en las tendencias tecnológicas (esta tecnología es aprendida lentamente).
- Las empresas mexicanas no apoyan a sus recursos para certificarse en las herramientas de trabajo.

Escenario "Sin Control".

Este escenario es el más pesimista de todos y por lo tanto, es en el que a México no le conviene estar. Debido a los cambios de gobierno, se rompe con la continuidad en las políticas de apoyo a la industria del software, mientras tanto a las empresas mexicanas no les interesa el desarrollo de sus recursos humanos y prefieren mantenerse en una postura mediocre. No les importa el crecimiento, y antes que arriesgar, se mantienen en un estado pasivo perdiendo los nichos de mercado internacionales. Las condiciones que permiten este escenario son:

- No hay apoyos por parte de los gobiernos federal y estatal para incentivar la industria del software.
- No existe inversión en infraestructura tecnológica.
- La actualización de los planes de estudio en las carreras universidades afines a la tecnología informática, no se llevan a cabo al ritmo requerido que la industria lo demanda a nivel mundial.
- No hay exploración de los mercados internacionales.
- No existe una legislación adecuada que promueva la inversión.
- Las empresas mexicanas no emprenden medidas para mantener actualizados a sus recursos humanos en las tendencias tecnológicas. La adopción de nuevas tecnologías es lenta.

- Las empresas mexicanas no apoyan a sus recursos para certificarse en las herramientas de trabajo.

Escenario "Frágiles".

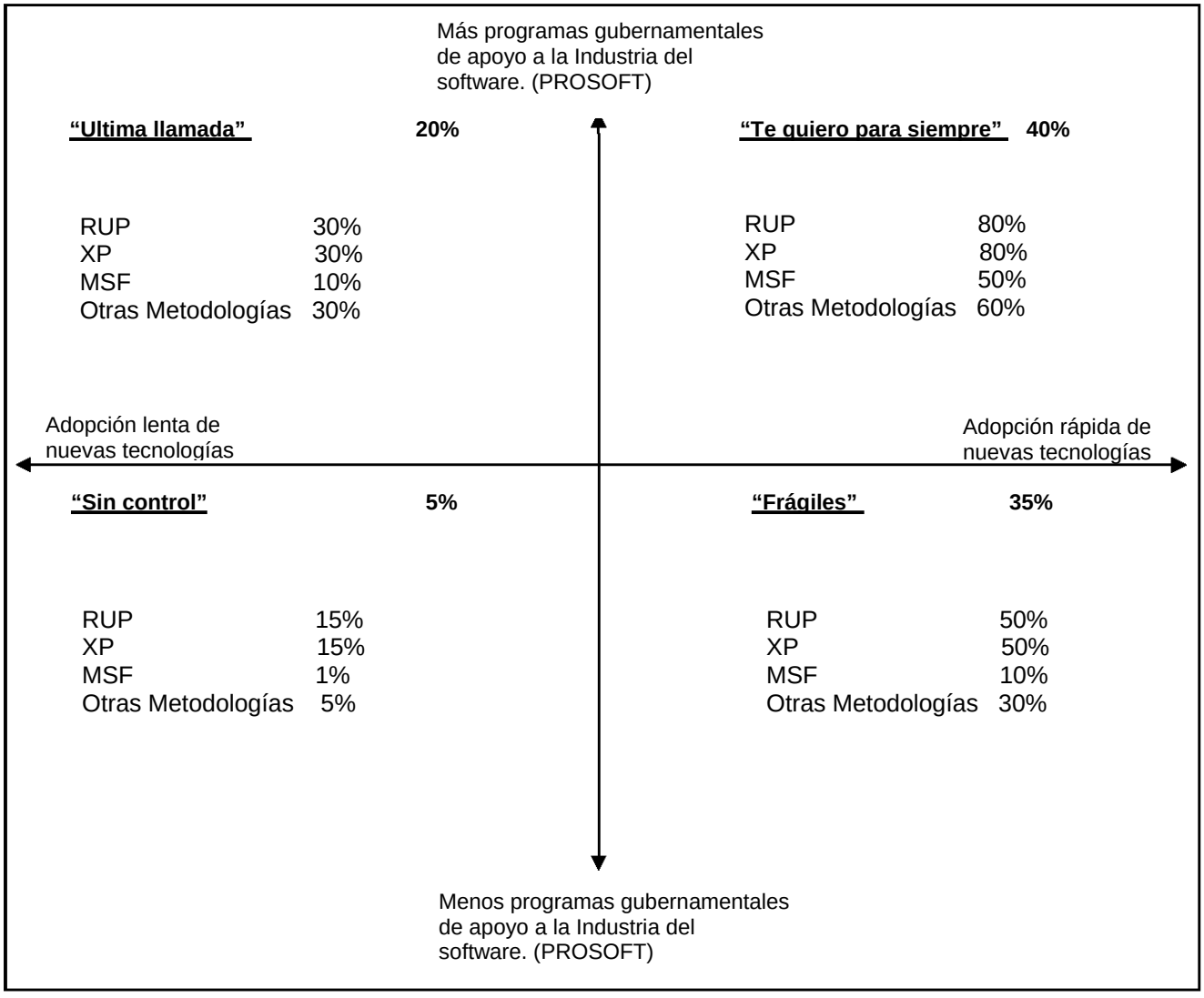
Este es el escenario que se vislumbra más propicio para caso el mexicano. Una de las causas del estancamiento económico en México es la falta de continuidad en los programa gubernamentales a través de los sexenios. Cada vez, la disputa política por llegar al poder se vuelve mas engorrosa, y los nuevos gobernantes se olvidan del bien común y se enfocan al beneficio propio. En consecuencia, cada nuevo gobierno piensa que las políticas económicas y sociales implementadas por el gobierno anterior son obsoletas, y antes que darle un seguimiento y verificar a través de una investigación científica sus resultados, decide terminarlas para implementar las propias. Esta falta de continuidad en el gobierno propicia que las empresas mexicanas, como comúnmente se dice, se rasquen con sus propias uñas, y se encarguen de mantenerse al día en las tendencias tecnológicas como única oportunidad de competencia hacia el exterior. Las condiciones que propician este escenario son:

- No existe un seguimiento a los programas de apoyo por parte de los gobiernos federal y estatal para incentivar la industria del software.
- Las empresas mexicanas invierten en infraestructura tecnológica por cuenta propia. La actualización es más lenta.
- Las universidades realizan las adecuaciones necesarias para estar al día en sus planes de estudio.
- Las empresas mexicanas se promueven por cuenta propia en los mercados internacionales. Son pocas las reconocidas a nivel mundial.
- Las empresas mexicanas emprenden programas de capacitación y certificación para mantener actualizados a sus recursos humanos.
- Existen remuneraciones acordes a los niveles de certificación de los recursos humanos.

Escenarios para el año 2010.

El siguiente cuadro muestra los escenarios posibles para el año 2010 con base en las variables descritas anteriormente.

Escenarios posibles al año 2010 (cuadro 2).



Como se puede observar en el diagrama anterior, la tendencia al año 2010 es tener un equilibrio entre el apoyo gubernamental y los esfuerzos individuales de las empresas mexicanas para estar al día en la tecnología informática. Esto dará como resultado mantener el crecimiento de la industria en el mercado local e internacional. Por otro lado y con un porcentaje muy similar, se

observa una tendencia a esfuerzos individuales de las empresas para mantenerse dentro del mercado, en contraste, el cambio de gobierno en el año 2006 incrementa las posibilidades de perder el seguimiento al PROSOFT y con esto perder otra gran oportunidad para el crecimiento del sector.

A continuación se describe el comportamiento de las variables secundarias con respeto a estos escenarios.

Escenario *"Te quiero para siempre".* Probabilidad de que ocurra 40%.

Rational Unified Process (RUP).	Cada vez más empresas hacen uso, casi en su totalidad, de las prácticas de RUP, cuentan con programas del gobierno para alcanzar una madurez en sus procesos : ▪ Uso del lenguaje UML como estándar para modelado. ▪ Desarrollo de la producción de software a través de un ciclo iterativo. ▪ Diseño del producto orientado a componentes. ▪ Análisis de requerimientos a través de Casos de Uso.
Extreme Programming (XP).	Cada vez más empresas hacen uso, casi en su totalidad, de las prácticas de XP, cuentan con programas del gobierno para alcanzar una madurez en sus procesos : ▪ El juego de la planeación. Iteración constante entre el cliente y los desarrolladores. ▪ Entregables cortos. Generación de entregables del producto en tiempos más cortos. Por lo menos uno cada semana. ▪ Diseños sencillos. Diseñar soluciones prácticas, que puedan entender los programadores. ▪ Elaboración de escenarios de pruebas unitarias,

	programación y ejecución de las mismas. ▪ Refactoring. Reestructura del sistema para eliminar duplicidad, tener mayor simplicidad y agregar flexibilidad. ▪ Integración continúa de código. La integración del código generado se realiza varias veces al día, permitiendo tener un repositorio actualizado y libre de conflictos. ▪ Establecer una cultura de trabajo, 40 horas a la semana, permite mejorar la calidad de vida y el rendimiento de los recursos humanos. ▪ Manejo de estándares de codificación ▪ Cliente dentro del proyecto, disponible a cualquier momento para aclarar requerimientos de negocio.
Microsoft Solutions Framework (MSF).	Cada vez más empresas hacen uso, en un buen porcentaje de las prácticas de MSF, y cuentan con programas del gobierno para alcanzar una madurez en sus procesos : ▪ Desarrollo de la producción de software a través de un ciclo iterativo.
Otras Metodologías.	Empresas hacen uso de más y mejores prácticas en apoyo al proceso de producción de Software, y cuentan con programas del gobierno para alcanzar una madurez en sus procesos : ▪ Utilizan herramientas Open Source para modificarlas y adaptarlas a sus necesidades. ▪ Uso de herramientas más flexibles para llevar a cabo la administración del código (CVS - version control system) ▪ Hacen uso de patrones de diseño, lo que permite acortar tiempos de entrega. No es necesario afrontar

	problemas que alguien ya resolvió y obtuvo buenos resultados. ▪ División de los Casos de Uso en Features (Características), como unidad mínima de medición de funcionalidad.

Escenario *"Frágiles".* Probabilidad de que ocurra 35%.

Rational Unified Process (RUP).	Las empresas hacen uso en un 50% de las prácticas de RUP, y cuentan con poca ayuda del gobierno para madurar sus procesos: ▪ Uso del lenguaje UML como estándar para modelado. ▪ Desarrollo de la producción de software a través de un ciclo iterativo. ▪ Diseño del producto orientado a componentes. ▪ Análisis de requerimientos a través de Casos de Uso.
Extreme Programming (XP).	Las empresas hacen uso en un 50% de las prácticas de XP, y cuentan con poca ayuda del gobierno para madurar sus procesos: ▪ El juego de la planeación. Iteración constante entre el cliente y los desarrolladores. ▪ Entregables cortos. Generación de entregables del producto en tiempos más cortos. Por lo menos uno cada semana. ▪ Diseños sencillos. Diseñar soluciones prácticas, que puedan entender los programadores. ▪ Elaboración de escenarios de pruebas unitarias, programación y ejecución de las mismas. ▪ Refactoring. Reestructura del sistema para eliminar duplicidad, tener mayor simplicidad y agregar

	flexibilidad. ▪ Integración continúa de código. La integración del código generado se realiza varias veces al día, permitiendo tener un repositorio actualizado y libre de conflictos. ▪ Establecer una cultura de trabajo, 40 horas a la semana, permite mejorar la calidad de vida y el rendimiento de los recursos humanos. ▪ Manejo de estándares de codificación ▪ Cliente dentro del proyecto, disponible a cualquier momento para aclarar requerimientos de negocio.
Microsoft Solutions Framework (MSF).	Empresas hacen uso en un 10% de las prácticas de MSF, y cuentan con poca ayuda del gobierno para madurar sus procesos: ▪ Desarrollo de la producción de software a través de un ciclo iterativo.
Otras Metodologías.	Empresas hacen uso en un 30% de más y mejores prácticas en apoyo al proceso de producción de Software. ▪ Utilizan herramientas Open Source para modificarlas y adaptarlas a sus necesidades. ▪ Uso de herramientas más flexibles para llevar a cabo la administración del código (CVS - version control system) ▪ Hacen uso de patrones de diseño, lo que permite acortar tiempos de entrega. No es necesario afrontar problemas que alguien ya resolvió y obtuvo buenos resultados. ▪ División de los Casos de Uso en Features (Características), como unidad mínima de medición de funcionalidad.

Escenario *"Ultima llamada".* Probabilidad de que ocurra 20%.

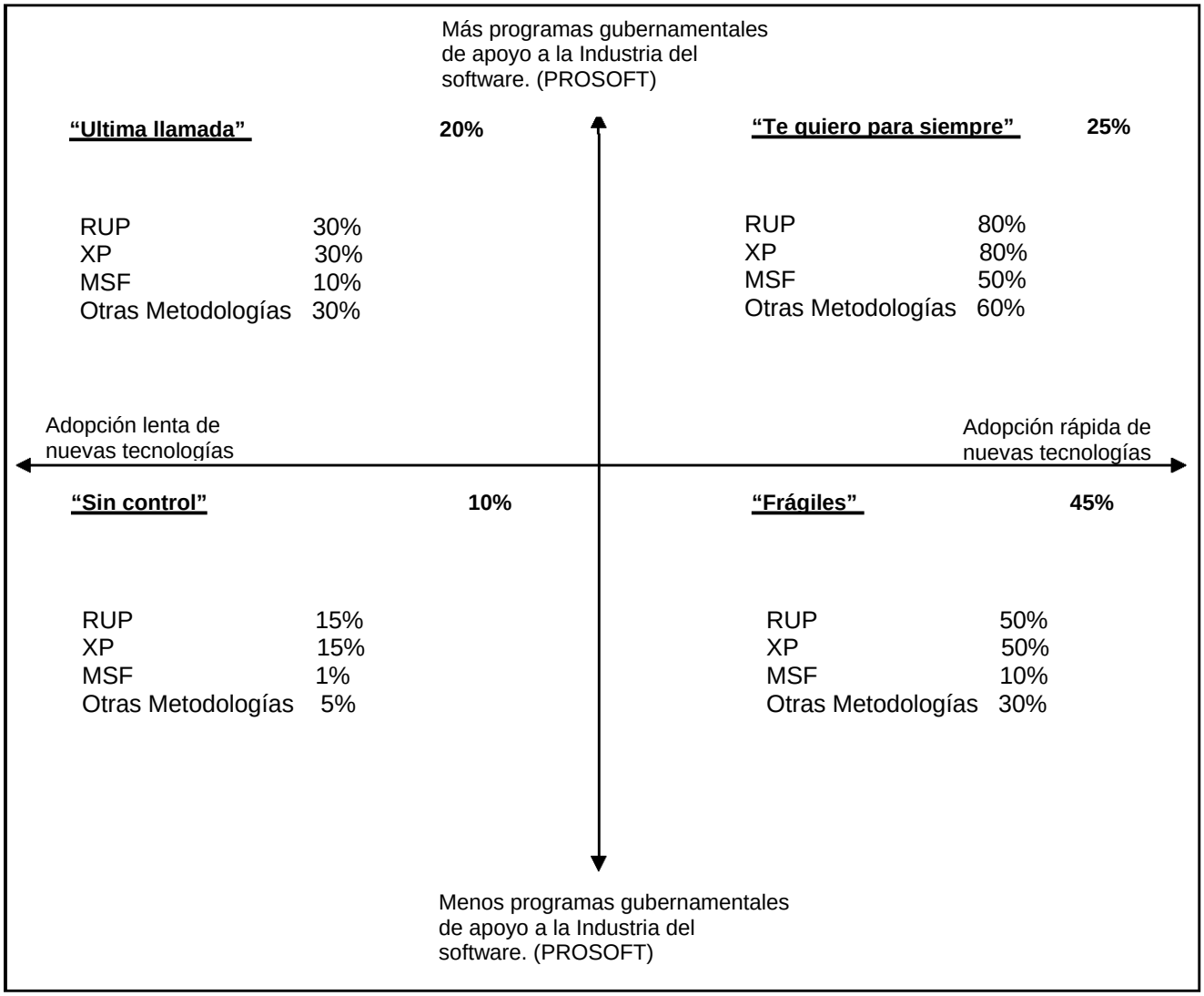
Rational Unified Process (RUP).	El interés de las empresas por hacer uso de las prácticas de RUP es poca, a pesar de la existencia de apoyos gubernamentales solo se utilizan en un 30% sus prácticas.
Extreme Programming (XP).	Al igual que para RUP, las empresas hacen uso en un 30% de las prácticas de XP, a pesar de la existencia de apoyos gubernamentales.
Microsoft Solutions Framework (MSF).	La utilización de las prácticas más importantes de MSF es de un 10%, por parte de las empresas de producción de software.
Otras Metodologías.	Cada vez menos empresas hacen uso de prácticas en apoyo al proceso de producción de Software provenientes de las metodologías de Scrum, Open Source, Cristal family y Feature Driven Development.

Escenario *"Sin control".* Probabilidad de que ocurra 5%.

Rational Unified Process (RUP).	El uso de las prácticas de RUP es casi nulo. Solo se utiliza en un 10%, lo que provoca que el proceso se encuentre fuera de control.
Extreme Programming (XP).	XP es utilizado en un 10%, existe casi nulo control sobre el proceso. No existe interés por las empresas para adoptar nueva tecnología y el gobierno no apoya esto.
Microsoft Solutions Framework (MSF).	La utilización de las prácticas más importantes de MSF es prácticamente nula.
Otras Metodologías.	La utilización de prácticas en apoyo al proceso de producción de Software provenientes de otras metodologías como Scrum, Open Source, cristal family y Feature Driven Development, es casi nula para este escenario.

Escenarios para el año 2015.

Escenarios posibles al año 2015 (cuadro 3).

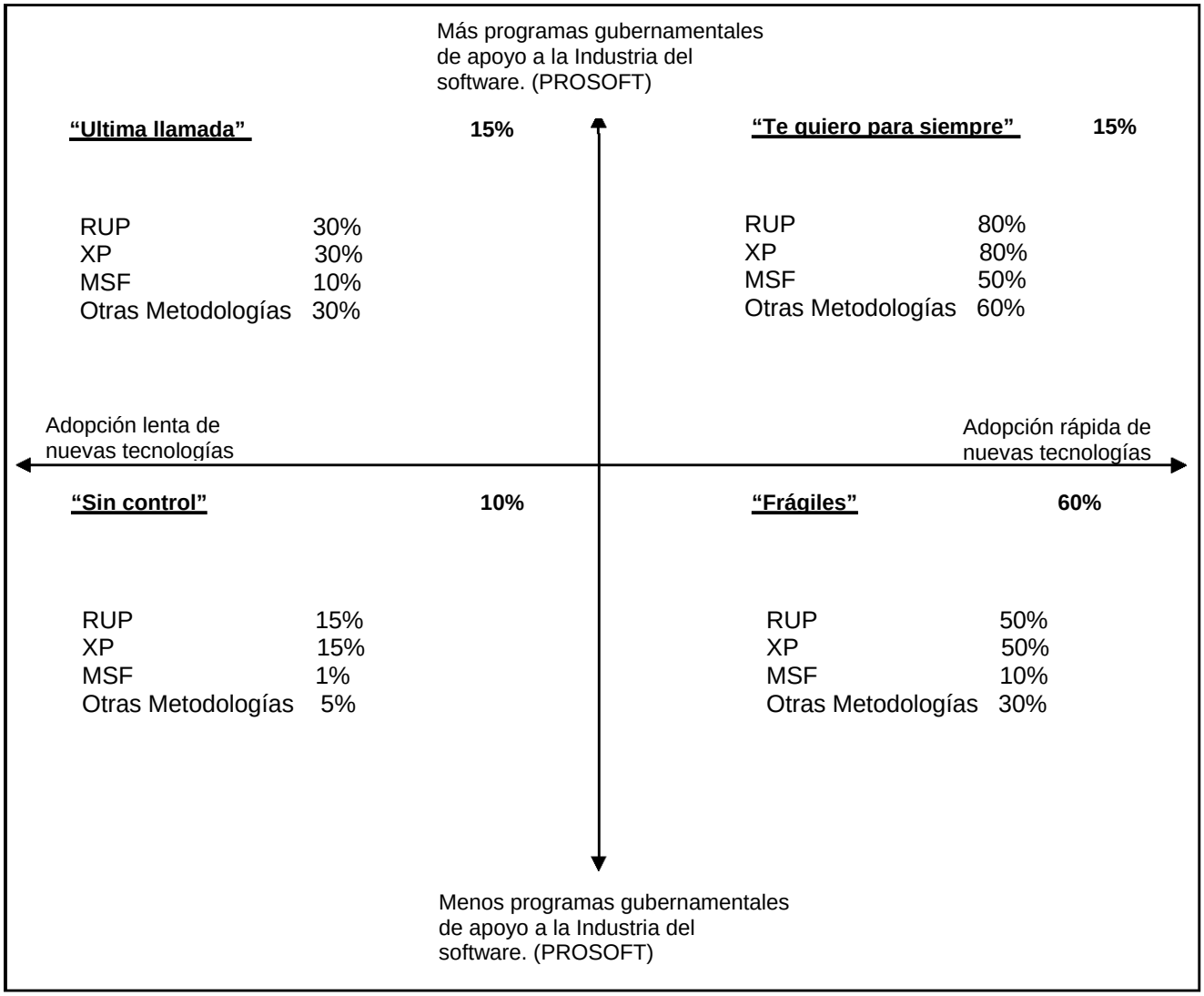


Debido a la falta de planeación en el gobierno, en los cambios de sexenio, los programas iniciados por la administración anterior son desechados y definidos otros para el nuevo ciclo. Esta falta de seguimiento provoca que diferentes áreas de la economía caigan constantemente en recesión. La tendencia al año 2015 se dirige precisamente a este fenómeno. Basta dar un vistazo a las experiencias del pasado para asegurar que es muy dicifil que un gobierno nuevo le dará continuidad a un programa de un gobierno anterior. Con base en estas circunstancias, y como se puede observar en el diagrama anterior, la tendencia al año 2015 es la disminución del apoyo

gubernamental y una lucha individual de las empresas por actualizarse y mantenerse en el mercado.

Escenarios para el año 2020.

Escenarios posibles al año 2020 (cuadro 4).



Conforme el tiempo avanza, se puede observar una tendencia marcada hacia la incertidumbre en la cuestión de la implementación de apoyos gubernamentales, en contraste, los esfuerzos individuales de las empresas sigue siendo el factor fundamental para mantener en la competencia

a la industria del software en México. Esto es lo que se puede observar en el cuadro anterior, y que a continuación se detalla para sus diferentes variables secundarias.

La principal aportación que arroja la creación de escenarios, es el seguimiento de un problema a lo largo de un periodo determinado de tiempo, lo cual permite mitigar la incertidumbre y contar con los elementos necesarios para emprender acciones que lo aminoren y en el caso perfecto, que lo resuelvan. Para este trabajo de investigación, los escenarios, permiten observar las tendencias de la industria dedicada a la producción de software en México a 15 años, analizar sus aciertos y deficiencias y contar con los elementos para modificar algún comportamiento erróneo.

Conclusiones

La complementación de las diferentes prácticas de las metodologías RUP, XP y MSF permite controlar las principales variables que afectan el proceso de producción de software:

- Definición de Requerimientos.
- Iteración con el cliente.
- Tiempos de codificación.
- Número de errores.
- Planificación del proyecto.
- Capacitación de recursos humanos.
- Administración de cambios.

Y por consecuencia mejorar el proceso de producción del mismo. Además, el uso de las prácticas de las metodologías llamadas "Ágiles", las cuales fueron descritas en el capítulo dos en el apartado 2.6, complementadas con las tres metodologías en estudio, permite reforzar algunos otros puntos del proceso, tales como el análisis, el diseño y la implementación.

Como se vio en el apartado 3.2 del capítulo tres, el uso de herramientas de modelado, consolidadas como estándar internacional, permite un mayor entendimiento de nuevas tecnologías. Entre las herramientas más importantes propuestas para su uso, en las diferentes fases del proceso de producción de software, se encuentra el lenguaje de modelado unificado (UML), el cual se ha convertido en el estándar mundial para el modelado de sistemas de software.

El uso de patrones de diseño permite acortar tiempos y garantizar la solución más eficiente para un proceso particular de un software. Los patrones de diseño son una serie de soluciones a problemas que comúnmente se repiten. Durante algunos años estos se han documentado con base en problemas prácticos en diferentes proyectos de producción de software, y se han difundido con el fin de utilizarlos para mejorar el proceso. (Apartado 3.3 Diseño, Capitulo 3)

Como se observó en el capítulo 3, del trabajo de investigación, la mayoría de las prácticas propuestas en esta guía pertenecen a las metodologías RUP y XP, ya que son estas las que proporcionan la mayor cobertura sobre las variables aquí tratadas.

Las prácticas propuestas en esta guía, para la mejora del proceso de producción de software son:

Fase	Práctica	Metodología
Planificación		
	Desarrollo Iterativo	XP, RUP, Scrum, FDD, MSF
	Juego de la planeación	XP
	Un Plan general y varios de iteración para el proceso.	XP, RUP
Análisis		
	Cliente en Sitio	XP
	Utilización de Casos de Uso para documentación de requerimientos	RUP
	Utilización de UML como lenguaje de modelado	RUP
	Desarrollo basado en componentes (definición de componentes)	RUP, MSF
	Uso de ***Feature*** como unidad mínima de medida pera los requerimientos	FDD
	Análisis y diseño en términos de roles, artefactos, actividades, y workflow	RUP
	Disciplina de requerimientos	RUP
Diseño		
	Diseño escalable	XP
	Diseño simple	XP
	Diseño basado en componentes	RUP, XP, MSF
	Reutilización de diseño	XP
	Uso de UML como lenguaje de modelado	RUP
	Generación de Diagramas de iteración y clases	RUP
	Revisiones diarias de trabajo	Scrum
	Utilización de Patrones de Diseño y Desarrollo	RUP, XP, MSF

Implementación		
	Definir la organización del código en términos de los subsistemas a implementar y organizados en capas	RUP
	Integrar el código producido individualmente o por equipo en un único sistema ejecutable	RUP
	Incorporar al cliente como parte del equipo de implementación	XP, RUP
	Manejar estándares de codificación	XP
	Generación de casos de prueba antes de codificar y realizar la prueba de los componentes desarrollados como unidades	XP
	Codificación por capas	XP
	Realizar integraciones de código en forma frecuente	XP
	Propiedad colectiva de código	XP
	40 horas a la Semana	XP
	Generación de Builds	Open Source
	Uso de un repositorio único de código	Open Source
	Revisión continúa de código	XP
	Refactorización	XP
Implantación		
	Prueba del software en su ambiente operacional final (prueba beta)	RUP, XP, MSF
	Empaquetado del software para la entrega.	RUP, XP, MSF
	Distribución del software	RUP, XP, MSF
	Instalación del software	RUP, XP, MSF
	Capacitación de los usuarios y de la fuerza de ventas	RUP, XP, MSF
	Migración del Software existente o de la bases de datos	RUP, XP, MSF

Mantenimiento		
	Desarrollo basado en componentes (definición de componentes).	RUP, MSF
	Análisis y diseño en términos de roles, artefactos, actividades, y workflow	RUP
	Definir la organización del código en términos de los subsistemas a implementar y organizados en capas	RUP
	Disciplina de requerimientos	RUP
	Uso de un repositorio único de código	Open Source
	Manejar estándares de codificación	XP
	Utilización de Patrones de Diseño y Desarrollo	RUP, XP, MSF
	Codificación por capas	XP
	Refactorización	XP

Conclusiones Generales

Como se observo a largo de este trabajo de investigación y con base en la información analizada en los capítulos que lo componen, se tiene:

- El uso y elección de la metodología más adecuada para el proceso de producción de software es fundamental para su éxito (Apartado 1.3, capítulo 1).

- Las metodologías Rational Unified Process (RUP), Extreme Programming (XP) y Microsoft Solution Framework (MSF), contienen características que están orientadas a escenarios particulares. Algunas de estas características son semejantes y se pueden complementar, y así tener un proceso más completo del ciclo de vida de un software (Apartado 2.3, 2.4, 2.5 y 2.7).

- La complementación de las diferentes prácticas de las metodologías RUP, XP y MSF, proporcionan los elementos necesarios para controlar las más importantes variables bajo las cuales fueron analizadas. Estas variables son: La definición de requerimientos, la iteración con el cliente, los tiempos de codificación, el número de errores detectados en el producto entregado, el tiempo consumido y el esfuerzo planeado, la planificación del proyecto, la capacitación de recursos humanos y la administración de cambios (Apartado 3.7 Evaluación de la propuesta, capítulo 3).

- La planeación es la actividad principal del proceso de producción de software. En esta se definen las tareas y el orden de ejecución en las diferentes fases del proceso. La planeación no es estática y dependiendo de las circunstancias del proyecto, se deben ajustar las tareas necesarias para mitigar las posibles desviaciones (Apartado 3.1 Planeación, capítulo 3).

Con base en el objetivo planteado para este trabajo de investigación, el análisis comparativo de las metodologías RUP, XP y MSF en el capítulo dos y la complementación de sus practicas en el capitulo tres, se cumple el objetivo de presentar una guía metodológica para la mejora del proceso de producción de software. La propuesta de mejora esta sustentada en el capítulo tres, en el cual se mostró que a través de la complementación de dichas metodologías se puede alcanzar un mejor control sobre los diferentes puntos de riesgo del proceso. Por lo anterior se puede deducir que la hipótesis planteada para este trabajo de investigación fue comprobada con éxito.

Se espera que con esta guía, las empresas y personas dedicadas a la producción de software, cuenten con una herramienta actualizada y efectiva que les permita llevar con mayor control el proceso de producción de un software.

A diferencia de la mayoría de las investigaciones, el principal problema enfrentado durante el desarrollo del presente trabajo, fue la enorme cantidad de información existente sobre el tema, lo que obstaculizó el proceso y en consecuencia requirió de un mayor esfuerzo para el análisis y diseño de la propuesta.

Como parte fundamental del proceso de producción de software (explicado en el apartado 3.1 del capítulo 3), la planeación es la base sobre la cual se sustentan todas y cada una de las tareas que componen dicho proceso.

La evaluación de la propuesta metodológica permite observar que la complementación de las metodologías más predominantes en la actualidad, proporcionan los elementos necesarios para la mejora del proceso de producción de software. Y en consecuencia, contemplan en mayor medida los aspectos fundamentales de dicho proceso.

Los escenarios analizados para la industria de la producción de software en México, y desarrollados en el apartado 3.8 del capítulo 3, arrojan la siguiente conclusión:

- Variables Principales.
  - *La adopción de nuevas tecnologías*. La evolución continúa de la tecnología informática para adaptarse a los constantes cambios en los procesos de producción de un software, y por consecuencia, la importancia que toma la adopción de esta nueva tecnología como parte fundamental para el éxito del proceso.
  - *Los programas gubernamentales en apoyo a la industria del software.* Los apoyos gubernamentales como una parte fundamental para el desarrollo de la industria del software en México. La importancia de implementar estrategias (por parte del gobierno mexicano) que fomenten la industria y propicien un bienestar económico en el sector.

- Variables Secundarias.
  - El uso de tres de las principales metodologías en la actualidad: *Rational Unified Process*

(RUP), Extreme Programming (XP) y Microsoft Solution Framework (MSF), complementadas con otras metodologías: Scrum, Crystal Family, Open Source y Feature Driven Development con valiosas practicas.

- Escenarios
  - *Te quiero para siempre.* Cubre la relación más optimista entre las variables primarias. El gobierno de México aplica programas para el fortalecimiento de la industria del software, mientras las empresas mexicanas crean programas de capacitación para mantenerse al día en las tendencias tecnológicas.
  - *Última llamada.* Es la ultima llamada para lograr una unión de fuerzas entre el sector publico y privado. Existen continuidad en las políticas de apoyo al sector de la industria del software, sin embargo las empresas no asimilan el apoyo para mejorar y actualizar sus procesos.
  - *Sin Control.* Es el escenario más pesimista. La política mexicana rompe con la continuidad en apoyo a la industria del software y a las empresas no les interesa el desarrollo, prefieren mantener una postura mediocre. Se mantienen en un estado pasivo, perdiendo los nichos de mercado internacionales.
  - *Frágiles.* Es el escenario más probable para México. La falta de continuidad en los programa gubernamentales en apoyo a la industria del software, propician que las empresas se rasquen con sus propias uñas y se encarguen de mantenerse al día en las tendencias tecnológicas como única oportunidad de competencia hacia el exterior.

Probabilidad de ocurrencia de escenarios por año:

Escenario/Año	2010	2015	2020
Te quiero para siempre	**40%**	**25%**	15%
Última llamada	20%	20%	15%
Sin Control	5%	10%	10%
Frágiles	**35%**	**45%**	**60%**

Por ultimo, a nivel profesional, es muy gratificante observar la importancia y responsabilidad que tiene el licenciado en Matemáticas Aplicadas y Computación dentro de la industria del software, ya que gracias al perfil formado durante los nueve semestres de esta carrera, cuenta con los

elementos necesarios para el análisis, investigación, solución y mejoras de problemas orientados a los sistemas computacionales.

AMITI	Asociación mexicana de la industria de tecnologías de información.
ANUIES	Asociación Nacional de Universidades e Instituciones de Educación Superior.
ANIEI	Asociación Nacional de Instituciones de Educación en Informática.
ERP	Enterprise resource planning. (sistemas de planeación de recursos)
INEGI	Instituto Nacional de estadística, geografía e informática.
PDA	Personal Digital Assistant. (Ayudante personal digital)
PIB	Producto Interno Bruto.
TI	Tecnología Informática.
RUP	Racional Unified Process.
UML	Unified Modeling Lenguage.
FDD	Feature Driven Development.

Bibliografía

Abrahamsson, Pekka, Salo, Outi, Ronkainen, Jussi & Warsta, Juhani. Agile software development methods. Review and analysis. Espoo 2002. VTT Publications 478. 107 p.

Ackoff, Russell L. “Planificación de la empresa del futuro”. Editorial LIMUSA, 2004.

Ackoff, Russell L. “El arte de resolver problemas”. Editorial Limusa México, 1981.

Deepak_Alur, John_Crupi, Dan_Malks. Core J2EE™ Patterns: Best Practices and Design Strategies. Ed. Pearson Education; 1ra edición (Junio 26, 2001).

Grady Booch, James Rumbaugh, Ivar Jacobson. “The Unified Modeling Language User Guide”. Editorial Addison Wesley.

Kent Beck, Extreme Programming Explained – Embrace Change. Editorial Addison Wesley. 1999.

Kent Beck, Martin Fowler. “Planning Extreme Programming”. Editorial Addison Wesley. Octubre 2000.

Marlys Keeton. “Microsoft Solutions Framework (MSF): A Pocket Guide”. Van Haren Publishing. Abril 1, 2005.

Martin Fowler, Kendall Scott. “UML Distilled Second Edition A Brief Guide to the Standard Object Modeling Language”. Segunda Edition Agosto 18, 1999. Editorial Addison Wesley

Richard Hightower, Nicholas Lesiecki. “Java Tools for Extreme Programming-Mastering Open Source Tools Including Ant, JUnit, and Cactus”. Wiley Computer publishing John Wiley & Sons, Inc.

Per Kroll, Philippe Kruchten. “The Rational Unified Process Made Easy: A Practitioner's Guide to Rational Unified Process”. Editorial Addison Wesley. Abril 2003.

Pete McBreen. “Questioning Extreme Programming”. Editorial Addison Wesley. Julio 19, 2002.

Philippe Kruchten. “The Rational Unified Process: An Introduction (Third Edition)”. Editorial Addison Wesley. Diciembre 19, 2003.

Direcciones electrónicas

http://www.amiti.com.mx
http://www.inegi.com.mx
http://www.extremeprogramming.org
http://www.opensource.org/
http://www.microsoft.com/msf