

Catálogo de Patrones de ciclo de vida en desarrollo de Software aplicables en la Industria

Colombiana

Alvarez, J.

Director: Cobo, L.

Maestría en Gerencia de Sistemas de Información y Proyectos Tecnológicos

Universidad EAN

Hoja de aprobación

Evaluadores:

Carmen Elizabeth Chaparro

Alix Erica Rojas

Aprobado: SI NO

Calificación: _____

Resumen

Este trabajo de investigación recopila el conocimiento de múltiples y fuentes y plantea unos patrones de ciclo de vida de desarrollo de software (SDLC¹) aplicables en cualquier industria, basándose en las experiencias recopiladas por teorías, procesos y las aplicaciones de estos, enmarcándolas en el uso que puede tener para la industria del software en Colombia. Contiene una descripción de las variables a analizar para determinar el patrón a usar, así como los riesgos y ventajas de usar cada patrón. En algunos patrones se analizará el uso de fábricas de software en su contexto actual, con las implicaciones de trabajar desarrollos en gran escala con diferentes grupos que pueden estar o no en zonas geográficas diferentes.

Palabras clave: SDLC, Patrones, Desarrollo de software, Fábrica de software, Metodologías de desarrollo de software, Procesos de desarrollo de software, Metodologías ágiles, Industria del software en Colombia.

¹ Por su sigla en inglés Software Development Lifecycle

Tabla de contenido

Catálogo de Patrones de ciclo de vida en desarrollo de Software aplicables en la Industria Colombiana.....		10
Introducción		10
Justificación		13
Objetivos General y Específicos.....		15
General.....		15
Específicos		15
Marco teórico		16
Diseño metodológico		21
Variables para determinar los patrones de Ciclo de Vida.....		21
Selección de variables.....		22
Valores posibles de las variables.....		28
Riesgos aplicables en los procesos de SDLC		29
Riesgos genéricos aplicables.....		37
Modelo de implementación.....		40
Implicaciones del ciclo de desarrollo de software		40
Modelos de Ciclo de desarrollo de software.....		41
Patrones de Ciclo de Vida.....		54
Modelos de Patrones		54

Patrones de ciclo de vida.....	57
Árbol de decisión sobre los patrones de SDLC	98
Aplicabilidad en el entorno colombiano	100
Entorno del medio colombiano	100
SDLC aplicados en el mercado colombiano	104
Metodologías aplicables.....	106
Conclusiones	113
Referencias.....	116

Tabla de Figuras

Figura 1 Actividades preponderantes de las empresas de software en Colombia. Tomado de (Fedesoft & Ministerio de Tecnologías de la Información y las comunicaciones de Colombia, 2015, p. 14).....	11
Figura 2 Modelo SDLC en Cascada, tomado de Royce (1998, p.7).....	18
Figura 3 Taxonomía de Riesgos de Outsourcing tomado de Gasca-Hurtado & Manrique (2013, p. 46).....	32
Figura 4 Tomado de Royce, W. (1970, p. 329).....	42
Figura 5 Ubicación de estrategias de ejecución por tipo de proyecto Modelo V. Tomado de (Bundesrepublik Deutschland, 2008, pp. 1–17).....	44
Figura 6 Modelo V completo tomado de (Bundesrepublik Deutschland, 2008, pp. 3–74).....	45
Figura 7 Modelo en Espiral tomado del artículo de Boehm (1988, p. 64).....	46
Figura 8 Modelo en Espiral actualizado tomado de The incremental commitment Spiral Model (B. Boehm, Lane, Koolmanojwong, & Turner, 2014).....	47
Figura 9 Composición del RUP. Tomado de (Kroll & Kruchten, 2003).....	49
Figura 10 Tomado de www.mountangoatsoftware.com/scrum	52
Figura 11 Organización en un proceso SCRUM. Tomado de (Satpathy, 2016).....	53
Figura 12 Flujo patrón Desarrollo de sistemas en tiempo real basado en modelos. Fuente (Technische Universität München, 2002).....	59
Figura 13 Flujo propuesto patrón Desarrollo de sistemas en tiempo real basado en modelos. Fuente (Technische Universität München, 2002).....	60
Figura 14 Modelo para la fase de implementar el objetivo embebido final, tomado de (Technische Universität München, 2002).....	61

Figura 15 Proceso del patrón de proceso Conjunto de pruebas de Bootstrapping tomado de (Bergner & Rausch, 2002)	65
Figura 16 Método de Implementación Progresiva Tomado de (Soares & Borba, 2002).....	69
Figura 17 Proceso incremental, Tomado de (IBM Corporation, 2005)	73
Figura 18 Patrón evolutivo, tomado de (IBM Corporation, 2005)	75
Figura 19 Modelo planteado por el Departamento de Justicia (2003).....	75
Figura 20 Modelo propuesto para el patrón Entrega Incremental, tomado de (IBM Corporation, 2005)	77
Figura 21 Estructura de Gran diseño, tomado de (IBM Corporation, 2005)	79
Figura 22 Estructura de iteraciones de ME. Tomado de (The US Department of Justice, 2003). 81	
Figura 23 Ciclo de desarrollo con Fábrica de pruebas. Fuente propia.....	84
Figura 24 Proceso de desarrollo Zipper. Fuente propia	86
Figura 25 Ciclo de vida propuesto en Colpensiones. Tomado de (Colpensiones & Tecnocom, 2015)	88
Figura 26 Fases de un ciclo de vida, tomado de (Walker Royce, 1998).....	89
Figura 27 División de trabajo en componentes. Fuente propia	90
Figura 28 Proceso mostrando el proceso con Nearshore y offshore. Tomado de (Erni, 2015, p. 24)	91
Figura 29 Ciclo de vida FTS. Tomado de (Carmel et al., 2010).....	93
Figura 30 Ciclo de desarrollo con distribución global. Tomado de (Gupta, Sloan School of Management., & Seshasai, 2004)	94
Figura 31 Árbol de decisión en la elección de un patrón de SDLC. Tomado de: Fuente propia..	99
Figura 32 Ciclo de ATDD Tomado de (Hendrickson, 2008)	108

Figura 33 Escenarios de BDD. Tomado de (Duarte & Fernandes, n.d.).....	109
Figura 34 Proceso de transformación de MDSD. Tomado de (Stahl et al., 2006).....	110
Figura 35 Ciclo de levantamiento Volère. Tomado de (Robertson & Robertson, 2013)	112

Tabla de tablas

Tabla 1 Comparación de variables de selección de Patrón de SD Fuente propia.	27
Tabla 2 Comparación de proyectos Scrum y tradicionales. Tomado de (Satpathy, 2016, p. 19)..	54
Tabla 3 Patrones aplicables a los diferentes Modelos de SDLC. Tomado de:Fuente Propia	95
Tabla 4 Relación de patrones contra variables. Tomado de: Fuente propia.....	97
Tabla 5 Comparación proyectos empresas grandes y PyMEs Bogotá. Tomado de (González et al., 2016, p. 132)	102
Tabla 6 Factores de éxito en los proyectos empresas de Bogotá. Tomado de (González et al., 2016, p. 132)	103

Catálogo de Patrones de ciclo de vida en desarrollo de Software aplicables en la Industria Colombiana

Introducción

Los proyectos de desarrollo de software tienen tasas de éxito bajas en comparación a los pertenecientes a otras ingenierías y aplicaciones de la ciencia como la Arquitectura, debido a que la Ingeniería de Software es nueva, si consideramos que la idea de crear software nació a partir de un ensayo de Alan Turing en 1935. Según The Standish Group(1995), la probabilidad de tener un proyecto totalmente exitoso es inferior al 39% dependiendo de factores como el tamaño del proyecto, implicando que hay errores en la administración de los mismos que dificultan cumplir con los tiempos y presupuesto estimados; sus investigaciones determinan que los problemas más comunes están relacionados con el levantamiento de requerimientos o alcance del proyecto con 48% de causas asociadas, mientras que el 28% de los problemas están relacionados con la gestión y 24% se deben a otras múltiples causas. Pero hay otras investigaciones como las llevadas a cabo por el Grupo EQUITY, donde demuestran basándose en los cálculos de trabajos estadísticos de Boehm y DeMarco y sus propias mediciones que las cifras de Standish están erradas(Eveleens & Verhoef, 2010), la cantidad de proyectos de software totalmente efectivos es menor a 16%. El manejo de los requerimientos, es uno de los principales problemas objetivo que han abordado Boehm (1988) al plantear su modelo en espiral, y que ha sido afrontado en los procesos ágiles como Scrum donde se mantiene una lista de deseables o Backlog (Schwaber & Sutherland, 2011) sobre los cuales se hace mayor profundidad al momento de construir el software, respondiendo al planteamiento de Pressman (2010, p. 66), "...En muchas situaciones, usted no va a poder definir los requerimientos plenamente antes de que el proyecto inicie. Usted

debe ser suficientemente ágil para responder a un ambiente de negocios fluido.”. Adicionalmente Bieg(2014), lo identifica como fundamental para el éxito de cualquier proyecto.

Por lo indicado anteriormente, se puede concluir que existen unas inconsistencias en la administración de proyectos de software, que están generando pérdida de recursos, con cifras inaceptables para otras industrias, por ejemplo, pensemos lo que significaría para el transporte que sólo entre un 16 y 39 por ciento de los proyectos de infraestructura fueran exitosos, con esta investigación se pretende dar una base para reducir la incertidumbre en el momento de planear un proyecto de software, al proponer una guía a través de los patrones que permita establecer actividades y fases necesarias para aumentar la probabilidad de éxito.

En el ámbito colombiano, la industria del SSA (Software y servicios asociados) tiene cada vez más relevancia, pasó de comercializar 2.6 billones en 2010 a 7.5 billones en 2014 (Tecnosfera, 2015), ha tenido crecimiento del 24% desde el 2012 al 2014, de 2012 a 2013 creció 7% y del 2013 al 2014 un 17% adicional (Mintic & Fedesoft, 2015, p. 111), y la segunda actividad con mayor cantidad de empresas participantes es el desarrollo de software (Mintic & Fedesoft, 2015, pp. 11–12), como se aprecia en la figura siguiente:

Productos y Servicio	Cantidad	Participación
Manejo de centros de datos (data center)	851	25%
Desarrollo / fábrica de software	772	23%
Mesas de ayuda (Otras)	477	14%
Testing de software	330	10%
Infraestructura como servicio	300	9%
Consultoría e implementación	143	4%
Mantenimiento o soporte de aplicaciones	143	4%
Software como servicio	116	3%
Otro	115	3%
Plataformas tecnológicas como servicio	90	3%
Cloud computing	27	1%
Gerencia	6	0%
Total General	3370	

Figura 1 Actividades preponderantes de las empresas de software en Colombia. Tomado de (Fedesoft & Ministerio de Tecnologías de la Información y las comunicaciones de Colombia, 2015, p. 14)

Pero el crecimiento de la industria del software en Colombia se ve limitado por factores como: calidad de productos, especialización del recurso humano, niveles de inversión en capacidades de innovación, certificación de empresas, regulaciones de exportación y ventas (Martínez, Arango, & Robledo, 2015, p. 97; Palomino, 2011, p. 31) y son pocas las empresas que cuentan con el capital suficiente para abordar un proceso de madurez como CMMI-DEV, ya que el 99.1% de estas no superan las ventas anuales por más de 1000 millones de pesos y 60% no supera los 294 millones, aunque el producir software es la actividad con el mayor margen de ganancia bruta pero con la menor rotación de cartera (Fedesoft & Ministerio de Tecnologías de la Información y las comunicaciones de Colombia, 2015, pp. 21–28), estas características implican que una empresa con mejor rotación de cartera debido a la capacidad de implementar a tiempo los proyectos de desarrollo, tiene mayor factibilidad de sobrevivir, es allí donde el uso de un patrón puede mejorar la probabilidad de tener el proyecto correctamente formulado. Es claro que la rotación de cartera no sólo depende de la capacidad para generar los entregables en los tiempos estipulados en la planeación del proyecto, también está ligado a la forma de pago del contrato firmado y otras características más apropiadas para un estudio de contratación de software.

Otro de los problemas que aborda este estudio es la necesidad cada vez mayor de trabajar grandes proyectos con fábricas de software como describe Greenfield(2003), lo cual implica complicaciones de ensamble de código y trabajo en procesos como los estudiados por Nomura, et al.(2007), y Azanza, Díaz & Trujillo(2008), pero mejoras al costo mediante el uso de cadenas de producción. Adicionalmente el surgimiento de países como India, Irlanda e Israel (Unctad & Ki-moon, 2012) con capacidad para desarrollo de software a mejores precios (Balduino, 2005) generó la necesidad de tener equipos de desarrollo distribuidos geográficamente, esto implica

nuevos retos que impactan la calidad del software, es por esto que existen iniciativas (Carmel, Espinosa, & Dubinsky, 2010) para integrar esfuerzos de desarrollo y aprovechar la diversidad horaria.

Justificación

La presente investigación busca establecer las posibles formas de configurar las etapas de un proyecto de software, para que este sea más efectivo, dependiendo de las necesidades específicas a implementar, permitiendo reutilizar el conocimiento de diferentes investigaciones, metodologías y planteamientos, al tener definido el ámbito de aplicación de cada modelo de ciclo de vida y sus implicaciones.

El desarrollo de software es una de las industrias con mayor posibilidad de apoyar el progreso en países en vías de desarrollo, como lo manifiesta la UNCTAD (Unctad & Ki-moon, 2012) en su Reporte económico: “La habilidad de un país para adoptar, adaptar y desarrollar soluciones tecnológicas apropiadas y aplicaciones depende de la fuerza de sus capacidades domésticas. Esto aplica en particular al área del software, debido a que ésta implica una tecnología de propósito general con relevancia a un amplio rango de campos de desarrollo sociales y económicos”.

Para Colombia, la industria del software hace parte de los seis sectores de la economía que estaban planeados en 2009 como los que iban a tener mayor fortalecimiento, con una visión clara, tener exportaciones por 12.500 millones de dólares en esos seis sectores. Los sectores que fortalecerían el crecimiento de la industria colombiana según este informe eran: Autopartes,

Energía eléctrica, Industria gráfica, Industria Textil, Turismo de salud, Cosméticos, Servicios tercerizados a distancia (BPO&O) y Software y servicios TI. Pero si vemos las cifras actuales de exportaciones los únicos sectores de los mencionados que exportaron en un volumen relevante fueron: Energía eléctrica con 48 millones de dólares, Industria Textil con 21 millones de dólares y autopartes con 10 millones de dólares, lo cual implica que el desarrollo de software sigue sin dar los resultados esperados para la economía colombiana y por lo tanto la investigación y trabajo relacionado con mejoras a este campo son beneficiosos para los propósitos de crecimiento en este sector.

En Colombia el uso de metodologías sigue los estándares internacionales, de acuerdo al estudio realizado por García (2014, p. 6), el 82% de las empresas encuestadas en la base de Fedesoft tiene una metodología particular para desarrollar cada proyecto aparte de Cascada y al menos el 44% de estas usa RUP mezclada con metodologías ágiles como Scrum y XP, este estudio es completamente compatible con estas metodologías y adiciona la capacidad de prever una estructura de proyecto con mejor capacidad de éxito, que implementar un proyecto con estructura genérica.

Como implicaciones prácticas del presente estudio, podemos decir que el tener una guía de referencia para decidir el patrón adecuado de SDLC (por su significado en inglés, Software Development Life Cycle) en la fase de planeación es un buen inicio para cualquier proyecto, los patrones se van a basar en historial de buenas prácticas como las explicadas por Pressman (2010), Humphrey (2000) y Boyde (2012), teorías demostradas por múltiples proyectos (Ambler, 1998) y unas caracterizaciones que permitirán elegir el modelo adecuado. En términos de

Ingeniería de Software, se da un ahorro en tiempo y mitigación del riesgo si se basa en un modelo previamente experimentado y comprobado por otros equipos de proyecto en el mundo.

La utilidad metodológica de esta investigación es servir de paso inicial en la generación de un plan de trabajo en un proyecto de software, y pretende servir como ventana para la profundización sobre teorías o conocimientos alrededor del patrón seleccionado por el gerente de proyecto, este trabajo no suple el entendimiento que puede adquirirse de la investigación específica de un modelo de SDLC, ya que es una guía de referencia sobre las mejores formas de proceso a implementar. Por ejemplo, si un gerente de proyecto determina por medio de este estudio que la forma más apropiada de implementar un patrón es usando RUP, el libro de Jacobson, Booch y Rumbaugh (2005) lo pueden guiar en profundidad.

Objetivos General y Específicos

General

Construir una guía rápida de elección de patrones de ciclo de vida de desarrollo de software, de acuerdo a las características específicas del ambiente en el cual se van a desarrollar los proyectos, tomando en cuenta las teorías y la aplicabilidad que tienen en proyectos reales del medio colombiano.

Específicos

1. Recopilar el conocimiento de múltiples teorías enunciadas y utilizadas en múltiples proyectos de desarrollo de software, acerca de la forma en la cual se debe planear un

- proyecto de este tipo, en un proceso denominado Ciclo de vida del desarrollo de software (SDLC).
2. Seleccionar los patrones de ciclo de vida aplicables debido a sus características y utilidad a lo largo de múltiples implementaciones exitosas en el mercado.
 3. Encontrar la utilidad de los patrones en el entorno colombiano, y el modo de uso de estos para planear proyectos con menor incertidumbre.
 4. Determinar las variables del entorno del proyecto de desarrollo de software que puedan ser relevantes en el momento de establecer cuál es el patrón de SDLC adecuado para un proyecto de software.
 5. Determinar los riesgos críticos en el proceso de desarrollo de software que puedan generarse al usar un patrón específico de SDLC.
 6. Establecer metodologías complementarias que ayudan en el uso de algunos patrones, por ejemplo, el uso de metodologías de levantamiento de requerimientos adecuadas para algunos SDLC.

Marco teórico

A continuación, se encuentran las principales ideas de esta monografía y cómo se apoyan en la literatura existente.

La primera idea que se trabajará es el manejo de patrones. En ciencias aplicadas como la arquitectura o la ingeniería de software, los patrones representan las mejores prácticas, soluciones probadas, lecciones aprendidas e información depurada y clasificada que ayuda a generar conocimiento con facilidad; los patrones, al ser expresados en un lenguaje claro, con su forma de implementación, análisis de riesgo y formas de uso bien limitadas, ayudan a las

personas que requieran utilizarlos, a resolver problemas para los cuales tendrían que implementar soluciones probando y fallando en repetidas ocasiones, hasta encontrar una solución similar (Alexander, C., 1979, citado por Tesanovic 2001). Los patrones son pues una forma de acumular conocimiento con el objetivo de ayudar a evolucionar la Ingeniería de Software, y como lo afirma Brooks en su famoso libro “The Mythical Month-man” (Brooks, 1995) “No hay balas de plata”, ningún patrón es la solución mágica para todos los problemas, es por esto que es necesario tener bien claros los prerequisites de implementación y las consecuencias de su uso.

Para determinar el patrón de ciclo de desarrollo de software más adecuado para cada situación, es necesario encontrar los principales factores que permitirán tomar la decisión; dichos factores se tomarán a partir de diferentes artículos como el de Öztürk (2013), y se relacionará con las secciones específicas de las obras de Mishra & Dubey(2013, p. 68), el estudio del Centro de servicios de cuidado y ayuda médica de Estados Unidos (CMS) (2008, pp. 1–10), el artículo de Hanafiah & Kasirun (2007, p. 212), la investigación de Guntamukkala, Wen & Tarn (2006), el documento del Departamento de Justicia de los Estados Unidos (The US Department of Justice, 2003) donde hacen un análisis y proponen patrones de ciclo de vida basándose en parámetros, el artículo de Vaghela (Vaghela, 2015), la investigación de Maheshwari & Jain (Maheshwari, Jain, Maheshwari, & Jain, 2012), el artículo de Seema & Malhotra(Seema & Malhotra, 2015), el artículo de Rastogi (Rastogi, 2015) y finalmente los criterios de configuración de RUP de IBM (IBM Corporation, 2005).

Una parte fundamental de esta agrupación de patrones, consiste en la definición de cuáles son las posibles opciones o formas de configurar un proyecto y sus diferentes etapas, en lo que se

denomina Ciclo de vida de desarrollo de software, SDLC (Software Development Life Cycle) por su sigla en inglés. Sus primeras definiciones surgieron en los años setenta del siglo pasado y en Ambler (1998, p. 6), se encuentra una recopilación de las principales opciones. Inicialmente una de las formas más comunes fue el modelo en cascada.

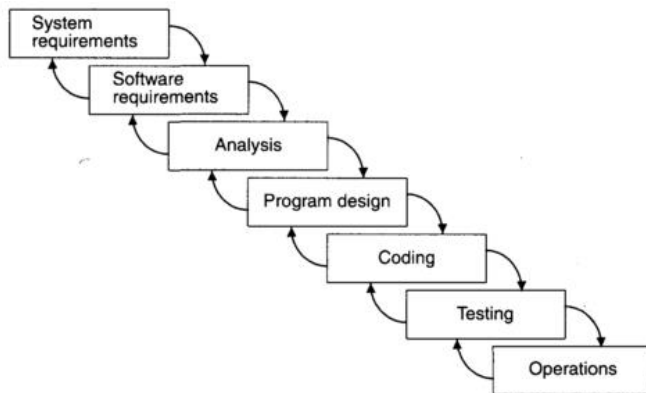


Figura 2 Modelo SDLC en Cascada, tomado de Royce (1998, p.7)

Como se puede apreciar en la figura anterior, serializa las etapas más comunes en el desarrollo de software, y este mismo SDLC siguió siendo el modelo más utilizado en diferentes tipos de desarrollo como se explica en Royce (1998, p. 6), aun cuando metodologías cíclicas como el modelo propuesto por Boehm (1988) o el proceso unificado (Jacobson, Booch, & Rumbaugh, 1999) tuvieron popularidad y difusión. En Europa el Modelo V planteado por el ejército alemán en 1992 (Bartelt et al., 2006; Bundesrepublik Deutschland, 2008) tuvo buena recepción por su planteamiento de hacer énfasis en los entregables de calidad, y hubo otros modelos que no tuvieron tanta acogida como RAD (McConnell, 1996), aunque abrieron el panorama para las metodologías ágiles. Sin embargo en la últimos quince años las metodologías ágiles (Beck et al., 2001; Pressman, 2010, p. 67) han tenido bastante acogida por dar la impresión de mitigar el impacto del mayor problema que tiene el desarrollo de software, la indefinición de requerimientos (Bieg, 2014; The Standish Group International, 2013).

En la determinación de riesgos en los diferentes patrones de SDLC es importante revisar el artículo de Miler & Gorski (Miler & Gorski, 2004), en Chaos Report (The Standish Group, 1995) se encuentran causas para los proyectos con problemas, la Taxonomía de riesgos del trabajo de Gasca-Hurtado & Manrique (2013), y en las obras de Tao(Tao, 2006) , Pressman(2010, Chapter 28), Ambler(1998), Claro Testing (2008), Boyde (2012, Chapter 1.6) y Royce (1998, p. 228), se encuentran análisis de los riesgos de diferentes tipos de implementación de SDLC.

Hay otros escenarios de desarrollo donde la literatura de ciclos de vida no ha sido muy profunda, que es en el uso de fábricas de software para desarrollo y la coordinación de esfuerzos de forma efectiva para generar desarrollos a gran escala con los mejores resultados. Es en este punto de la investigación donde se plantearán patrones para la administración de proyectos con una o más fábricas de software, con presencia local o distribuida geográficamente, desarrollo paralelo e integración y la división más apropiada de tareas que se puede utilizar basándose en las investigaciones de Greenfield (2003), Nomura, et al. (2007) y Azanza, Díaz & Trujillo (2008). Otra parte importante es el manejo de trabajo en *Offshore* con proveedores que se analiza en los trabajos de Ambler(1998, Chapter 6.2.1.3), Prikladnicki & Audy (2012) y Pries-Heje, Baskerville, & Hansen (2005). Adicionalmente se analizará la implementación de equipos de trabajo distribuidos que plantea Carmel (2010), sin descartar la inclusión de nuevas fuentes que puedan aportar a esta área del conocimiento.

También se hará una breve semblanza del mercado de software colombiano, y cómo estos patrones pueden tener relevancia para lograr los objetivos que se tienen trazados a nivel de país

para esta industria, para esto se utilizará la información encontrada en el Informe De Caracterización Del Sector De Software Y Tecnologías De La Información En Colombia (Fedesoft & Ministerio de Tecnologías de la Información y las comunicaciones de Colombia, 2015), así como los estudios de Díaz & Ospina (2014), Martínez et al. (2015), Fúquene, Castellanos, & Fonseca (2007), Palomino (2011), y finalmente es muy importante el aporte del estudio de González, Sánchez, & Velandia (2016). Adicionalmente las siguientes investigaciones muestran lo que se conoce sobre el uso de SDLC en los proyectos de desarrollo en Colombia, primero se va a relacionar la investigación de la Universidad de Los Andes alrededor del proyecto Qualdev (Casallas & Arboleda, 2004), que buscaba dar una fórmula de procesos dinámicos para desarrollos ágiles, luego las investigaciones de Arboleda (2002), Moreno, Rengifo, & Navia (2010) y Garcia (2014).

En la última parte de la investigación se explicará el uso coherente que tienen algunas metodologías o modelos incorporados al uso de patrones de SDLC, como el Model Driven Software Development, o desarrollo de software orientado por modelos (Stahl, Völter, Bettin, Haase, & Helsen, 2006), y procesos de requerimientos dinámicos como el planteado por la metodología Volère (Robertson & Robertson, 2013) o BDD Behavior Driven Development (Duarte & Fernandes, n.d.).

En esta monografía se generarán patrones de SDLC, lo cual solo fue encontrado con ese nombre en una de las fuentes (The US Department of Justice, 2003), pero sin hacer el análisis de riesgos de implementación y ventajas de uso, hace un análisis claro de la implementación, pero descarta los procesos de ciclo de vida alternativos y parece guiarse por un proceso en cascada con detalle

de requerimientos para cada etapa. Adicionalmente se usarán los patrones generados por el Departamento de Defensa de los Estados Unidos(1994), explicados en mayor detalle en el análisis de RUP de IBM (IBM Corporation, 2005) y especialmente se usará el trabajo de la Universidad de München (Technische Universität München, 2002) en el taller de Patrones de desarrollo de software. Con respecto a los patrones planteados por Ambler (1998), corresponden más a patrones propios de las fases de lo que denomina un Proceso de Software Orientado a Objetos, la generación de patrones de software para SDLC va a ser un aporte de este trabajo con el modelo de patrones estándar, que permite conocer los riesgos y ventajas de cada uno, así como su forma de ser aplicado.

Diseño metodológico

Diseño de la investigación: Cualitativa

Tipo de estudio: Investigación Analítica observacional

Unidad de estudio: Proyectos de desarrollo de software

Unidad de información: Procesos para el desarrollo de software

Unidad de Observación: Patrones de SDLC

Unidad de análisis: Usabilidad, riesgos, modelo y metodologías complementarias

Variables para determinar los patrones de Ciclo de Vida

Para poder determinar los criterios de elección de los patrones de ciclo de vida, se tomarán como base diferentes fuentes como el paper de Öztürk (2013, p. 803), en el cual se propone usar lógica difusa para determinar el SDLC apropiado, adicionalmente se puede usar otros estudios que comparan los diferentes ciclos de vida para extraer las variables relevantes,

como el de Mishra & Dubey(2013, p. 68), los criterios de división de los proyectos es necesario dar valores discretos a cada una de estas, para que, al relacionar los valores posibles de estas variables, se pueda seleccionar el patrón más adecuado.

Selección de variables

Primero es necesario analizar las variables seleccionadas por Öztürk (2013, p. 803) para hacer su taxonomía de proyectos de software, las cuales son: Estabilidad y claridad de requerimientos, Tiempo de desarrollo, Tamaño del proyecto, Experiencia del equipo de desarrollo y el usuario, Número de riesgos críticos y Complejidad.

Para Mishra & Dubey(2013, p. 68), los criterios que mencionan son usados para comparar diferentes SDLC y estos son: Especificación de requerimientos, Costo, Simplicidad, Riesgo involucrado, Experiencia, Flexibilidad al cambio, Involucramiento de los usuarios, Mantenimiento y Duración.

En la investigación de Guntamukkala, Wen & Tarn (2006, p. 273), se usaron bastantes criterios para diferenciar los proyectos de desarrollo de software:

- 1 Los requerimientos de proyecto están en constante cambio
- 2 El alcance es desconocido
- 3 Los requerimientos están bien entendidos
- 4 El alcance no está bien entendido
- 5 Los usuarios finales son muy sofisticados en términos de avances tecnológicos
- 6 El equipo de trabajo es altamente inexperto

- 7 El equipo de trabajo carece de experiencia técnica
 - 8 El equipo de trabajo es pequeño (entre 6 y 10 personas)
 - 9 La arquitectura del sistema no está bien entendida
 - 10 El producto final va a ser modificado frecuentemente en el futuro
 - 11 Los riesgos no están bien entendidos
 - 12 La tecnología que se va a usar para desarrollar el software es nueva
 - 13 Las restricciones de Tiempo/Cronograma no son bien entendidas
 - 14 Los hitos del proyecto no están claramente definidos
 - 15 Los miembros del equipo no tienen familiaridad con las tareas y herramientas
 - 16 El presupuesto es ajustado
 - 17 La infraestructura requerida para el desarrollo del software no está disponible actualmente.
- Estos criterios se pueden agrupar en: Definición de alcance y requerimientos, Experiencia del equipo de trabajo, Tamaño del equipo de trabajo, Novedad en el uso de tecnologías, Manejo de riesgos, Definición de la administración del proyecto y Flexibilidad al cambio.

Revisando el estudio del Centro de servicios de cuidado y ayuda médica de Estados Unidos (CMS) (2008, pp. 1–10), se pueden deducir las siguientes variables, que son las comparadas en cada SDLC analizado: Tamaño, Estabilidad de los requerimientos, Flexibilidad de tiempo, Conocimiento del usuario final, Experiencia del equipo de trabajo, Estabilidad del equipo de trabajo, Tipo de software desarrollado y Manejo del riesgo.

En el artículo de Hanafiah & Kasirun (2007, p. 212) se hace un análisis interesante y muy pertinente para esta monografía, ya que genera un software que mediante el análisis de variables

determina el SDLC más apropiado para el Proyecto; las variables que proponen para dividir los proyectos son: Tamaño, Duración, Cantidad de recursos, Modularización, Interacción con el usuario, requerimientos no funcionales y nivel de riesgo.

Para el Departamento de Justicia de los Estados Unidos en su página web

<http://www.justice.gov/archive/jmd/irm/lifecycle/table.htm> (The US Department of Justice, 2003) los criterios a utilizar son: Tipo de desarrollo, tamaño del proyecto, Criticidad del desarrollo, Riesgo de variabilidad de requerimientos y Complejidad del desarrollo.

En el artículo de Vaghela (Vaghela, 2015), los criterios escogidos fueron: Especificación de requerimientos, Costo, simplicidad, Riesgo involucrado, Flexibilidad al cambio, Involucramiento de los usuarios, Mantenimiento, Duración, Garantía de éxito y Satisfacción del cliente.

El artículo de Maheshwari & Jain (Maheshwari et al., 2012), usa los siguientes criterios:

Especificación de requerimientos, Entendimiento de los requerimientos, Costo, Disponibilidad de componentes reutilizables, Complejidad del sistema, Riesgos, Involucramiento del usuario, Garantía de éxito, Fases simultáneas, Tiempo de implementación, Flexibilidad, Cambios incorporados, Experiencia requerida, Control de costos y Control de recursos.

En el artículo de Seema & Malhotra (Seema & Malhotra, 2015) se encuentran los siguientes criterios: Especificación de requerimientos, Costo, Garantía de éxito, Complejidad del sistema, Fases simultáneas, Riesgos, Experiencia requerida, Cambios incorporados, Involucramiento del

usuario, Flexibilidad, Mantenimiento, Integridad y seguridad, Reusabilidad, Documentación y entrenamiento y Marco de tiempo.

En RUP de IBM (IBM Corporation, 2005) el ciclo de vida se puede personalizar usando los siguientes criterios: Conocimiento del dominio del problema, cantidad de riesgo, experiencia del equipo de desarrollo, grado de novedad en el proyecto y estabilidad del producto.

Adicionalmente a estas taxonomías de proyectos, esta investigación se centra en dos atributos adicionales, equipos distribuidos geográficamente y uso de fábricas de software.

Revisando los valores en común en estos estudios se encuentra el siguiente cuadro:

	Öztürk	Mishra & Dubey	Guntamukkal a, Wen & Tarn	CMS	Hanafiah & Kasirun	The US Department of Justice	Vaghela	Maheshwari & Jain	Seema & Malhotra	IBM	Total
Estabilidad y claridad de requerimientos	X	X	X	X			X	X	X	X	8
Tiempo de desarrollo	X				X			X	X		4
Tamaño del proyecto	X			X							2
Experiencia del equipo de desarrollo	X	X	X	X				X	X	X	7
Número de riesgos	X	X	X	X	X	X	X	X	X	X	10
Costo		X					X	X	X		4
Simplicidad del proyecto		X				X	X	X	X		5
Flexibilidad al cambio		X	X	X			X	X	X	X	7
Involucramiento de usuarios		X		X	X		X	X	X		6

Mantenimiento y Duración		X	X		X		X		X		5
Estabilidad del equipo de trabajo				X							1
Tipo de software desarrollado				X		X				X	3
Requerimientos no funcionales					X				X		2
Tamaño del equipo			X		X						2
Novedad en el uso de tecnologías			X								1
Criticidad del desarrollo						X					1
Garantía de éxito							X		X		2
Satisfacción del cliente							X				1
Control de costos y recursos								X			1
Fases simultáneas								X	X		2

Tabla 1 Comparación de variables de selección de Patrón de SD Fuente propia.

Las variables que se tomarán para este estudio son las que tienen más estudios en común, que son:

- Número de riesgos, todos los estudios coincidieron en esta variable.
- Estabilidad y claridad de requerimientos, ocho de los estudios lo mencionan como importante y en otros estudios específicos de proyectos como el de Bieg donde se le da toda la importancia para lograr el éxito en estos, de hecho se afirma que el 47% de los proyectos fallan debido a manejo de requerimientos defectuoso (2014, p. 5).
- La experiencia del equipo de desarrollo también es un factor común en la decisión del SDLC a utilizar, siete de los estudios lo señalan.
- La flexibilidad al cambio en el proyecto es importante para 7 de los estudios, lo cual es normal debido a que esta flexibilidad permite cambiar el alcance y costo del proyecto.
- Involucramiento de usuarios es importante en todos los procesos ágiles, por lo cual es un parámetro que coincide con tres de los estudios y es normal que sea importante tenerlo en cuenta.

Adicionalmente a las taxonomías de proyectos de estos estudios, esta investigación se centra en dos atributos adicionales, distribución geográfica del equipo y uso de fábricas de software.

Valores posibles de las variables

Para hacer más sencilla la separación de proyectos se darán valores nominales a las variables, que los usuarios de los patrones podrán seleccionar.

- Número de riesgos: Bajo o medio y Alto.
- Estabilidad y claridad de requerimientos: Claramente definidos y estables, y Vagamente definidos o inestables.

- Experiencia del equipo de desarrollo: Baja experiencia y Alta experiencia
- La flexibilidad al cambio en el proyecto: Baja flexibilidad y Alta flexibilidad
- Involucramiento de usuarios: Bajo o Normal Involucramiento y Alto involucramiento.
- Distribución geográfica del equipo: Equipo local, y Equipo distribuido geográficamente
- Uso de fábricas de software: Desarrollo con una fábrica de software, Desarrollo con dos fábricas de software y Desarrollo con más de dos fábricas de software.

En el caso que la variable no sea determinante en el patrón se escribirá NA, indicando que no aplica su análisis para el caso estudiado, o que cualquier valor de la variable es válido para el patrón.

Riesgos aplicables en los procesos de SDLC

Los riesgos más comunes, aplicables en los procesos de ciclo de vida de desarrollo de software son los siguientes de acuerdo al trabajo de Miler & Gorski (2004, p. 8):

- La documentación del modelo de negocio no representa bien al negocio debido a un entendimiento pobre del dominio de negocio y falta de modelado.
- La falta de reuniones de entendimiento del negocio produce identificación incompleta del proceso de negocio
- Un glosario de términos de negocio ambiguo produce un modelo de negocio inconsistente.
- Excesiva presión en el cronograma reduce la productividad.
- La gerencia del Proyecto hace más énfasis en desarrollos heroicos que en reportes de estado precisos, lo cual disminuye su habilidad para detectar y corregir problemas.

- Las herramientas de trabajo no funcionan como se espera; los desarrolladores necesitan tiempo para crear soluciones alternativas o cambiarse a una nueva herramienta.
- Los usuarios finales no se comprometen con el proyecto y consecuentemente no proveen el apoyo necesario.
- El contratista no se compromete con el Proyecto y consecuentemente no provee el desempeño requerido.
- Las áreas del producto vagamente especificadas consumen más tiempo del esperado.
- Los módulos tendientes a fallar requieren más pruebas, diseño y trabajo de implementación del esperado.
- Desarrollo de una interfaz de usuario equivocada resulta en rediseño e implementación adicional.
- Los conflictos entre miembros del equipo resultan en comunicación pobre, diseños de baja calidad, errores de interfaz y retrabajo extra.
- Problemas de gerencia de Proyecto generan cronogramas y planeación ineficientes.
- Funcionalidad necesaria no puede ser implementada usando el código seleccionado o las librerías de clases; los desarrolladores deben cambiar a nuevas librerías o construir por sí mismos la funcionalidad necesaria.
- Un manejo de riesgos poco incentivado falla en detectar los mayores riesgos del Proyecto.

El Chaos Report (The Standish Group, 1995, p. 9) trae dos listados de factores de riesgo:

1. Proyectos con problemas de tiempo alcance o recursos
 - 1.1. Falta de retroalimentación del usuario final

- 1.2. Requerimientos y especificaciones incompletas
 - 1.3. Requerimientos y especificaciones cambiantes
 - 1.4. Falta de apoyo ejecutivo
 - 1.5. Incompetencia tecnológica
 - 1.6. Falta de recursos
 - 1.7. Expectativas irreales
 - 1.8. Objetivos poco claros
 - 1.9. Marcos de tiempo irreales
 - 1.10. Nueva tecnología
2. Proyectos cancelados
 - 2.1. Requerimientos incompletos
 - 2.2. Falta de involucramiento del usuario final
 - 2.3. Falta de recursos
 - 2.4. Expectativas irreales
 - 2.5. Falta de apoyo ejecutivo
 - 2.6. Requerimientos y especificaciones cambiantes
 - 2.7. Falta de planeación
 - 2.8. El producto no se necesitaba
 - 2.9. Falta de Gerencia de IT
 - 2.10. Desconocimiento tecnológico

En el trabajo de Gasca-Hurtado & Manrique(2013) encontramos la siguiente taxonomía de riesgos para proyectos con *Outsourcing*:

I. Acuerdos y Contratación Proveedor	II. Entorno de Desarrollo	III. Ingeniería de Adquisición
<p>A. Ejecución de acuerdos</p> <ol style="list-style-type: none"> 1. Rendimiento 2. Gestión de Revisiones 3. Identificación de inconvenientes 4. Relación con el proveedor 5. Monitorizaciones de riesgos <p>B. Monitorización de procesos</p> <ol style="list-style-type: none"> 1. Procesos críticos 2. Monitorizaciones de procesos 3. Análisis de resultados <p>C. Aceptación del producto</p> <ol style="list-style-type: none"> 1. Resultados de validación 2. Resultados de verificación 3. Requisitos contractuales 4. Criterios de aceptación <p>D. Gestión de factura del proveedor</p> <ol style="list-style-type: none"> 1. Revisión de facturas 2. Resolución de errores o inconvenientes 3. Aprobación de la factura <p>E. Procedimiento</p> <ol style="list-style-type: none"> 1. Identificación de proveedores 2. Establecimiento del pliego de condiciones 3. Revisión del pliego de condiciones 4. Distribución y mantenimiento del pliego de condiciones <p>F. Selección de proveedores</p> <ol style="list-style-type: none"> 1. Evaluación de propuestas 2. Establecimiento de planes de negociación 3. Selección de proveedores <p>G. Establecimiento del contrato</p> <ol style="list-style-type: none"> 1. Acuerdo 2. Contrato 	<p>A. Requisitos del cliente</p> <ol style="list-style-type: none"> 1. Elicitación 2. Priorización <p>B. Requisitos contractuales</p> <ol style="list-style-type: none"> 1. Establecimiento 2. Asignación <p>C. Análisis y validación</p> <ol style="list-style-type: none"> 1. Establecimiento de escenarios 2. Análisis y comprobación 3. Validación <p>D. Evaluación de la solución técnica</p> <ol style="list-style-type: none"> 1. Selección 2. Análisis 3. Ejecución <p>E. Gestión de interfaz</p> <ol style="list-style-type: none"> 1. Selección 2. Gestión 	<p>A. Procedimiento de Validación</p> <ol style="list-style-type: none"> 1. Selección de productos 2. Entorno de validación 3. Procedimiento y criterios de validación <p>B. Desarrollo de validación</p> <ol style="list-style-type: none"> 1. Ejecución 2. Análisis <p>C. Procedimiento de verificación</p> <ol style="list-style-type: none"> 1. Selección de productos 2. Entorno de verificación 3. Procedimiento y criterios de validación <p>D. Desarrollo de revisiones</p> <ol style="list-style-type: none"> 1. Preparación 2. Ejecución 3. Análisis de datos <p>E. Productos de trabajo</p> <ol style="list-style-type: none"> 1. Ejecución verificación 2. Análisis de resultados de verificación

Figura 3 Taxonomía de Riesgos de Outsourcing tomado de Gasca-Hurtado & Manrique (2013, p. 46)

Como se puede apreciar, los riesgos se concentran alrededor de tres puntos críticos dentro del proceso de desarrollo: Acuerdos y contratación del proveedor, Entorno de desarrollo e Ingeniería de outsourcing, que se asemejan a tres momentos de los proyectos que son la preparación o concepción del proyecto, implementación y postmortem.

En el artículo de Tao (Tao, 2006), se pueden encontrar implícitos los siguientes riesgos:

1. No satisfacer las necesidades del cliente
2. Sistemas innecesariamente complejos
3. Problemas de personal (renuncias, vacaciones, terminación de proyectos o reducciones)
4. Problemas arquitectónicos (Baja flexibilidad, dificultades de mantenimiento o escalabilidad)

Para Pressman (2010, Chapter 28), existen unas categorías de riesgos:

1. Riesgos del proyecto
2. Riesgos técnicos
3. Riesgos del negocio

Para la identificación de riesgos propone unas subcategorías:

1. Tamaño del producto
2. Impacto en el negocio
3. Características de los interesados
4. Definición del proceso
5. Ambiente de desarrollo
6. Tecnología a ser construida
7. Tamaño del equipo y experiencia

De acuerdo al Ejército de los Estados Unidos de América, citado por Pressman (2010, p. 749) las categorías de riesgo a ser tomadas en cuenta son:

1. Riesgos de Desempeño
2. Riesgos de Costo
3. Riesgos de Soporte
4. Riesgos de Cronograma

En la obra de Ambler (1998, Chapter 4.2.2) se encuentra la clasificación se hace por fase del proyecto así:

1. Fase inicial
 - 1.1. Presiones de la alta gerencia hacen iniciar el proyecto demasiado temprano
 - 1.2. Presión para traer personal al proyecto muy temprano
 - 1.3. Esté alerta al síndrome del juguete o moda nueva
 - 1.4. Resistencia al cambio
2. Definición de los documentos iniciales de Gestión
 - 2.1. Estimados o cronogramas iniciales
 - 2.2. Estimados poco precisos
 - 2.3. Falta de apoyo de la gerencia
3. Fase de justificación
 - 3.1. Políticos
 - 3.2. Saltarse la etapa de justificación
 - 3.3. No mirar los tres aspectos de factibilidad
 - 3.4. Personal poco calificado desarrolla la evaluación del proyecto
4. Otros riesgos potenciales a ser identificados en la fase de Justificación
 - 4.1. Uso de tecnologías no probadas
 - 4.2. Esperar por productos prometidos, pero no lanzados al mercado todavía
 - 4.3. Inexperiencia en la organización con un producto, proceso o tecnología
 - 4.4. Nadie ha intentado un proyecto de este tamaño con la tecnología propuesta
 - 4.5. Uso de varias tecnologías o procesos no familiares
 - 4.6. Potencial falta de habilidad para operar o soportar la aplicación una vez en producción

5. Definición de la Infraestructura del proyecto
 - 5.1. Tomar malas decisiones acerca de: contratistas, tecnología, procesos o entregables.
6. Fase de construcción
 - 6.1. Iniciar sin una Gestión de versiones y configuraciones
 - 6.2. Requerimientos cambiantes
 - 6.3. Soluciones mágicas (*Balas de plata*)
 - 6.4. Futuras liberaciones de software crítico para el proyecto
 - 6.5. Enfrentar demasiadas cosas al mismo tiempo
 - 6.6. Afectar a los desarrolladores con las políticas
 - 6.7. Tener cronogramas programados por calendario y no por trabajo
 - 6.8. Fallas de comunicación con la alta gerencia
 - 6.9. Calcular actividades con alto riesgo de desviación
 - 6.10. Diferencias entre la visión de los desarrolladores y los clientes
 - 6.11. Dejar lo desconocido para después
7. Fase de Diseño
 - 7.1. Miopía. Subestimación del problema.
 - 7.2. Falta de una arquitectura común
 - 7.3. La codificación inicia demasiado pronto
8. Fase de Programación
 - 8.1. Programadores héroes
 - 8.2. Falta de entrenamiento en la infraestructura
 - 8.3. Salida prematura
 - 8.4. No seguir el diseño

8.5. Falta de documentación

8.6. Demasiado enfoque en problemas de optimización

9. Fase de generalización

9.1. Dejar la generalización para futuros proyectos

9.2. Falta de apoyo de la alta gerencia

9.3. Pensar que la fase de generalización sólo sirve para la reutilización de software

10. Fase de pruebas

10.1. Falta de conocimiento

10.2. El proceso de pruebas no es visto como un proceso continuo

10.3. Desarrolladores probando su propio desarrollo

10.4. No se hacen revisiones de código

10.5. No hay estándares de desarrollo o guías de implementación

10.6. Falta de tiempo

10.7. Actitud de “Usted sólo puede probar código”

10.8. Enfoque inadecuado en la tecnología usada para desarrollar

10.9. Subestimación de la importancia de las pruebas de regresión

10.10. Falta de apoyo de la alta gerencia

En el libro de Boyde (2012, Chapter 1.7) se encuentran los siguientes riesgos:

1. Sobrecostos

2. Retrasos en entrega

3. Fallas de calidad

4. Funcionalidad mal implementada

Son realmente los mismos de cualquier proyecto, no necesariamente de software.

En el análisis de Claro Testing (2008), se hace énfasis en el los siguientes riesgos:

1. Retrabajo ocasionado por los defectos o faltantes de las primeras etapas
2. Desincronización entre el desarrollo y las pruebas propuestas.
3. Alejamiento entre el cliente y el proyecto.
4. Hacer énfasis en el costo y tiempo y no en la calidad
5. Enfocar demasiado las pruebas en ciertos puntos de la especificación.

Riesgos genéricos aplicables

Al analizar los diferentes factores de riesgo coincidentes en la literatura, se pueden agrupar así:

Inicialmente se tomarán las categorías enunciadas por Pressman.

1. Riesgos del proyecto: Los autores coinciden en que una deficiente determinación de alguno de los elementos de la triada básica de un proyecto: alcance, presupuesto y tiempo, implican un riesgo inherente para el proyecto. Otro de los elementos comunes y de alto impacto, es la falta de involucramiento del usuario en el desarrollo, esto implica que se puede descartar el uso de metodologías de desarrollo ágiles, ya que uno de sus fundamentos es involucrar a las personas de negocio con las personas técnicas(Beck et al., 2001, p. Principles). En la investigación de The Standish Group (1995, p. 9) este riesgo es el que tiene mayores factores asociados, mostrando que los principales problemas en los proyectos de software son abordables desde de las metodologías de proyectos. Algunos de los riesgos enumerados por Ambler, pertenecientes a esta categoría son: Estimados poco precisos, no mirar los tres aspectos de factibilidad, enfrentar demasiadas cosas al mismo tiempo, falta de apoyo de la alta gerencia y falta de tiempo para pruebas.

2. **Riesgos técnicos:** Los riesgos técnicos son naturales de esta clase de proyectos, el desarrollo de software se encuentra en una etapa artesanal como lo manifiesta Grenfield (2003), nuestra producción todavía tiene todos los problemas del trabajo coordinado y la producción de tecnología a partir de librerías, frameworks, plataformas de desarrollo, herramientas y otros artefactos tecnológicos con los que se puede desarrollar software de forma más sencilla, pero conllevan el riesgo de tener problemas por incompatibilidad de estos artefactos combinados, o el desconocimiento o falta de experiencia en el uso de estas bases tecnológicas.

3. **Riesgos del negocio:** El desarrollo de software tiene riesgos adicionales a otros proyectos de ingeniería, el desarrollo de un producto percibido como barato de modificar, hace que los cambios de alcance o modificación del objetivo de negocio se dé con mayor facilidad, como lo explica Pressman los principales riesgos de negocio que se pueden presentar son:
 1. Construir un excelente software que nadie quiere realmente. Este riesgo es señalado por la mayoría de estudios, y es algo común, un cambio regulatorio o una modificación al mercado y es probable que el problema para el que se desarrolló el software ya no exista, o haya cambiado tanto que es necesario reconstruir el producto.
 2. Construir un software que ya no encaja con las necesidades estratégicas del negocio. Los problemas de especificación y las modificaciones del mundo del problema hacen que el software realmente no encaje con las estrategias planteadas inicialmente.
 3. Construir un producto que la fuerza de ventas no entiende cómo vender. Cuando el desarrollo es un producto comercial, es posible que el mercado no requiera la

- solución o sea percibida por los vendedores como inútil o carente de sentido. Nada es más difícil de vender que un producto en el que el vendedor no crea.
4. Perder el apoyo de los patrocinadores del proyecto por un cambio de foco o cambio de personas en posiciones de poder.
 5. Perder compromiso de presupuesto o personal. Tao lo manifiesta como problemas de personal.
-
4. Riesgos de comunicación: Uno de los problemas más notorios en el desarrollo de software es el problema de comunicación que hay entre las personas de negocio y las personas técnicas, este riesgo involucra problemas de especificación, pero adicionalmente se puede presentar en otras etapas, por ejemplo, en la etapa de pruebas una comunicación ineficiente genera falsos hallazgos y retrabajos por incomprensión de las especificaciones o el diseño de la aplicación. En los entornos de desarrollo con equipos distribuidos geográficamente se puede presentar: horarios de empalme reducidos por diferencias horarias, incomprensión de requerimientos por diferencias de idiomas o cultura y conflictos por coordinación de tareas (Se puede dar también localmente cuando una misma actividad es desarrollada en varios horarios por personas diferentes, v.g. desarrollo de software o pruebas en diferentes turnos sobre el mismo componente).

Modelo de implementación

En este capítulo se abordarán los componentes y modelos de desarrollo de software utilizados en la industria con mayor aceptación, y la forma en la cual estos se pueden aplicar con los patrones generados en esta investigación. Inicialmente se explicarán componentes de los SDLC y las implicaciones de su uso, y cómo estos pueden ser utilizados para implementar los patrones, y en la última parte se abordará un estudio breve de los modelos.

Implicaciones del ciclo de desarrollo de software

Los SDLC tienen divisiones normalmente llamadas fases o iteraciones y en algunos de ellos reciben nombres como ciclo o *sprint* (En el caso de modelos como Scrum), o tienen fases claramente establecidas. En los ciclos de desarrollo seriales como Cascada o el Modelo en V (Bundesrepublik Deutschland, 2008), sólo se dan fases las cuales siempre van consecutivas unas a otras. En procesos cíclicos como el propuesto por Boehm (1988) o el Unified Process (Jacobson et al., 1999) las divisiones naturales son ciclos y en el caso del proceso unificado adicionalmente existen fases (Principio, Elaboración, Construcción y Transición). En ocasiones es la configuración específica de los componentes de un SDLC los que deben variarse para implementar la solución propuesta por los patrones. En la obra de Ambler(1998, p. 35) se hace referencia a una serie de patrones diferente al manejado en este trabajo, allí se manejan patrones de tarea, etapa y fase, mientras que aquí se busca dar sentido a toda la estructura de componentes del SDLC para solucionar una problemática particular.

El usar los SDLC implica a una serie de reglas o preconceptos como en el caso de Scrum, donde se espera involucramiento de los usuarios finales, equipos de trabajo con individuos auto organizados, multifuncionales y que rinden cuentas al mismo equipo (Schwaber & Sutherland, 2011), o en Extreme Programming, conocido como XP, donde se espera una serie de valores específicos (Pressman, 2010, pp. 72–79), que implican organizaciones particulares de la forma de trabajo.

Modelos de Ciclo de desarrollo de software

Los SDLC más utilizados en el mercado se nombrarán en esta investigación como alternativas de implementación de los patrones, v.g. si el patrón tiene como posible alternativa el uso de un SDLC orientado a programación ágil, no hay un criterio definido en ninguna literatura que permita seleccionar uno en especial, pero se orientará hacia el uso de una metodología orientada a los criterios establecidos en esta clase de SDLC.

A continuación, se hará una breve reseña de los SDLC más reconocidos en la industria actualmente y que serán utilizados para las posibles implementaciones de los patrones:

- **Modelo en Cascada:** Popularizado en la década de 1970, pero existente desde hace más tiempo, Boehm sugiere que desde antes de 1956 (1988), es la forma más común, intuitiva y usada para desarrollar software en la industria (Ambler, 1998, p. 37); el artículo de Winston Royce (1970), fue uno de los más influyentes durante años para trabajar con este SDLC, con algunas propuestas de mejora que realmente no tuvieron

mucha acogida (Walker Royce, 1998, p. 11). Se basa en el paso de resultados de trabajo de una etapa a la siguiente de forma serial, las etapas propuestas son: requerimientos de sistema, requerimientos de software, análisis, diseño, codificación, pruebas e implementación en producción. El esquema de estas etapas se ve en el diagrama a continuación como lo explicó Royce en su artículo:

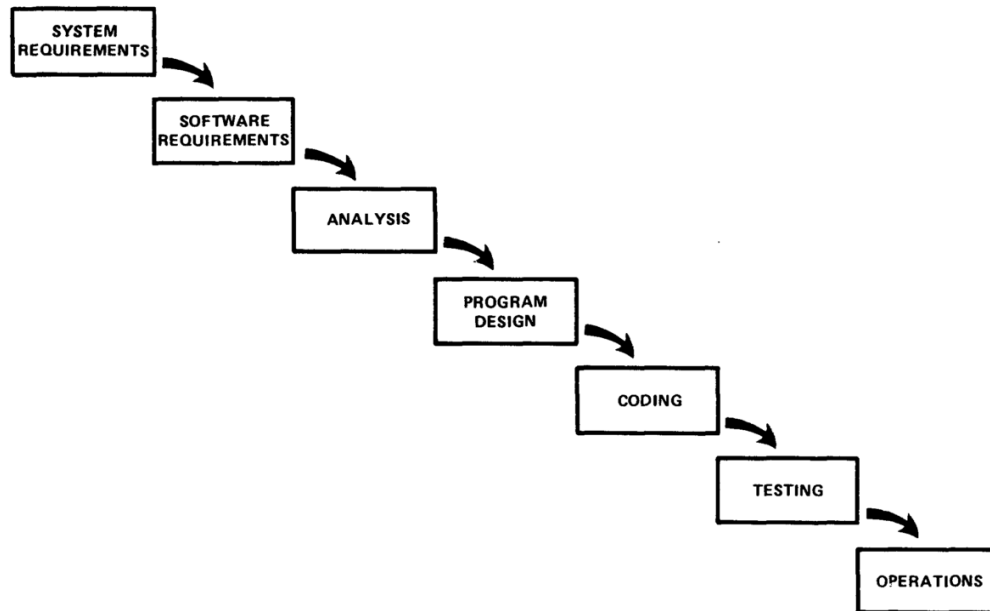


Figura 4 Tomado de Royce, W. (1970, p. 329)

Una de sus ventajas es tener claramente definidos los entregables por etapa, es usado en proyectos donde el presupuesto debe ser definido desde el inicio, sin embargo, uno de los mayores obstáculos que empezaron a verse reflejados en el modelo Cascada, aparecieron con las primeras aplicaciones orientadas a usuario final con interfaz gráfica (B. W. Boehm, 1988, p. 63), en ese momento las modificaciones a las especificaciones se empezaron a hacer cada vez más necesarias, y se entendió que las estimaciones se basan en una falacia: “Los requerimientos se pueden documentar y mantener estables”; alrededor de desmentir dicha falacia se estructuraron los procesos cíclicos y ágiles, por la misma naturaleza del software, como elemento que impacta

los procesos organizacionales, es imposible mantener inalterados los requerimientos. Cuando se detectan errores, el proceso debe devolverse a la etapa en la cual se debe corregir, volviendo a pasar por todas las etapas de nuevo, lo cual como lo manifiesta Walker Royce (Hijo de Winston Royce, autor original del artículo sobre el modelo Cascada) (1998, p. 13), incrementa el tiempo de resolución de los riesgos.

- Modelo en V: Propuesto por el gobierno alemán en 1997 en el documento: “Estándares de desarrollo para sistemas IT de la República Federal de Alemania”, se basa en el avance secuencial de las actividades como el modelo Cascada, pero permite anticipar los procesos de calidad disminuyendo la incertidumbre en el proceso de desarrollo, puede mejorar la comunicación con los interesados, reducir costos y adicionalmente, se afirma en el documento del gobierno Alemán que se asegura la calidad, pero es algo discutible (Clarotesting.com, 2008). Actualmente se maneja un sucesor del Modelo en V original o V-Modell 97 como suelen denominarlo, se llama V-Modell XT(Bundesrepublik Deutschland, 2008).

El modelo es configurable con estrategias específicas según el tipo del proyecto:

- Proyecto de desarrollo de un sistema (Adquiriente)
- Proyecto de desarrollo de un sistema (Proveedor)
- Proyecto de desarrollo de un sistema (Adquiriente/Proveedor)
- Introducción y mantenimiento de un modelo de proceso específico para una organización

Igual que en Cascada hay grupos de trabajo con roles, actividades y entregables específicos por actividad. De acuerdo a la forma en la que se configure, puede tener grupos de trabajo dispersos

para cumplir diferentes actividades. Parte de las configuraciones por estrategia se pueden ver en la figura a continuación.

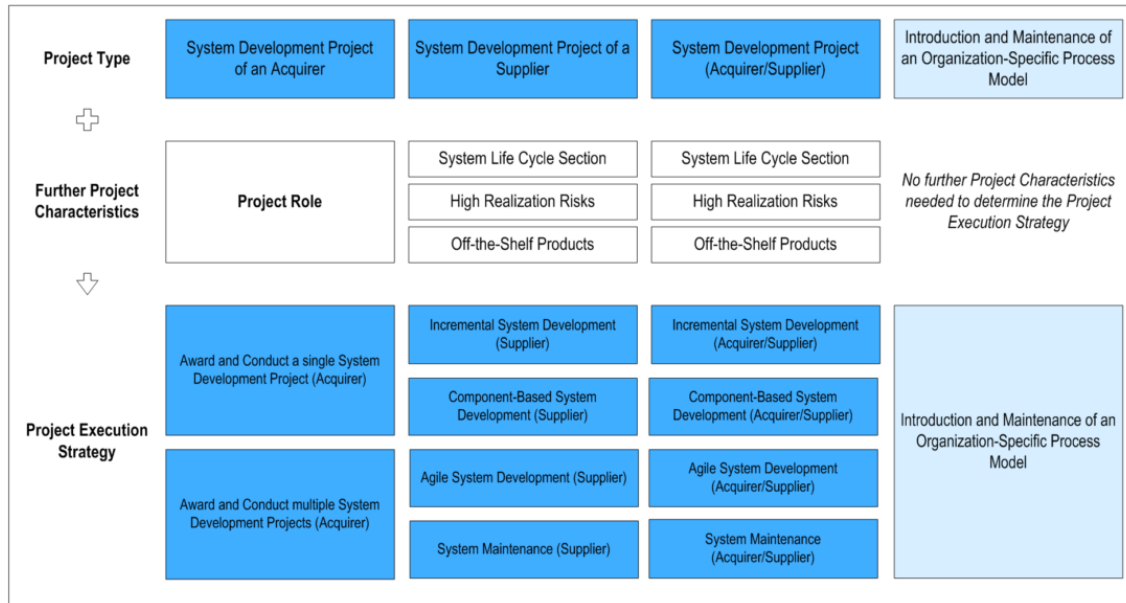


Figura 5 Ubicación de estrategias de ejecución por tipo de proyecto Modelo V. Tomado de (Bundesrepublik Deutschland, 2008, pp. 1–17)

En cada etapa hay una *Puerta de Decisión* donde de acuerdo a la calidad de los entregables propios de la etapa, un comité decide si se procede a la siguiente etapa, se devuelve en el proceso o se cancela el proyecto.

El Modelo V es más que un SDLC, es una metodología de gestión de proyectos, aplicable a hardware y software que incluye actividades relacionadas con contratación, subcontratación y manejo de multiproyectos, y que involucra un ciclo de vida determinado por controles de calidad, como se observa representado en la figura que se encuentra a continuación.

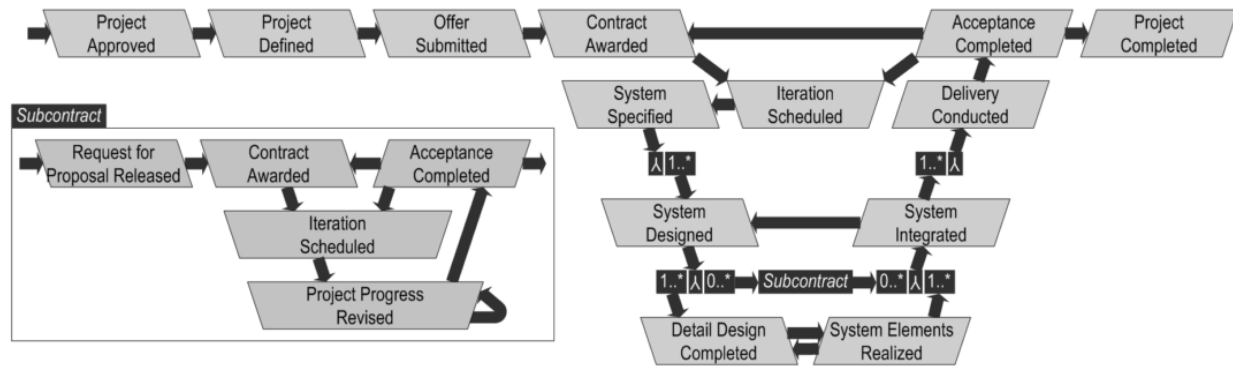


Figura 6 Modelo V completo tomado de (Bundesrepublik Deutschland, 2008, pp. 3–74)

- Modelo en espiral: Propuesto por Barry W. Boehm (1988), se basa en la mitigación del riesgo al generar ciclos de desarrollo, fue creado debido a los problemas que se estaban teniendo en la industria del software con el modelo Cascada, de hecho en la investigación original se refleja que los problemas con este modelo son un tema recurrente.

Una de las fortalezas del modelo propuesto por Boehm es que enfrenta la incertidumbre sobre las especificaciones mediante ciclos sucesivos de maduración, en los cuales el usuario final va recibiendo entregables y avanza hacia una definición más concreta en cada ciclo. Se basa en que es posible que los usuarios finales digan: “No sé lo que quiero, pero cuando lo vea lo sabré”. A continuación, se encuentra el diagrama del modelo espiral originalmente propuesto.

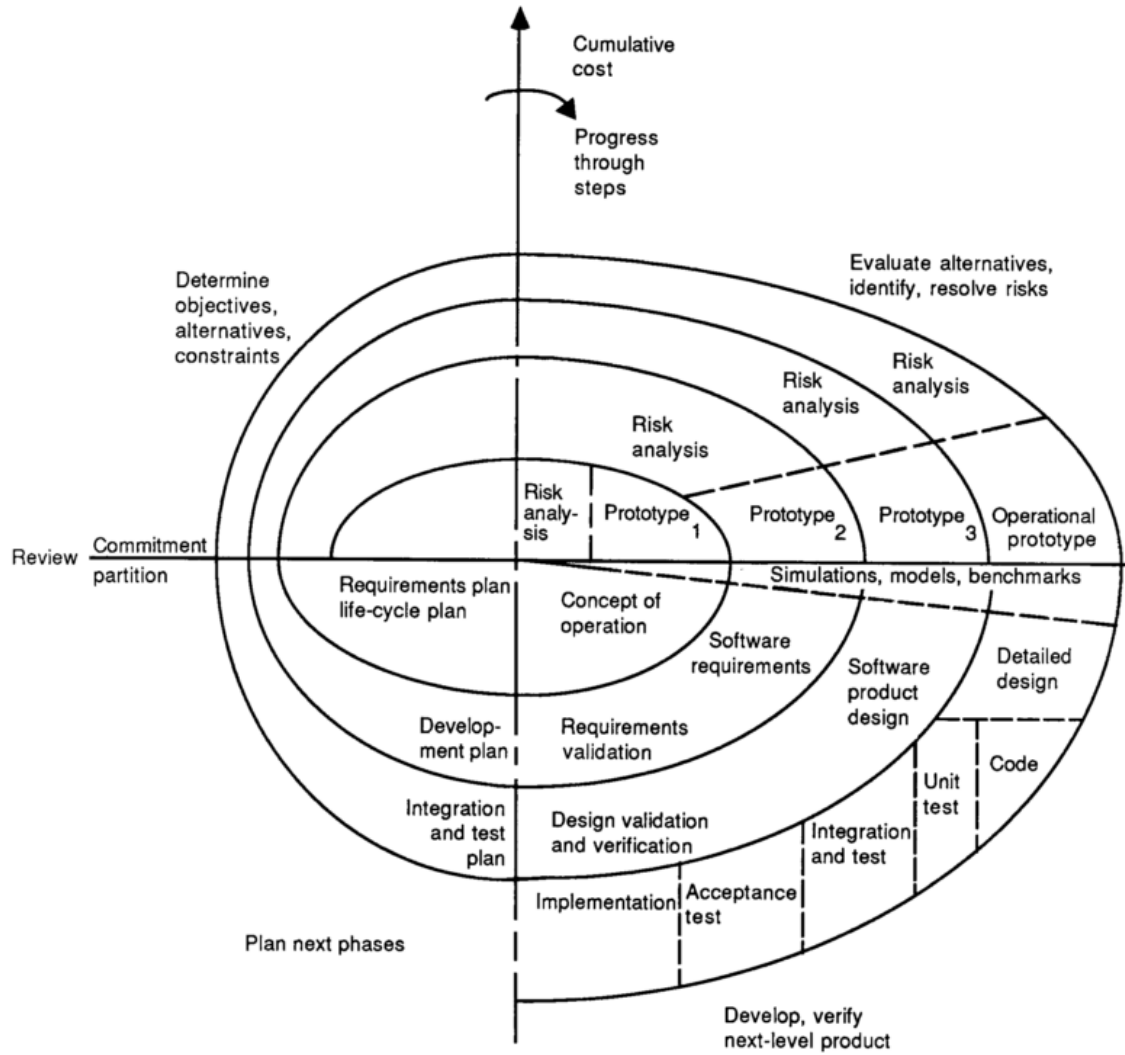


Figura 7 Modelo en Espiral tomado del artículo de Boehm (1988, p. 64)

Como se puede apreciar, el modelo contempla la generación de prototipos (En este caso tres), hasta tener la certeza de lo que se desea desarrollar y en su ciclo final generar el desarrollo que realmente quiere el usuario. Al final de cada ciclo se hacen revisiones de los objetivos logrados y el siguiente ciclo a implementar, determinando: Objetivos, restricciones, alternativas, riesgos, resolución de riesgos, resultados de la gestión de riesgos, plan para la siguiente fase y los compromisos.

Por supuesto no es hay número fijo el ciclos iniciales o prototipos que se deban desarrollar, ni certeza sobre la utilidad para el usuario final del prototipo desarrollado en cada ciclo, lo cual puede convertirse en un gasto no determinable al inicio del proyecto sin retornos tempranos en el peor de los casos. Otro riesgo importante es que, al reutilizar código y estructuras entre ciclos, es posible caer en algo que el autor denomina, código espagueti, o código de reutilización muy compleja, lo cual es lógico si haciendo un símil, al inicio del proyecto se va a construir un edificio de cinco pisos, pero realmente se necesita un rascacielos, las decisiones arquitectónicas iniciales del modelo original pueden estar completamente erradas, lo cual puede llevar a retrabajos y costos más elevados de lo planeado para todo el proyecto.

El modelo ha ido evolucionando, una forma más avanzada del modelo se propuso por su creador y se puede ver a continuación:

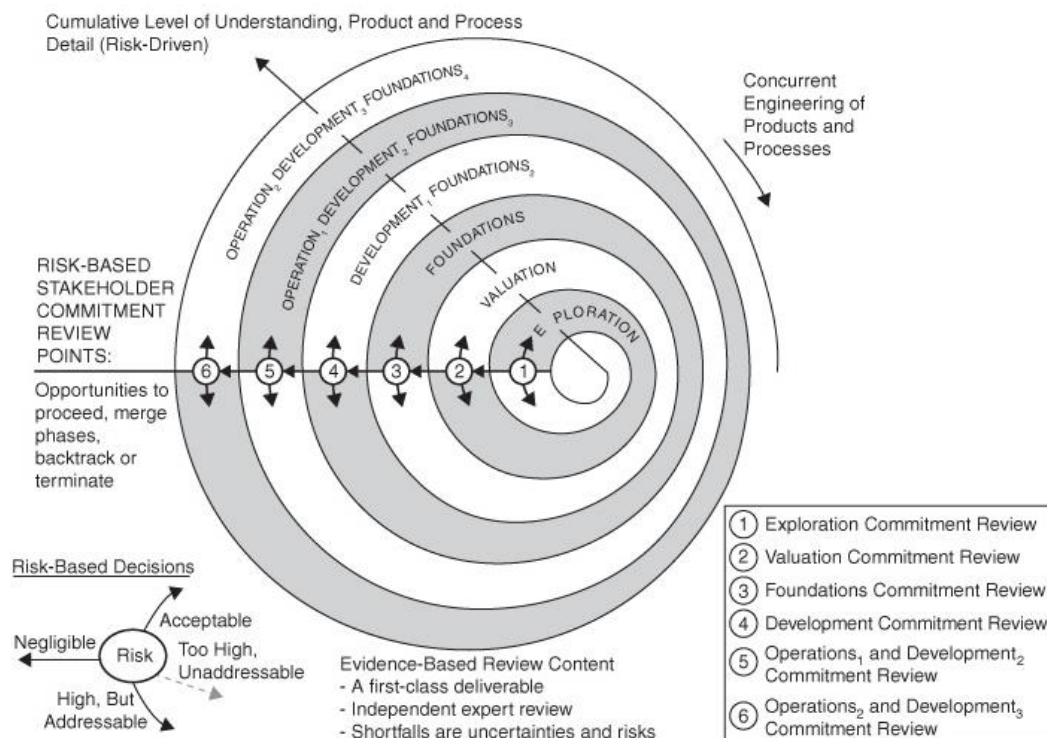


Figura 8 Modelo en Espiral actualizado tomado de *The incremental commitment Spiral Model* (B. Boehm, Lane, Koolmanojwong, & Turner, 2014)

En su nueva versión tiene cuatro principios (B. Boehm et al., 2014):

1. Guía de los *Stakeholders* basado en el valor
 2. Compromiso y rendimiento de cuentas incremental
 3. Ingeniería multidisciplinaria concurrente
 4. Decisiones basadas en evidencia y riesgo
-
- Proceso Unificado: Originalmente propuesto por la empresa Rational (Empresa adquirida por IBM), por esto su nombre original fue RUP (Rational Unified Process, el Proceso Unificado de Rational), propuesto en 1999 por Ivar Jacobson, Grady Booch y James Rumbaugh (1999). Es descrito como: “RUP es una aproximación al desarrollo de software que es iterativa, centrada en arquitectura y guiada por casos de uso” (Kroll & Kruchten, 2003). Se compone de cuatro fases principales: Inicial, Elaboración, Construcción y Transición, pero cada fase se puede configurar para tener iteraciones (Repeticiones) de acuerdo a las necesidades del proyecto, durante las fases y las iteraciones se desarrollan una serie de disciplinas que son las actividades necesarias para llevar a cabo el desarrollo del software. Como se puede observar en la figura a continuación la configuración de disciplinas, fases e iteraciones permiten desarrollar el software esperado.

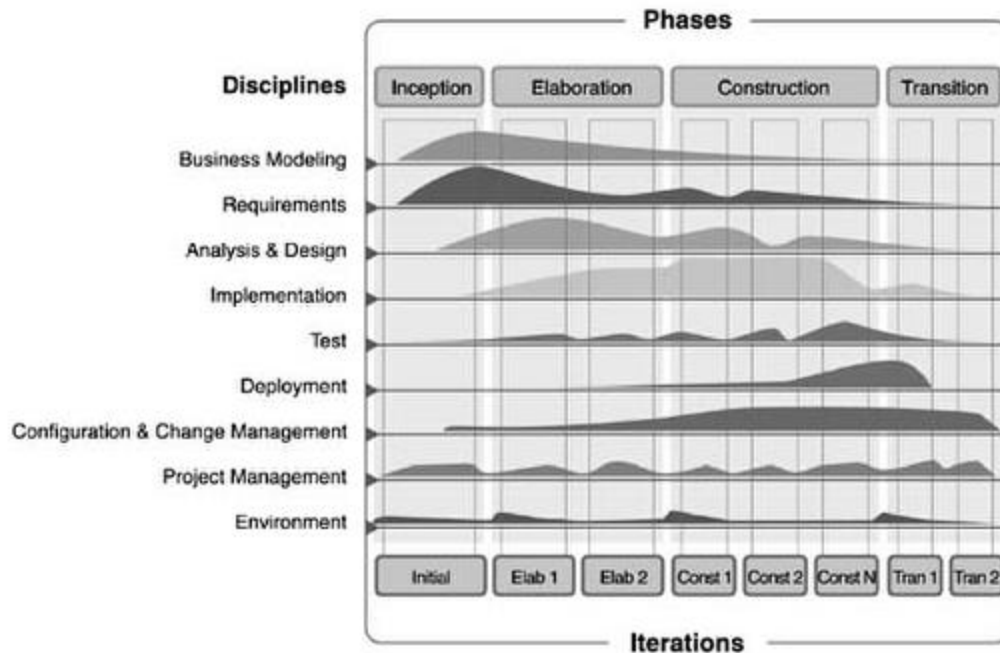


Figura 9 Composición del RUP. Tomado de (Kroll & Kruchten, 2003)

Se basa en un modelo cíclico, con herramientas de configuración y que establece claramente los entregables, actividades y roles responsables, como los otros modelos, busca aproximarse a la creación de software desde un esquema orientado a la reducción de los riesgos. Al igual que el Modelo en Espiral, pretende dar soluciones ejecutables para mejorar la validación de las necesidades del cliente.

Unas de las mayores críticas que ha tenido es por su gran cantidad de documentación, aunque este punto es configurable según el tamaño del proyecto (Borges, Monteiro, & Machado, 2011), en líneas generales sus autores han hecho énfasis en que cada proyecto debe definir los artefactos que son requeridos. Generalmente es uno de los SDLC más aceptados y utilizados en la industria, pero su implementación requiere mayores conocimientos en administración de proyectos y control de las fases e iteraciones, tuvo un gran despliegue cuando Rational e IBM lo promocionaron como el proceso apropiado para trabajar con lenguajes orientados a objetos, como Java.

- Extreme Programming (XP): La programación extrema (XP) es una metodología de Ingeniería del software formulada por Kent Beck (Inteco, 2009) cuando trabajaba para Chrysler, en un proyecto que duró siete años. Los planteamientos quedaron plasmados en el libro Extreme Programming Explained: Embrace Change (Beck, Andres, & Gamma, 2005). Está hecho para grupos entre dos y diez personas, es considerado una disciplina ya que requiere una configuración de trabajo diferente a lo que se usa tradicionalmente, por lo cual representa un choque cultural para las personas que se involucran inicialmente.

También pretende manejar los riesgos usuales en proyectos de software mediante iteraciones cortas con resultados rápidos, tratando de minimizar el costo de los defectos al resolver los defectos en las fases iniciales del proceso. Sus valores son:

1. Comunicación
2. Simplicidad
3. Retroalimentación
4. Coraje

A partir de estos valores se generan unos principios fundamentales:

1. Retroalimentación rápida
2. Asumir simplicidad
3. Cambio incremental
4. Aceptar el cambio
5. Calidad de trabajo

Sus prácticas son algo de lo más conocido:

Planeación de las iteraciones, Pequeñas liberaciones, Metáforas, Diseño simple, Pruebas, Refactorización, Programación por pares, Pertenencia colectiva del código, Integración continua, 40 horas a la semana, Cliente en sitio y Estándares de codificación.

En XP algunas de sus prácticas, como la planeación de etapas, pequeñas liberaciones, integración continua, estándares de codificación y cliente en sitio, son ampliamente adoptadas, otras como: programación por pares, 40 horas a la semana y pertenencia colectiva, no han tenido mucha acogida.

- Scrum: Es un marco de trabajo para desarrollar y sostener productos complejos (Schwaber & Sutherland, 2011). Fue propuesto en 1986 por Hirotaka Takeuchi y Ikujiro Nonaka (1986), usando el símil del Rugby, donde los jugadores llevan la bola por todas las etapas del juego como un mismo grupo, unidos para lograr el objetivo, aunque realmente empezó a usarse en los años noventa por el impulso de Ken Schwaber.

Las divisiones de trabajo, o iteraciones del desarrollo se conocen como *Sprint* y son determinadas por el equipo de trabajo con la aceptación por parte del cliente, quien define la priorización de funcionalidades a desarrollar, por lo general se utilizan Historias de usuario para hacer las especificaciones, y las dudas son resueltas por el usuario en sitio. Al final de cada *Sprint* se espera un resultado ejecutable y usable por el cliente, la duración de estos por lo general debe ser inferior a tres semanas de trabajo, la suma de las Historias desarrolladas se conoce como Incremento. Las Historias de usuario que no se han desarrollado se acumulan en un repositorio llamado *Backlog*.

A continuación, se encuentra una gráfica que muestra de forma simple el proceso de Scrum:

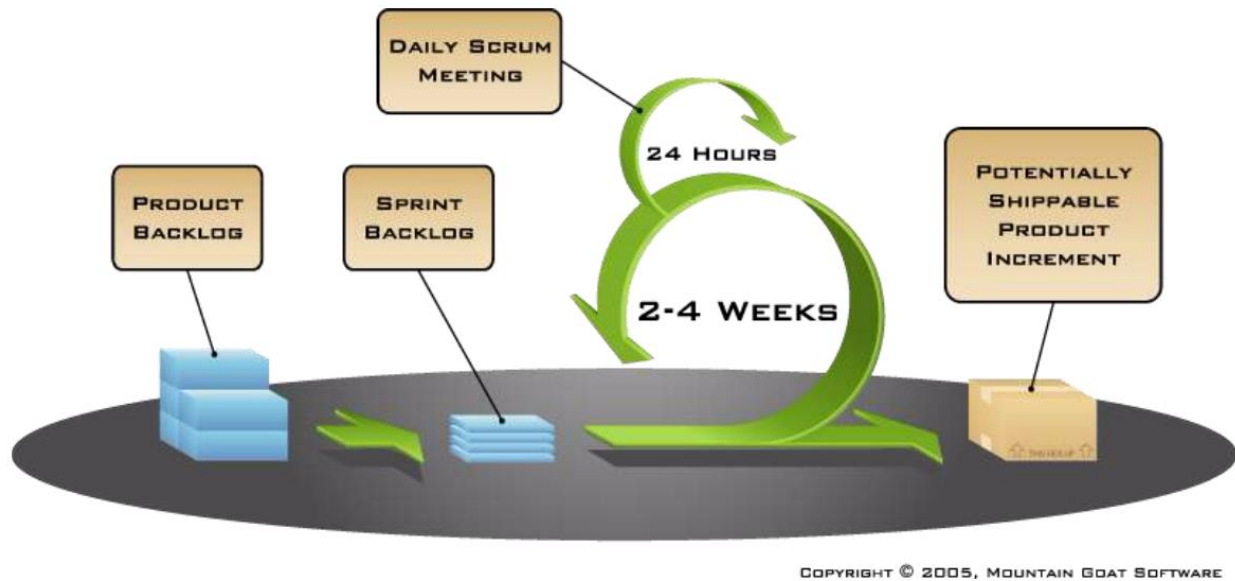


Figura 10 Tomado de www.mountaingoatsoftware.com/scrum

Tiene cuatro eventos de inspección y adaptación: Planeación del *Sprint*, *Scrum* diario, Revisión del *Sprint* y Retrospectiva del *Sprint*. La reunión de Scrum diario es una reunión de 15 minutos, donde el equipo tiene la oportunidad para revisar con el cliente las inquietudes del equipo de trabajo, y el avance de las actividades programadas, dando la oportunidad al *Scrum Master* de resolver los riesgos detectados y apoyar en las insuficiencias de conocimiento o bloqueos tecnológicos. El *Scrum Master* es el líder encargado de hacer que el equipo sea escuchado y que este se acoja a las prácticas establecidas por Scrum.

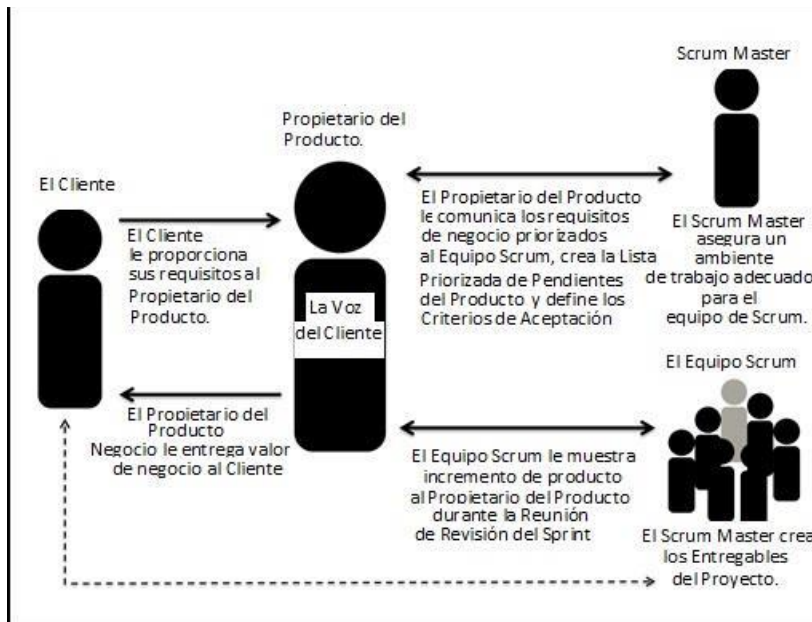


Figura 11 Organización en un proceso SCRUM. Tomado de (Satpathy, 2016)

Algunas de las críticas que ha recibido Scrum son comunes con otros SDLC cíclicos, como no tener control del tiempo o presupuesto totales requeridos, adicionalmente, es posible que un proyecto se desdibuje y termine convertido en un servicio de desarrollo sin límite de tiempo, lo cual va en contravía de los principios de gerencia de proyectos. A continuación, se encuentra un cuadro comparativo de SCRUM con metodologías tradicionales.

Ítem	Scrum	Gestión de proyectos tradicional
El énfasis está en	Personas	Procesos
Documentación	Sólo mínima—según se requiera	Exhaustiva
Estilo de procesos	Iterativo	Lineal
Planificación por adelantado	Baja	Alta
Priorización de los requisitos	Según el valor del negocio y regularmente actualizada	Fijo en el plan de proyecto
Garantía de calidad	Centrada en el cliente	Centrada en el proceso
Organización	Auto-organizada	Gestionada
Estilo de gestión	Descentralizado	Centralizado
Cambio	Actualizaciones a la lista priorizada de pendientes del producto	Sistema formal de gestión del cambio

Ítem	Scrum	Gestión de proyectos tradicional
Liderazgo	Liderazgo colaborativo y servicial	Mando y control
Medición del rendimiento	El valor del negocio	Conformidad con el plan
Retorno sobre la inversión (RSI)	Al comienzo y a lo largo del proyecto	Al final del proyecto
Participación del cliente	Alta durante todo el proyecto	Varía en función del ciclo de vida del proyecto

Tabla 2 Comparación de proyectos Scrum y tradicionales. Tomado de (Satpathy, 2016, p. 19)

Patrones de Ciclo de Vida

Modelos de Patrones

A continuación, se describirá el modelo de patrones aplicable para este cuerpo de conocimiento que es la administración de proyectos de software, el formato estándar descrito por Alexander, C (1979, citado por Tesanovic 2001, p. 6), contiene los siguientes elementos básicos:

1. Nombre: Nombre corto de fácil recuerdo que describe la solución o el ámbito de aplicación, también puede ser un mnemotécnico de la forma en la que se conforma
2. Problema: Situación problemática que tiende a solucionar
3. Contexto: Variables aplicables para determinar su uso
4. Fortalezas: Ventajas de implementarlo
5. Solución: Descripción de la solución propuesta en el patrón
6. Ejemplo: Ejemplo de aplicación
7. Resultado de aplicarlo: Estado después de ser aplicado
8. Racional de uso: Explicación de las razones lógicas por las cuales el patrón es aplicable, cómo su funcionamiento produce los resultados esperados.

9. Patrones relacionados: Otros patrones aplicables en un contexto similar, puede ser para reemplazar el actual patón o para formar una solución más compleja.
10. Usos conocidos: Aplicaciones del patrón conocidas en ambientes laborales que apoyen su uso.

Pero revisando la documentación en SFB (Nehmer et al., 2000) mencionada en el trabajo de Iida(1999), el modelo usado para los patrones tiene los siguientes atributos:

- Problema: Dificultad a ser solucionada usando el patrón.
- Condiciones: Factores adicionales para las necesidades del patrón, restricciones.
- Contexto: Describe la situación para la aplicabilidad del patrón y el contexto resultado de aplicarlo o situación a ser establecida incluyendo diagrama del proceso.
- Descripción: descripción en lenguaje natural
- Observaciones: otros comentarios incluyendo racional de uso, claves de clasificación, patrones relacionados, etc.

Y en los patrones de procesos de desarrollo de software trabajados en el Taller de la Universidad de Múnich (2002), se manejaron los siguientes atributos de los patrones trabajados planteados por Gnatz, M., Marschall, F., Popp, G., Rausch,A. y Schwerin, W.:

- Nombre: Nombre del patrón.
- También conocido como: Otros nombres del patrón si los tiene.
- Autor: Nombres de los autores del patrón.
- Propósito: Un resumen de la intención del patrón y razón fundamental.
- Problema: Dificultad de desarrollo a ser solucionada usando el patrón. Si es posible incluyendo una situación donde el patrón pudo ser usado

- Contexto: La situación o estado del proyecto de desarrollo en el cual el patrón de proceso puede ser aplicable. Contiene los prerequisites, artefactos y estructura para usar el patrón. En este trabajo se adicionarán las variables que ayudarán a decidir su uso.
- Solución: El proceso de software de desarrollo sugerido incluyendo las actividades de desarrollo dentro del patrón de proceso. La solución propuesta puede ser descrita usando técnicas textuales y gráficas.
- Consecuencias: Beneficios que trae el patrón, cualquier posible desventaja y riesgos involucrados.
- Usos conocidos: Usos conocidos del patrón en proyectos de desarrollo. Estas aplicaciones de ejemplo ilustran la aceptación y usabilidad del patrón, y pueden proveer guías prácticas, claves y técnicas útiles de aplicar el patrón, pero también mencionando contra ejemplos y fallas.
- Ver también: Referencias a patrones que resuelven problemas similares y patrones que ayudan a refinar el patrón que se describe. Fuentes no relacionadas con el patrón también se pueden referenciar.

Este último modelo es el que se va a utilizar en este trabajo, en este ya hay varios patrones enunciados en el taller de la universidad de Múnich y se adicionarán a los patrones de RUP (IBM Corporation, 2005), los patrones enunciados por el Departamento de Justicia de los Estados Unidos (2003, Chapter <https://www.justice.gov/archive/jmd/irm/lifecycle/ch13.htm#para13.3>) y los patrones enunciados por el grupo SFB (Iida, 1999), todos estos patrones se compararán y

unificarán cuando correspondan al mismo concepto. Adicionalmente, se enunciarán los patrones correspondientes a desarrollo con fábricas de software o outsourcing.

Patrones de ciclo de vida

Esta sección contiene los patrones representados en diferentes fuentes bibliográficas y serán unificados de acuerdo al problema y solución propuesta.

Atributo	
Nombre	Desarrollo de sistemas en tiempo real basado en modelos. Tomado de Proceedings of the 1st Workshop on Software Development Patterns (SDPP'02) (Technische Universität München, 2002)
También conocido como	Diseño de sistemas embebidos en tiempo real basado en modelos
Autor	Proyecto HRS, Orehek, M., Instituto para Sistemas computacionales en tiempo real de la Universidad Técnica de Múnich
Propósito	<p>Desarrollo de sistemas embebidos con limitaciones estrictas de tiempo real utilizando un modelo gráfico central para describir los diferentes aspectos funcionales del diseño. El modelo se utiliza como una especificación ejecutable en un entorno virtual.</p> <p>En las tres fases del proceso de desarrollo, se cubren los aspectos especiales como la simulación de comportamiento físico de los nuevos componentes diseñados, la tarea de diseño de sistemas de control asociado y la implementación final, son cubiertas teniendo en cuenta los aspectos en tiempo real.</p>
Problema	<p>Es necesario adicionar componentes aún no completamente implementados para ser integrados y controlados por el software a desarrollar. Los retos consisten en desarrollar componentes físicos y el software que los van a controlar.</p> <p>Las partes de los componentes físicos (por ejemplo, actores) e incluso el hardware controlador final (por ejemplo, tarjeta de conexión micro) se desarrollan durante el proceso de diseño general. Por tanto, el diseño de software funcional tiene que ser desacoplada de dichos aspectos de forma constante evolución, evitando restricciones y costes innecesarios.</p> <p>La aplicación de software final tiene que tener en cuenta al lado de los requisitos no funcionales y funcionales también, al igual que los peores escenarios de tiempos de respuesta a ciertos eventos.</p>
Contexto	<p>No hay certeza sobre todos los componentes a desarrollar y el poder probar el ensamble de dichos componentes es fundamental para el resultado final.</p> <p>Los requisitos de trabajo necesarios son: especificación de requerimientos (parte funcional y no funcional).</p> <p>Los entregables de trabajo producidos son: modelos de simulación, modelos de controladores validados (con los correspondientes análisis de la estabilidad y la estimación de calidad, etc.), generación rápida de prototipos (datos medidos, la estimación de destino integrados, etc.), datos de medición, sistema final diseñado (solución objetivo incorporado con resultados de análisis en tiempo real).</p> <p>Variables: Número de riesgos: alto</p>

	<p>Estabilidad y claridad de requerimientos: inestables Experiencia del equipo de desarrollo: Alta (Es un tema muy específico) Flexibilidad al cambio: Alta Involucramiento del usuario: Normal Distribución geográfica: Equipo local Uso de Fábricas de software: Desarrollo <i>inhouse</i> o con una fábrica</p>
<p>Solución</p>	<p>En la Figura 10 el patrón de proceso se representa en el más alto nivel. Durante el ciclo de desarrollo de productos (actividad: Desarrollar un sistema embebido en tiempo real) todos los componentes físicos necesarios del producto, sus estrategias de control correspondiente y sus implementaciones de software se han desarrollado y optimizado. El artefacto que inicia el trabajo es la especificación de requerimientos.</p> <p style="text-align: center;"><i>Figura 12 Flujo patrón Desarrollo de sistemas en tiempo real basado en modelos. Fuente (Technische Universität München, 2002)</i></p> <p>Para cada uno de los componentes físicos se crea un modelo de simulación, el cual se realiza como resultado de la actividad: especificar el ambiente. Estos modelos se utilizan para simular el comportamiento y optimizar de forma iterativa los componentes físicos diseñados, reduciendo al mínimo los tiempos de rotación y generación de costosos prototipos físicos. Sólo soluciones satisfactorias se realizan como prototipos y luego se validan mediante experimentos reales.</p>

Mientras los componentes físicos evolucionan, también las estrategias del controlador deben ser perfeccionadas durante la actividad: sistema de control de diseño. Antes de su aplicación, su calidad tiene que ser evaluada.

La actividad: implantar el modelo de controlador se divide en dos etapas. En un primer paso, los controladores simulados se realizan como prototipos rápidos y permiten obtener mediciones del mundo real del rendimiento alcanzado. Estos datos medidos se realimentan en el proceso de diseño para refinar los modelos físicos y aumentar la confianza y el conocimiento de los componentes desarrollados.

El soporte de herramientas de software necesario (por ejemplo, librerías de controladores y ambientes de ejecución) se puede construir al mismo tiempo que todas las demás tareas en curso de ser refinadas.

Para la implementación final, se ofrece una medida y estimación de los recursos objetivo utilizados, poniendo a disposición los parámetros necesarios para el análisis en tiempo real. Esto asegura el cumplimiento de la solución de software con los peores escenarios planteados en los requisitos de rendimiento de la especificación.

El modelo propuesto se muestra a continuación:

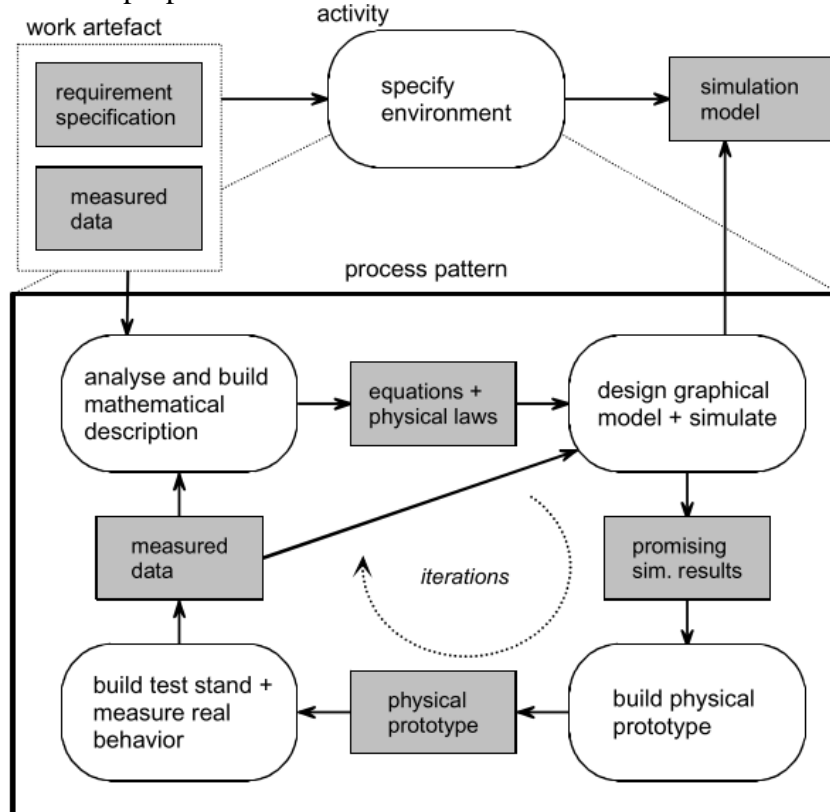


Figura 13 Flujo propuesto patrón Desarrollo de sistemas en tiempo real basado en modelos. Fuente (Technische Universität München, 2002)

En el caso, de que todos los requisitos de tiempo se puedan cumplir, el modelo utilizado está adaptado para la implementación final en tiempo real, utilizando bloques gráficos para conectar las señales de software para las interfaces de hardware. Después de una prueba final, el sistema integrado desarrollado está disponible.

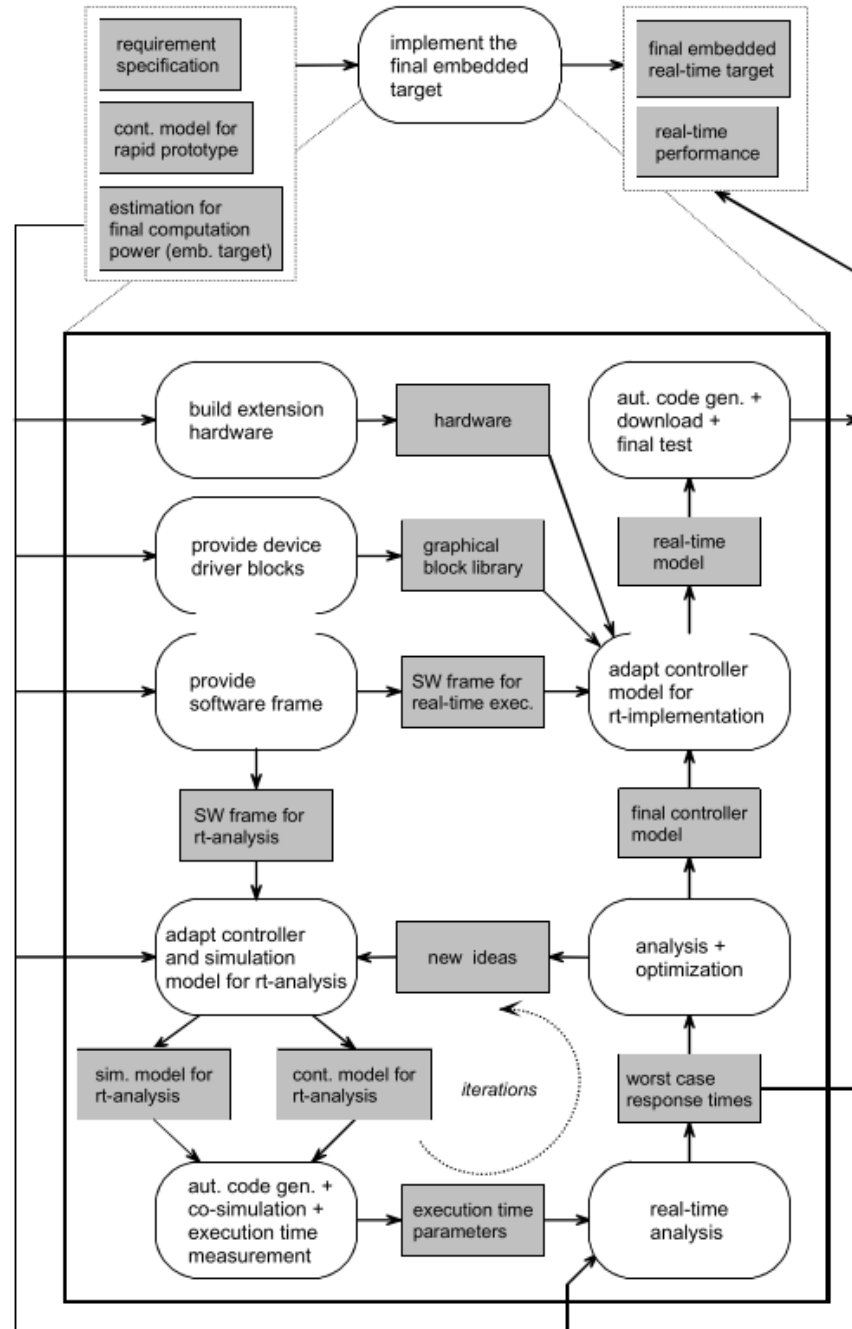


Figura 14 Modelo para la fase de implementar el objetivo embebido final, tomado de (Technische Universität München, 2002)

Los Modelos de ciclo de vida recomendados son los iterativos como el Modelo en espiral, RUP para pequeños proyectos, Scrum y XP, debido a

	que gradualmente se va reduciendo el riesgo y se va avanzando hacia un sistema que incorpora los componentes.
Consecuencias	<p>Beneficios:</p> <ul style="list-style-type: none"> • Usando una descripción central y uniforme (modelo dentro de una cadena de herramientas) facilita el intercambio de información entre los diferentes equipos de ingeniería y permite una fácil reutilización y regeneración de nueva información, pero todavía permitiendo concentrarse en diferentes aspectos del diseño del sistema (por ejemplo, el modelado de las leyes físicas en fase 1, diseño de control en la fase 2). • Debido a la función de simulación en las primeras fases del ciclo los tiempos de retorno puede reducirse a un mínimo y el número de prototipos realizados puede ser disminuido. Las nuevas ideas se pueden evaluar con mayor rapidez lo cual conduce a nuevas soluciones eficientes y componentes de hardware mejor entendidas. • La realización eficiente, garantizando todas las limitaciones de tiempo requeridos, se puede lograr mediante: <ul style="list-style-type: none"> - La adopción de una arquitectura de software analizable para mapear el modelo gráfico del controlador en el sistema embebido en tiempo real - - Proporcionar las ecuaciones matemáticas necesarias para el análisis - - Medir, respectivamente, deduciendo los parámetros específicos del modelo es necesario (por ejemplo, tiempos de ejecución, las prioridades de partes de modelo, etc.). <p>Desventajas:</p> <ul style="list-style-type: none"> • El enfoque basado en modelo presentado sólo se puede aplicar de manera eficiente con un soporte de la herramienta adecuada. • Hay un esfuerzo ligeramente mayor en el primer enfoque para proporcionar un soporte del Software reutilizable (biblioteca gráfica de bloques para objetivo incorporado) para el objetivo de hardware elegido, pero por otro lado permite acelerar proyectos futuros con el mismo objetivo final o pasar de un objetivo a otro. <p>Riesgos:</p> <ul style="list-style-type: none"> • Riesgos del proyecto: Los requerimientos son cambiantes así que no es fácil determinar alcance o costos realistas al inicio del proyecto. • Riesgos técnicos: La creación de drivers a la medida para los dispositivos y el ensamble de estos puede implicar restricciones dobles entre los tiempos especificados y la operación de los componentes, implicando reprocesos y afinamientos costosos en tiempo y recursos.
Usos conocidos	A partir del primer Vodafone Pilotentwicklung GmbH y ahora la nueva colmena de la empresa P21 GmbH está utilizando el patrón de proceso descrito dentro de los esfuerzos de desarrollo. P21 está ahora concentrado en diseñar otros componentes físicos del sistema de procesamiento de combustible.
Ver también	Diseño basado en modelos

Atributo	
Nombre	Conjunto de pruebas de <i>Bootstrapping</i> Tomado de (Bergner & Rausch, 2002)
También conocido como	-
Autor	Equipo SDPP02
Propósito	Validar la exactitud de una aplicación de negocios centrada en los datos mediante la construcción de un conjunto de pruebas de regresión. Usar <i>snapshots</i> (instantáneas) de bases de datos completas como base para la entrada inicial, el resultado y el resultado esperado de cada caso de prueba con el fin de descartar los efectos secundarios no deseados. Minimizar el esfuerzo para crear las <i>snapshots</i> de base de datos necesarias mediante la reutilización de las <i>snapshots</i> de resultados de casos de prueba como <i>snapshots</i> iniciales de entrada para otros casos de prueba.
Problema	<p>La exactitud y la consistencia de los datos gestionados por una aplicación de negocios suelen ser de suma importancia para la empresa en cuestión. Por ejemplo, un banco con un sistema bancario que involuntariamente pierde dinero en las cuentas de vez en cuando, estaría quebrado muy pronto. Por lo tanto, es esencial para realizar un número suficiente de casos de prueba durante el desarrollo de una aplicación de negocios, con el fin de garantizar la exactitud y robustez requerida. A medida que el número de casos de prueba necesarios suele ser muy alta, y a medida que los casos de prueba tienen que ser ejecutada muchas veces durante el desarrollo, un centro de pruebas de regresión automatizado es indispensable.</p> <p>Para asegurar la reproducibilidad de un caso de prueba de regresión, el sistema primero tiene que ser inicializado con un estado inicial del sistema claramente definido. A continuación, el escenario de prueba - se ejecuta - una secuencia de interacciones de usuario. Por último, el estado del sistema resultante tiene que ser comparado con el estado esperado del sistema. Para las aplicaciones de negocio centradas en datos, todos estos estados del sistema - el inicial, el resultado y el estado del sistema resultado esperado - deben incluir una <i>snapshot</i> de base de datos completa. Esto es necesario para descartar efectos secundarios no deseados que no se pueden detectar recurriendo sólo a los resultados observables de las operaciones ejecutadas por el caso de prueba.</p> <p>Por lo tanto, para crear un caso de prueba de regresión centrada en los datos que esté listo para ser utilizado para las pruebas, una <i>snapshot</i> de base de datos inicial y una <i>snapshot</i> de base resultado esperado tienen que ser creados. La elaboración y el mantenimiento de estas <i>snapshots</i> para una aplicación empresarial grande con algunos miles de casos de prueba es una tarea dolorosa y costosa.</p>

Contexto	<p>Como resultado del análisis del dominio de negocio de una aplicación, la funcionalidad considerada es capturada en forma de casos de uso. Con base a ellos, los correspondientes artefactos de trabajo de diseño e implementación pueden ser elaborados. Del mismo modo, los casos de uso se pueden utilizar para especificar e implementar casos de prueba del sistema.</p> <p>Los documentos de especificación de casos de prueba están estructurados jerárquicamente: Cada caso de prueba puede tener algunos casos de prueba sucesores que podrán ser usados después de su terminación exitosa. Para que esto sea posible, la <i>snapshot</i> de base de datos especificada por el documento de especificación de datos inicial de cada caso de prueba sucesor tiene que ser equivalente a la <i>snapshot</i> de base de datos especificada por el documento de especificación de datos de resultados esperada del caso de prueba predecesor correspondiente.</p> <p>Variables:</p> <ul style="list-style-type: none"> • Número de riesgos: NA • Estabilidad y claridad de requerimientos: Claramente definidos y estables, preferiblemente para minimizar el esfuerzo en cambios de <i>snapshot</i>. • Experiencia del equipo de desarrollo: Alta experiencia, específica para usar este tipo de pruebas y configurar los <i>snapshots</i>. • Flexibilidad al cambio en el proyecto: Baja flexibilidad, los cambios en <i>snapshots</i> programados como inicio de otros casos de pruebas crean unas dependencias fuertes que implican retrabajos en caso de requerir cambios. • Involucramiento de usuarios: NA • Distribución geográfica del equipo: NA • Uso de fábricas de software: Desarrollo con dos fábricas de software, por lo general debe ser una fábrica que hace el desarrollo y otra con la experiencia en pruebas y generación de planes con <i>snapshots</i>.
Solución	<p>Para reducir al mínimo el esfuerzo para la creación de documentos de especificación de casos de prueba, una técnica de <i>bootstrapping</i> (Especificar los parámetros de inicio) se puede aplicar para generar los documentos de especificación de datos iniciales necesarios y los documentos de especificación de datos de resultados esperados. El diagrama de actividad UML en la Figura 13, muestra las actividades que se tienen que llevar a cabo para aplicar el patrón de proceso.</p>

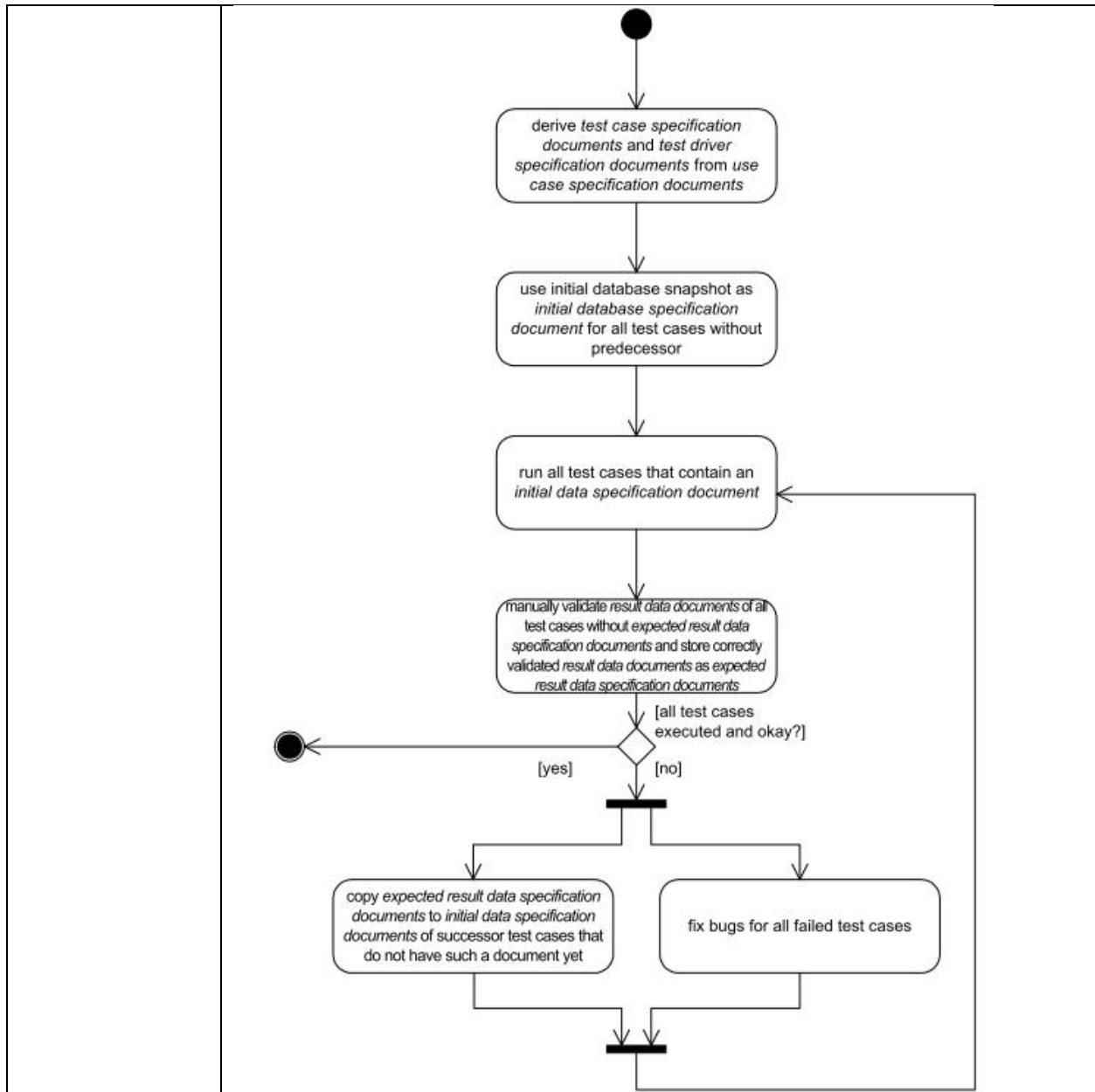


Figura 15 Proceso del patrón de proceso Conjunto de pruebas de Bootstrapping tomado de (Bergner & Rausch, 2002)

En primer lugar, los documentos de especificación de casos de prueba se elaboran. Contienen los correspondientes documentos de especificaciones piloto de pruebas, que pueden estar representados por las clases de casos de prueba JUnit Java, por ejemplo. Los casos de prueba sin un caso de prueba predecesor deben ser diseñados para ser ejecutados en un sistema recién instalado - su primer documento de especificación de datos define la *snapshot* de base de datos después de la instalación inicial del sistema.

En una segunda etapa, los documentos de especificación de datos iniciales de casos de prueba sin un caso de prueba predecesor se crean

	<p>efectivamente. Como se dijo anteriormente, se refieren a la snapshot de base de datos después de la instalación inicial del sistema - por lo general, una base de datos vacía con un poco de información de configuración, pero no hay datos empresariales.</p> <p>Ahora, los casos de prueba que no sólo requieren el estado de base de datos de un sistema recién instalado se pueden ejecutar. Estos casos de prueba producen un documento de datos de resultados en forma de una <i>snapshot</i> de base de datos, que tiene que ser examinada y validada por el desarrollador manualmente. Si la prueba revela un error en la aplicación, tiene que ser arreglado y los casos de prueba correspondientes tienen que ser ejecutados de nuevo. De lo contrario, los documentos de datos de resultados de los casos de prueba se almacenan como documentos de especificación de datos de los resultados esperados, sirviendo como predicción de pruebas de regresión para pruebas futuras. Técnicamente, esto se puede realizar por la descarga de <i>snapshots</i> de base de datos a los archivos y la evaluación de la equivalencia de tales archivos por medio de una herramienta de diferencias especializada.</p> <p>Los documentos de especificación de datos de los resultados esperados de los casos de prueba ya considerados a continuación sirven como documentos de especificación de datos iniciales para sus casos de prueba sucesores. Estos casos de prueba sucesor se pueden ejecutar, por lo tanto, de forma recursiva se generan los documentos iniciales de especificación de datos de casos de prueba adicionales. Como puede observarse, este proceso puede llevarse a cabo hasta que los documentos de especificación de datos iniciales y los documentos de especificación de datos de resultados esperados se han creado para todos los casos de prueba.</p> <p>Los Modelos de ciclo de vida recomendados son los seriales, con facilidad para definir escenarios de pruebas como Cascada y Modelo-V.</p>
Consecuencias	<p>Beneficios:</p> <ul style="list-style-type: none"> • Hace que sea posible la detección de efectos secundarios no deseados, recurriendo a las <i>snapshots</i> de base de datos como base para las entradas de prueba iniciales, así como los resultados reales y esperados de prueba. • Facilita la elaboración de un conjunto de pruebas de regresión, reduciendo al mínimo el esfuerzo necesario para la creación de <i>snapshots</i> iniciales y esperados de base de datos de resultados. • Proporciona una guía para la estructuración de un conjunto de pruebas por medio de las relaciones entre los sucesores de los casos de prueba. • Se lleva a completar los casos de prueba que se pueden ejecutar independiente o como un conjunto. <p>Desventajas</p>

	<ul style="list-style-type: none"> • El desarrollador debe tener cuidado de no olvidar los casos de prueba importantes que no se pueden agregar como sucesores a los casos de prueba ya existente. • Inicialmente, el desarrollador tiene que realizar los casos de prueba en el orden dado por la relación sucesores. • Si el esquema de base de datos cambia, el desarrollador debe adaptarse y volver a ejecutar todos los casos de prueba correspondientes. <p>Riesgos:</p> <ul style="list-style-type: none"> • Riesgos técnicos: Es una tecnología específica y requiere conocimientos y tecnología para lograrse. No es fácil encontrar las diferencias entre <i>snapshots</i>, se sugiere el uso de comparación de archivos, pero en casos de prueba con bastantes cambios puede ser complejo.
Usos conocidos	<p>El concepto de usar <i>snapshots</i> como la base para la entrada de prueba y resultados, se practica en muchas empresas de desarrollo (ejemplos alemanes conocidos por los autores incluyen la casa de software Healy Hudson AG, HypoVereinsbank AG, sd & m AG, y 4Soft GmbH). Un artículo sobre el entorno de prueba GOAL describe el uso de casos de prueba basados en datos combinados con reinicialización del conjunto de pruebas en Healy Hudson AG, un proveedor de software de adquisición alemán (cf Thomas Bonfig, Rainer Frömming, Andreas Rausch: Meta - Eine Tessa tinfrastuktur für unternehmensweite Anwendungen, OBJEKTspektrum 4/2000). El marco de pruebas correspondiente se ha desarrollado, integrado en el marco de pruebas JUnit, y se aplica en algunos proyectos en el desarrollo de la empresa alemana 4Soft GmbH.</p>
Ver también	-

Atributo	
Nombre	PIP: Patrón de Implementación progresiva Tomado de (Soares & Borba, 2002)
También conocido como	-
Autor	Soares, S. & Borba, P.
Propósito	Reducir la complejidad y mejorar la productividad de desarrollo. Reducir el impacto causado por los cambios de requisitos durante el desarrollo.
Problema	<p>Es difícil y costoso para validar y probar un sistema concurrente, distribuido y persistente. Por otra parte, la validación del sistema por lo general sólo se puede hacer más tarde en la fase de desarrollo. Este retraso para validar los requisitos del sistema aumenta los costos de corregir los errores detectados, ya que los desarrolladores pueden dedicar un esfuerzo considerable para poner en práctica los requisitos no funcionales a la aplicación incorrecta de los servicios del sistema.</p> <p>Para implementar un sistema persistente, distribuido y concurrente, PIP equilibra las fuerzas siguientes:</p> <ul style="list-style-type: none"> • Validación temprana de requisitos funcionales. Esto reduce el costo cambios y evita retrasos en el cronograma del proyecto. • Simplificar las pruebas probando cada aspecto (persistencia, distribución y control de concurrencia) por separado. Esta separación permite probar la versión funcional del sistema sin el impacto de la base de datos, red o errores de entorno concurrente. De hecho, cada requisito no funcional también se desarrollará y probará gradualmente, lo que evita que los errores de un aspecto afecten las pruebas de otro. • Transparencia de almacenamiento de datos. Esto es crucial para proporcionar inicialmente una versión no persistente del sistema con el fin de validar los requisitos funcionales, sin implementar persistencia. Después de eso, el sistema evoluciona a una versión persistente. • Independencia de las API de comunicación y middleware. Al igual que en el aspecto de persistencia, en una primera versión del sistema no existe un código de distribución, con el fin de permitir la validación temprana de requisitos funcionales. Sin embargo, el sistema debe evolucionar a una versión distribuida, sin afectar a los requisitos ya aplicados.
Contexto	<p>En el desarrollo de un sistema persistente, distribuido y concurrente, la implementación y pruebas son por lo general difíciles. Durante las pruebas, la base de datos, la distribución, concurrencia, y los errores funcionales pueden aparecer al mismo tiempo, incrementando la complejidad de depuración. Al utilizar EJB como la tecnología de persistencia y distribución, el tiempo de despliegue podría incrementarse. Para corregir los errores (incluyendo funcionales, de persistencia y errores de control de concurrencia) podríamos perder mucho tiempo compilando el código y desplegándolo en el servidor de aplicaciones. Podríamos tener que escribir</p>

	<p>programas específicos, para comprobar si los datos almacenados en la base de datos se ajustan a los resultados esperados. Del mismo modo, si el sistema se puede acceder al mismo tiempo, los programadores deben preocuparse por las ejecuciones concurrentes en la aplicación de los requisitos funcionales, incrementando la complejidad de programación.</p> <p>Variables:</p> <ul style="list-style-type: none"> • Número de riesgos: Alto, ayuda a mitigarlos • Estabilidad y claridad de requerimientos: Vagamente definidos o inestables. • Experiencia del equipo de desarrollo: Alta experiencia, es necesaria para la reconstrucción de los artefactos de software. • La flexibilidad al cambio en el proyecto: Alta flexibilidad, es necesario que el proyecto se adapte a los cambios propios del patrón, y tenga la disposición de aumentar el tiempo, para disminuir el riesgo de calidad. • Involucramiento de usuarios: NA • Distribución geográfica del equipo: NA • Uso de fábricas de software: NA
<p>Solución</p>	<p>Con el fin de resolver el problema mencionado, tenemos que implementar los requisitos funcionales, de persistencia, de distribución y control de concurrencia, de una manera progresiva. De hecho, debemos aplicar primero los requisitos funcionales, la interfaz de usuario, y el almacenamiento no permanente, con el fin de proporcionar un prototipo completamente funcional, y luego poner en práctica los otros aspectos, como se ilustra en la Figura 14. Aunque la figura sugiere un orden para la ejecución y prueba de los requisitos no funcionales, esto no es exigido por el patrón de proceso. De hecho, PIP sólo requiere que los diferentes aspectos a ser implementadas y probados se hagan de manera progresiva.</p> <div data-bbox="500 1249 1377 1465" data-label="Diagram"> </div> <p><i>Figura 16 Método de Implementación Progresiva Tomado de (Soares & Borba, 2002)</i></p> <p>Al extraer inicialmente del código los requisitos no funcionales, los desarrolladores pueden, por ejemplo, desarrollar rápidamente y probar los prototipos locales secuenciales y no persistentes, útiles para la captura y la validación de los requisitos del usuario. Como los requisitos funcionales se entienden bien y se estabilizan, esos prototipos se utilizan para obtener la versión final de la aplicación, mediante la implementación y prueba de persistencia, distribución y código de control de concurrencia gradualmente.</p>

	<p>Con el fin de apoyar esta aplicación progresiva, la separación de principios se debe aplicar durante las actividades de diseño. La arquitectura de software modular debe soportar el direccionamiento de los aspectos funcionales y no funcionales durante las actividades de codificación. Esto se puede lograr mediante el uso de patrones de arquitectura y diseño específicos.</p> <p>Alternativamente, esta separación de aspectos se puede lograr mediante el uso de programación orientada a aspectos. Por ejemplo, podríamos separar la persistencia, la distribución y los aspectos de control de concurrencia del código de negocio, mediante el uso de AOSD, una metodología orientada a aspectos, y mezclarlos con el prototipo funcional en una aplicación distribuida y persistente correcta.</p> <p>Los modelos de ciclo de vida recomendables son los cíclicos, como el modelo en espiral, RUP, o los modelos ágiles, como Scrum o XP.</p>
Consecuencias	<p>Beneficios:</p> <ul style="list-style-type: none"> • Productividad incrementada. Debido a la temprana validación de los requisitos funcionales y la simplificación de las pruebas, se incrementa la productividad del desarrollo. Los datos de un caso de estudio simple muestran que este incremento es aproximadamente un 10% y hubo una reducción del 50% de los esfuerzos por cambio de requerimientos. Estas cifras pueden ser mayores adicionando generación de código. • Las pruebas y depuración son más fáciles. PIP, naturalmente, ayuda a hacer frente a la complejidad inherente a las aplicaciones distribuidas y persistentes, al permitir que la prueba gradual de las diferentes versiones intermedias de la aplicación, lo que beneficia a la corrección del sistema. • Prototipado funcional temprano. En el mismo caso de estudio sencillo que se ha mencionado anteriormente, hay otros datos que muestran que el prototipo funcional se obtiene el 30% más temprano mediante el uso de un enfoque progresivo. <p>Desventajas:</p> <ul style="list-style-type: none"> • Reducción de la motivación del equipo. Los programadores pueden sentir que están generando más código del que es necesario, por ejemplo, por primera generan versiones no persistentes de las clases de almacenamiento de datos y luego sus versiones persistentes. Para evitar esto, el equipo de desarrollo debe estar convencido de los beneficios. • Pruebas de funcionamiento limitado. El enfoque progresivo no permite poner a prueba las situaciones donde las transacciones se deshacen, con el prototipo funcional. • Clases adicionales. Al implementar la persistencia debemos crear clases para almacenar objetos en un medio persistente. Sin embargo, con el fin de implementar el prototipo funcional, antes de implementar la persistencia, tenemos que crear clases para almacenar los objetos en una estructura no-persistente. Esto afecta a la productividad, ya que los

	<p>programadores deben implementar dos clases para almacenar instancias de un objeto. Herramientas de generación de código podrían resolver este inconveniente proporcionando automáticamente parte del desarrollo de las clases de almacenamiento de datos no persistentes y persistentes. De hecho, incluso en un enfoque no progresivo, algunas clases de almacenamiento no persistentes deben generarse para recuperar datos en respuesta a búsquedas del sistema.</p> <ul style="list-style-type: none"> • Modificaciones adicionales de las clases. Para implementar los requisitos funcionales, las clases suelen ser modificadas varias veces. Cuando se utiliza el enfoque de aplicación progresiva este número aumenta, ya que algunas clases deben modificarse para implementar la persistencia, a continuación, la distribución, y, por último, el control de concurrencia, disminuyendo la productividad. <p>Riesgos:</p> <ul style="list-style-type: none"> • Riesgos de proyecto: Aunque la aplicación del patrón puede mejorar la capacidad de prueba y disminución de errores al enfocarse en los aspectos funcionales primero, la repetición de pruebas puede hacer que el proyecto esté en riesgo al aumentar una de las restricciones básicas, lo que puede implicar mayores inversiones económicas para lograr el alcance. • Riesgos técnicos: Implementar requerimientos no funcionales tarde en el proyecto hace que se agregue un riesgo técnico y provoca la reescritura de código y adición de pruebas. En un símil, es como construir un edificio y al finalizarlo, introducir las tuberías o los parqueaderos del subsuelo, es posible, pero implica riesgos adicionales. • Riesgos de Negocio: Al aumentar el tiempo del proyecto se genera el riesgo de construir un software que al ser liberado ya no cumple con las necesidades del negocio.
<p>Usos conocidos</p>	<p>Los autores enumeran diversos sistemas desarrollados con este patrón: sistema de clientes de una empresa de telecomunicaciones, sistema de registro de quejas para una empresa de salud y diversos proyectos académicos.</p>
<p>Ver también</p>	<p>Desarrollo guiado por casos de uso: es uno de los modelos más usados con PIP junto con RUP. PDC: Persistent data collection, patrón para manejo de persistencia que independiza las capas y hace más sencillo introducir los cambios. DAP: Distributed Adapters Pattern, permite separar las interfaces de componentes para desacoplar la aplicación y reducir impactos por cambios. PDA: Pattern for Distribution Aspects, provee una estructura para distribución de código. Concurrency Manager: Este patrón proporciona una alternativa a la sincronización de métodos con el objetivo de aumentar el rendimiento del sistema.</p>

Atributo	
Nombre	Incremental. Tomado de (United States of America Department of Defense, 1994) y (IBM Corporation, 2005)
También conocido como	NA
Autor	Departamento de Defensa de los Estados Unidos de América
Propósito	Desarrollo de software con una especificación inicial, pero con un avance incremental hacia la solución requerida.
Problema	Al tener una especificación inicial se requiere un esfuerzo bastante fuerte en esta primera etapa, para que el software deseado sea implementado en un solo ciclo como se plantea en el Modelo Cascada.
Contexto	<p>La estrategia incremental determina las necesidades del usuario y define los requisitos del sistema y, a continuación, realiza el resto del desarrollo en una secuencia de compilaciones. La primera compilación incorpora partes de las funciones planificadas, la siguiente compilación añade más funciones y así sucesivamente hasta que se completa el sistema.</p> <p>Variables:</p> <ul style="list-style-type: none"> • Número de riesgos: Bajo o medio. • Estabilidad y claridad de requerimientos: Vagamente definidos o inestables. Su acercamiento segmentado es una ventaja para esta clase de proyectos. Aunque se espera que el usuario final sepa lo que desea. • Experiencia del equipo de desarrollo: Alta experiencia • La flexibilidad al cambio en el proyecto: Baja flexibilidad • Involucramiento de usuarios: Alto involucramiento. • Distribución geográfica del equipo: Equipo local • Uso de fábricas de software: Desarrollo con una fábrica de software, más de una fábrica es un riesgo de comunicación adicional que pone en peligro la aplicación de este patrón.
Solución	<p>Generar un desarrollo en iteraciones sucesivas con resultados parciales hasta la generación de la implementación total.</p> <p>Las iteraciones siguientes son características:</p> <ul style="list-style-type: none"> • Una breve iteración inicial para establecer el ámbito y la visión, y para definir el caso de negocio • Una única iteración de elaboración, durante la que se definen los requisitos y se establece la arquitectura • Varias iteraciones de construcción, durante las que se ejecutan los casos de uso y se implementa la arquitectura • Varias iteraciones de transición para migrar el producto a la comunidad de usuarios <p>En la figura a continuación, se puede ver el proceso planteado con sus iteraciones.</p>

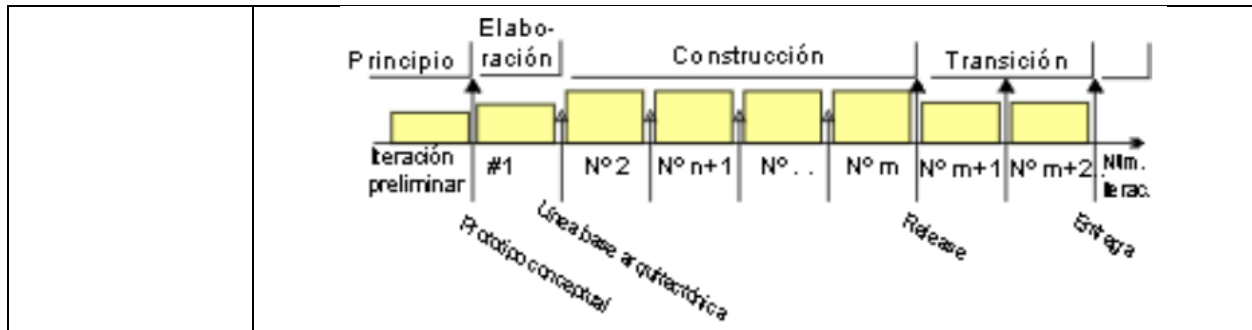


Figura 17 Proceso incremental, Tomado de (IBM Corporation, 2005)

Los modelos de ciclo de vida apropiados para este modelo son RUP y el Modelo en Espiral, con iteraciones determinables desde el inicio, en las metodologías ágiles no es claro cómo podrían implementarse las dos primeras iteraciones.

<p>Consecuencias</p>	<p>Beneficios:</p> <ul style="list-style-type: none"> • Desarrollo con aproximación gradual al resultado que permite validar la funcionalidad • Se generan entregas parciales que permiten tener funcionalidad antes de la entrega final. • Entendimiento de los requerimientos parcial por iteración, lo cual permite tener mejor conocimiento de la parte específica a desarrollar y luego preocuparse del resto de la especificación. <p>Desventajas:</p> <ul style="list-style-type: none"> • Para desarrollos relativamente pequeños representa un sobrecosto el uso de varias iteraciones. • Sin la participación de las personas de negocio, la diferenciación con el modelo Cascada no es tan notoria. <p>Riesgos:</p> <ul style="list-style-type: none"> • Riesgos de proyecto: Al desarrollar iteraciones el plan puede incrementarse y por lo tanto aumentar costos y tiempo. Siempre existe el riesgo de hacer retrabajos si se hacen varias liberaciones, ya que algunas veces el código que funcionó para una etapa inicial puede no ser útil para etapas posteriores.
<p>Usos conocidos</p>	<p>Toda clase de desarrollos se han elaborado con los métodos evolutivos.</p>
<p>Ver también</p>	<p>Modelo en espiral: permite abordar los problemas de forma incremental, en su versión más reciente aumenta el compromiso con cada ciclo.</p>

Atributo	
Nombre	Evolutivo Tomado de (United States of America Department of Defense, 1994) y (IBM Corporation, 2005)
También conocido como	Pilot development (The US Department of Justice, 2003)
Autor	Departamento de Defensa de los Estados Unidos de América
Propósito	Desarrollar el software sin tener un conocimiento definitivo de las necesidades del cliente.
Problema	En ocasiones el cliente puede reconocer que no conoce exactamente lo que necesita o quiere, pero cuando lo haya utilizado o visto lo podrá reconocer, es por esto que este patrón plantea un descubrimiento evolutivo de las necesidades.
Contexto	<p>La estrategia evolutiva difiere de la incremental en el reconocimiento de que las necesidades del usuario no se comprenden por completo, y que no pueden definirse todos los requisitos de entrada, sino que se van redefiniendo en cada compilación sucesiva.</p> <p>Variables:</p> <ul style="list-style-type: none"> • Número de riesgos: Bajo o medio. • Estabilidad y claridad de requerimientos: Vagamente definidos o inestables. No es necesario que el cliente tenga una idea exacta de lo que quiere lograr con el software. • Experiencia del equipo de desarrollo: Baja experiencia, el equipo tiene la oportunidad de aprender gradualmente la plataforma tecnológica. • La flexibilidad al cambio en el proyecto: Baja flexibilidad • Involucramiento de usuarios: Alto involucramiento, es necesario para determinar la funcionalidad final del software. • Distribución geográfica del equipo: NA. Es posible trabajar con un equipo remoto con apoyo de medios electrónicos, el avanzar gradualmente permite que la distancia no sea un riesgo insuperable. • Uso de fábricas de software: Desarrollo con una fábrica de software, más de una fábrica es un riesgo de comunicación adicional que pone en peligro la aplicación de este patrón.
Solución	<p>Generar aproximaciones a las necesidades del usuario para acercarse al software final mediante liberaciones sucesivas.</p> <p>Las iteraciones siguientes son características:</p> <ul style="list-style-type: none"> • Una breve iteración inicial para establecer el ámbito y la visión, y para definir el caso de negocio. • Varias iteraciones de elaboración, durante las que se perfeccionan los requisitos en cada iteración • Una única iteración de construcción, durante las que se ejecutan los guiones de uso y se amplía la arquitectura • Varias iteraciones de transición para migrar el producto a la comunidad de usuarios

En la figura 16 a continuación, se puede ver el proceso planteado con sus iteraciones.

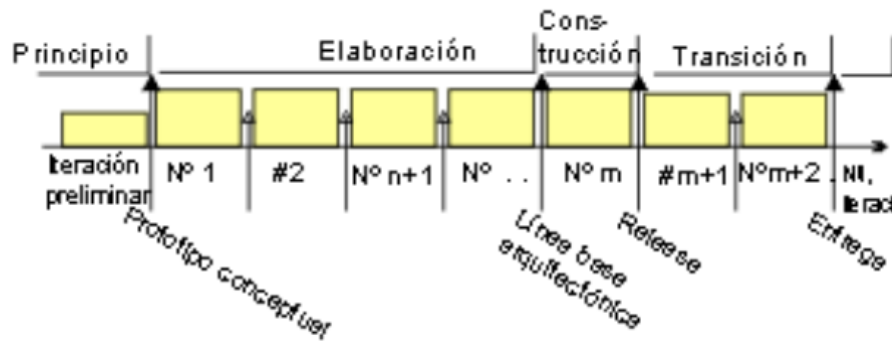


Figura 18 Patrón evolutivo, tomado de (IBM Corporation, 2005)

En Pilot development se adiciona una fase de pruebas y generación de pilotos para determinar realmente lo que quiere el usuario final.

Initiation Through Development Phases (Sequential or RAD)	█								
Test Phase		█							
Subsystem Integration Test		█							
System Test		█							
User Acceptance Test				█					
Pre-Pilot			█						
Pilot				█					
Evaluation					█				
Additional Development								█	
Additional Testing								█	
Implementation Phase									█
Operations & Maintenance Phase									█
Disposition Phase									█

Figura 19 Modelo planteado por el Departamento de Justicia (2003)

Los modelos de ciclo de vida apropiados para este modelo son RUP y el Modelo en Espiral. En los modelos ágiles el levantamiento de requerimientos no es tan fuerte por lo cual, aunque la filosofía es compatible, no es fácil hacer compatibles los ciclos de elaboración con los sprint o ciclos de desarrollo.

Consecuencias

Beneficios:

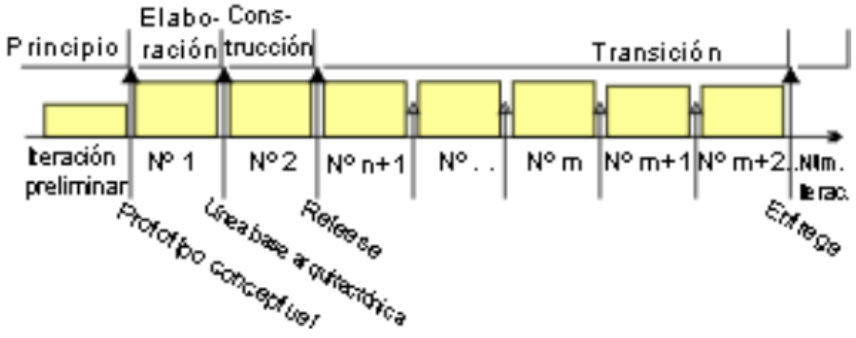
- Definición progresiva de los requerimientos
- Avance por parte de los desarrolladores hacia un conocimiento más específico de las herramientas de desarrollo

Desventajas:

- Avance incierto hacia la solución sin un marco de tiempo conocido.

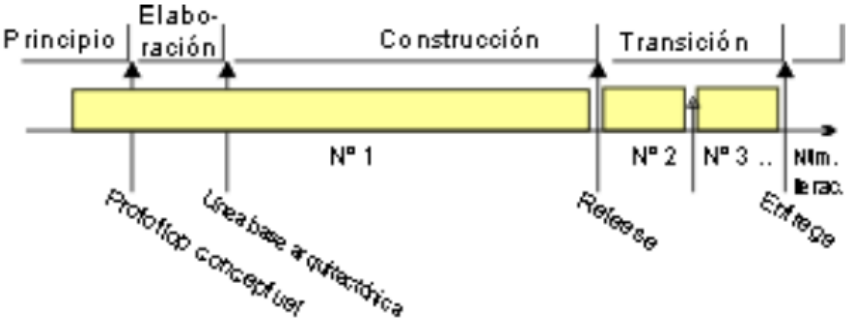
	<ul style="list-style-type: none"> • Es complejo de estimar o medir desde el inicio <p>Riesgos:</p> <ul style="list-style-type: none"> • Riesgo de proyecto: El proyecto se puede prolongar por un tiempo inestimable al inicio. • Riesgo del negocio: Perder compromiso de presupuesto o personal, si no se obtienen resultados durante las iteraciones de transición se arriesga el perder patrocinio. Si los requerimientos se hacen muy complejos se puede volver un proyecto poco deseable para los desarrolladores.
Usos conocidos	Desarrollo de nuevos productos, el uso de metodologías de innovación se pueden ver beneficiadas por el uso de este patrón.
Ver también	NA

Atributo	
Nombre	Entrega incremental Tomado de (IBM Corporation, 2005)
También conocido como	NA
Autor	Gilb, T. (1988)
Propósito	Generar un producto con una especificación inicial en un tiempo muy corto y tener la capacidad de entregar mejoras después de la salida inicial.
Problema	Es necesario crear un producto con fuertes presiones de tiempo, donde el riesgo de no tener un producto: "Ahora o nunca", implica un fracaso total del proyecto.
Contexto	<p>Algunos autores también establecen fases en las entregas de funcionalidades incrementales al cliente (Gilb, 1988) Esto puede ser necesario en casos de fuertes presiones con los plazos de comercialización, en los que la entrega temprana de ciertas funciones clave puede suponer beneficios comerciales significativos.</p> <p>En términos del enfoque de iteración y fase, la fase de transición va al principio y es la que más iteraciones tiene. Esta estrategia precisa de una arquitectura muy estable, que es difícil de adquirir en un ciclo de desarrollo inicial, para un sistema "sin precedentes".</p> <p>Variables:</p> <ul style="list-style-type: none"> • Número de riesgos: Bajo o medio. • Estabilidad y claridad de requerimientos: Claramente definidos y estables. El dominio del problema es familiar, la arquitectura y los requisitos pueden estabilizarse al principio del ciclo de desarrollo. el problema tiene un grado bajo de novedad • Experiencia del equipo de desarrollo: Alta experiencia. • La flexibilidad al cambio en el proyecto: Baja flexibilidad. Específicamente en los hitos del proyecto. • Involucramiento de usuarios: Bajo o Normal Involucramiento.

	<ul style="list-style-type: none"> • Distribución geográfica del equipo: NA. En grandes empresas de desarrollo es común manejar equipos distribuidos geográficamente. • Uso de fábricas de software: Una o más fábricas de software.
<p>Solución</p>	<p>Entregar un producto terminado rápidamente y hacer solución de problemas de forma incremental en iteraciones de transición.</p> <p>Las iteraciones siguientes son características:</p> <ul style="list-style-type: none"> • Una breve iteración inicial para establecer el ámbito y la visión, y para definir el caso de negocio • Una única iteración de la elaboración, durante la que se establece la línea base de una arquitectura estable • Una única iteración de construcción, durante la que se ejecutan los casos de uso y se sustancia la arquitectura • Varias iteraciones de transición para migrar el producto a la comunidad de usuarios  <p><i>Figura 20 Modelo propuesto para el patrón Entrega Incremental, tomado de (IBM Corporation, 2005)</i></p> <p>Los modelos de ciclo de vida apropiados para este patrón son los cíclicos como Espiral y RUP, pero los ágiles como Scrum o XP servirían perfectamente para lograr resultados en producción rápidamente y hacer un aporte incremental a funcionalidad o calidad al software en producción.</p>
<p>Consecuencias</p>	<p>Beneficios:</p> <ul style="list-style-type: none"> • Se puede entregar software en los tiempos esperados por el mercado o los clientes. • Por lo general es el incremento a una plataforma previamente establecida, por lo cual el riesgo se minimiza <p>Desventajas</p> <ul style="list-style-type: none"> • Es posible que los problemas de calidad se transmitan al cliente y termine asumiendo un papel de probador. • Se pueden generar problemas de producción para el negocio por los problemas de calidad del software que un proceso de calidad corto no logre detectar y solucionar. • La reputación de la empresa que produce el software se puede ver afectada si son demasiados los problemas que llegan a las primeras iteraciones de transición.

	<p>Riesgos</p> <ul style="list-style-type: none"> • Riesgos del proyecto: gran parte de las limitaciones del proyecto llegan hasta la primera entrega con la iteración de transición inicial, tiene una restricción de tiempo y su materialización tiene un alto impacto para la empresa cliente (<i>Stakeholder</i> patrocinador) y la desarrolladora. • Riesgos técnicos: El corto tiempo de pruebas puede significar un problema de calidad importante. • Riesgos de negocio: Los riesgos de negocio más probable es construir un software que ya no encaja en las necesidades específicas del negocio, determinado por el marco de tiempo, por ejemplo, si sale una nueva tecnología y el producto para usar esta tecnología, sale al mercado cuando ya hay un producto de la competencia con el tiempo de posicionarse y ejercer dominio.
Usos conocidos	<p>Con la mercantilización del software (Thomas, 2010), las empresas multinacionales de software se ven presionadas para generar versiones de productos con unos márgenes de tiempos muy reducidos, ya que hay otros gigantes de la industria listos a tomar su mercado. Es común el uso de este patrón que hacen Microsoft, IBM u Oracle, los cuales al liberar la primera versión de cualquier producto tienden a descargar en los usuarios parte de los costos de pruebas, que puede ser un riesgo para la implementación de proyectos COTS.</p>
Ver también	NA

Atributo	
Nombre	Diseño grande, Tomado de (IBM Corporation, 2005)
También conocido como	Esfuerzo reducido y RAD
Autor	Departamento de Defensa de los Estados Unidos de América
Propósito	Crear un desarrollo de software a partir de una especificación y un desarrollo previo.
Problema	Se desea hacer un pequeño incremento de funcionalidad bien definida a un producto muy estable.
Contexto	<p>El enfoque de cascada tradicional puede verse como un caso degenerado en el que sólo hay una iteración en la fase de transición. Se denomina "diseño grande"(United States of America Department of Defense, 1994). En la práctica, es difícil evitar las iteraciones adicionales en la fase de transición.</p> <p>Variables:</p> <ul style="list-style-type: none"> • Número de riesgos: Bajo o medio. • Estabilidad y claridad de requerimientos: Claramente definidos y estables.

	<ul style="list-style-type: none"> • Experiencia del equipo de desarrollo: Alta experiencia, tanto en el dominio del problema como con el producto existente • La flexibilidad al cambio en el proyecto: Baja flexibilidad • Involucramiento de usuarios: NA. • Distribución geográfica del equipo: NA. • Uso de fábricas de software: NA.
<p>Solución</p>	<p>Generar un ciclo de vida que permita adicionar la funcionalidad en sucesivas transiciones.</p> <p>Las iteraciones siguientes son características:</p> <ul style="list-style-type: none"> • Una breve iteración inicial para establecer el ámbito y la visión, y para definir el caso de negocio • Una iteración de construcción única y muy larga, durante la que se ejecutan los casos de uso y se sustancia la arquitectura • Varias iteraciones de transición para migrar el producto a la comunidad de usuarios  <p><i>Figura 21 Estructura de Gran diseño, tomado de (IBM Corporation, 2005)</i></p> <p>Todos los modelos de ciclo de vida son apropiados para este patrón, incluso Cascada con dos liberaciones.</p>
<p>Consecuencias</p>	<p>Beneficios:</p> <ul style="list-style-type: none"> • Permite controlar la adición de funcionalidad a un sistema funcional • Es un método controlado de desarrollar nueva funcionalidad <p>Desventajas:</p> <ul style="list-style-type: none"> • Puede verse envuelto en problemas por el reuso de software • Puede ser poco flexible a cambios en los requerimientos. <p>Riesgos:</p> <ul style="list-style-type: none"> • Riesgos técnicos: Se pueden presentar problemas por el reuso de software, si el cambio que se quiere introducir, por pequeño que sea, afecta la arquitectura o riñe con el diseño previamente establecido, es posible que se materialice un riesgo de alto impacto en el tiempo y costo del proyecto.
<p>Usos conocidos</p>	<p>Es el patrón de uso común en el mantenimiento de software.</p>
<p>Ver también</p>	<p>Patrón Reduced Effort y RAD Work pattern (The US Department of Justice, 2003)</p>

Atributo	
Nombre	Desarrollo evolutivo controlado (MED: Managed Evolutionary Development) Tomado de (The US Department of Justice, 2003)
También conocido como	NA
Autor	Departamento de Justicia de los Estados Unidos de América
Propósito	Controlar la adición de una funcionalidad con cambios sustanciales a un desarrollo previamente en funcionamiento.
Problema	Se tiene un sistema con una funcionalidad que va a ser modificada de forma importante, pero no se pueden determinar todos los cambios desde el inicio del proyecto.
Contexto	<p>El proceso de desarrollo basado en el MED combina una estrategia de desarrollo evolutivo con la entrega gradual. El desarrollo de sistemas con recursos del MED mediante la definición de una visión limitada de un futuro sistema y luego iterativamente el perfeccionamiento de los procesos de reingeniería de negocios, requisitos del sistema de información y arquitectura técnica. La estrategia de entrega incrementales dentro de MED se utiliza para encapsular parte del sistema general como un subsistema que será construido y desplegado. Los subsistemas se construyen cuando hay suficiente confianza de que van a proporcionar un conjunto coherente, y validado por el usuario de la funcionalidad del negocio. A medida que se obtiene experiencia de uso, las lecciones aprendidas se alimentan de nuevo en cada subsistema mediante la mejora de cada una de las versiones posteriores del sistema.</p> <p>Variables:</p> <ul style="list-style-type: none"> • Número de riesgos: Alto • Estabilidad y claridad de requerimientos: Vagamente definidos o inestables. • Experiencia del equipo de desarrollo: Alta experiencia • La flexibilidad al cambio en el proyecto: Alta flexibilidad • Involucramiento de usuarios: Alto involucramiento • Distribución geográfica del equipo: NA • Uso de fábricas de software: NA
Solución	MED es compatible con un ciclo iterativo, se compone de múltiples iteraciones de desarrollo y puesta en producción de implementación de subsistemas mediante la definición parcial de los requisitos funcionales y de datos a nivel de sistema y una arquitectura de sistema modular, lo que permite el refinamiento posterior, el desarrollo y la implementación de los subsistemas que pueden evolucionar para satisfacer las necesidades futuras del negocio. Con frecuencia, un nivel de liberación particular que contiene parcial, pero no completa, la funcionalidad se conoce como una "acumulación". Durante las fases de planificación y análisis de requerimientos, está prevista toda una serie de construcciones sucesivas de software, cada una de las cuales es diseñado, desarrollado, probado y puesto en práctica.

La fase de planificación es donde el director de proyecto, con la aprobación de los usuarios funcionales del sistema, determina los requerimientos funcionales que deben ser desarrollados en los diferentes subsistemas al estar relacionados. La fase de Análisis de Requerimientos se dividirá en fases para definir los requisitos del sistema y la estructura generales por subsistema. En la conclusión del análisis de cada subsistema comienza su propio ciclo de desarrollo para definir una arquitectura del subsistema objetivo. Cuando se utiliza el modelo de trabajo de MED, hay un hito explícito para los requerimientos de nivel de sistema y un hito para cada requerimiento de subsistema.

Las iteraciones propuestas son:

- Fases iniciales de Iniciación del proyecto y elaboración.
- Ciclos de construcción graduales para determinar las necesidades a implementar en el software.
- Se hacen fases finales para llevar el desarrollo al usuario final.

A continuación, se encuentra la figura 20, con el esquema de MED.

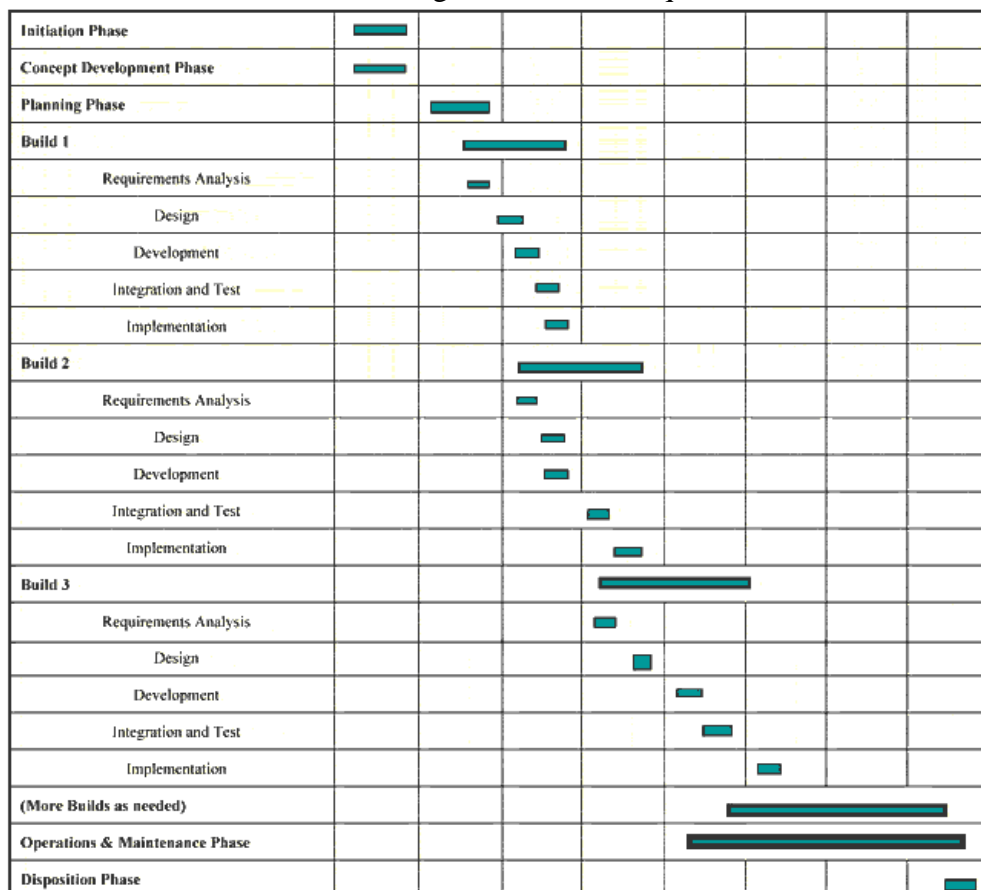


Figura 22 Estructura de iteraciones de ME. Tomado de (The US Department of Justice, 2003)

Los modelos de ciclo de vida que mejor se adaptan a este patrón son los iterativos Espiral, Rup, y los ágiles como Scrum y XP. Estos permiten

	generar la salida de software parcialmente y quedar enmarcados en un proyecto más largo.
Consecuencias	<p>Beneficios:</p> <ul style="list-style-type: none"> • División de un gran problema complejo en subetapas permitiendo que los cambios sobre los que se tiene mayor claridad o prioridad se aborden primero. • Se puede paralelizar el desarrollo de subsistemas si el código está adecuadamente modularizado. <p>Desventajas:</p> <ul style="list-style-type: none"> • El tener subproyectos dentro de un proyecto implica que la desviación en uno de estos puede afectar a los demás. • El tener submódulos en producción implica tener claramente divididos los equipos de soporte y de desarrollo. • Requiere una alta experiencia en gerencia de proyectos y proyección de recursos. <p>Riesgos:</p> <ul style="list-style-type: none"> • Riesgos de proyecto: al ser un proyecto de múltiples ciclos, es posible que el tiempo planeado se extienda, así como los costos. Se corre el riesgo de confundirse con un servicio de desarrollo de software. • Riesgos técnicos: Modificar la plataforma por los submódulos puede significar bastante retrabajo, aunque el enfoque de hacer los cambios por separado mitiga este riesgo. • Riesgos de comunicación: Tener diferentes submódulos puede implicar diferentes equipos de negocio pugnando por prioridad en el orden de desarrollo.
Usos conocidos	En el Departamento de Justicia de los Estados Unidos de América se determinó como una forma común de uso para el desarrollo de software.
Ver también	NA

Atributo	
Nombre	Fábrica de pruebas (Testing factory)
También conocido como	NA
Autor	Fuente propia
Propósito	Permitir que una empresa experta realice las pruebas del software antes de las pruebas de usuario final.
Problema	La empresa tiene experiencia en desarrollo de software, pero los desarrollos tienen una alta cantidad de defectos en producción, o se pretende que tengan mejor calidad antes de que sean verificados por los usuarios finales.
Contexto	<p>Para empresas que usan desarrollo interno o una fábrica de desarrollo el aseguramiento de la calidad se puede convertir en un requisito que tiende a comprimirse o desaparecer, se pretende que una fábrica con especialización en desarrollo de pruebas asegure la calidad del producto antes de las pruebas de usuario. El usar una fábrica u oficina de pruebas, permite que un tercero imparcial haga un control de calidad basado en los casos de uso o requerimientos especificados, por lo general, se puede establecer un mecanismo de “<i>Check and balance</i>” con los desarrolladores, ya que al ser un tercero independiente permite evaluar la calidad no sólo del desarrollo sino de las especificaciones, especialmente en un modelo de desarrollo de software como el Modelo en V.</p> <p>Las fábricas de pruebas generan los casos y escenarios de prueba basados en los requerimientos o casos de uso, y luego de aplicarlos sobre el software genera las evidencias de pruebas, las cuales son verificables en caso de dudas sobre el funcionamiento por parte de los usuarios finales.</p> <p>Variables:</p> <ul style="list-style-type: none"> • Número de riesgos: NA. es aplicable en cualquier cantidad de riesgos • Estabilidad y claridad de requerimientos: Claramente definidos y estables, se requiere que la especificación esté lo mejor definida posible, las variaciones o controles de cambios son costosos cuando también afectan casos de prueba. • Experiencia del equipo de desarrollo: Alta experiencia. Se recomienda en equipos especializados con buen manejo de la lógica del negocio. • La flexibilidad al cambio en el proyecto: Baja flexibilidad, por lo general se espera que las estimaciones de las dos fábricas sean sostenidas lo máximo posible. • Involucramiento de usuarios: Bajo o Normal Involucramiento, se espera que las especificaciones sean adecuadamente tomadas y que las verificaciones de calidad de la fábrica minimicen la interacción del usuario final.

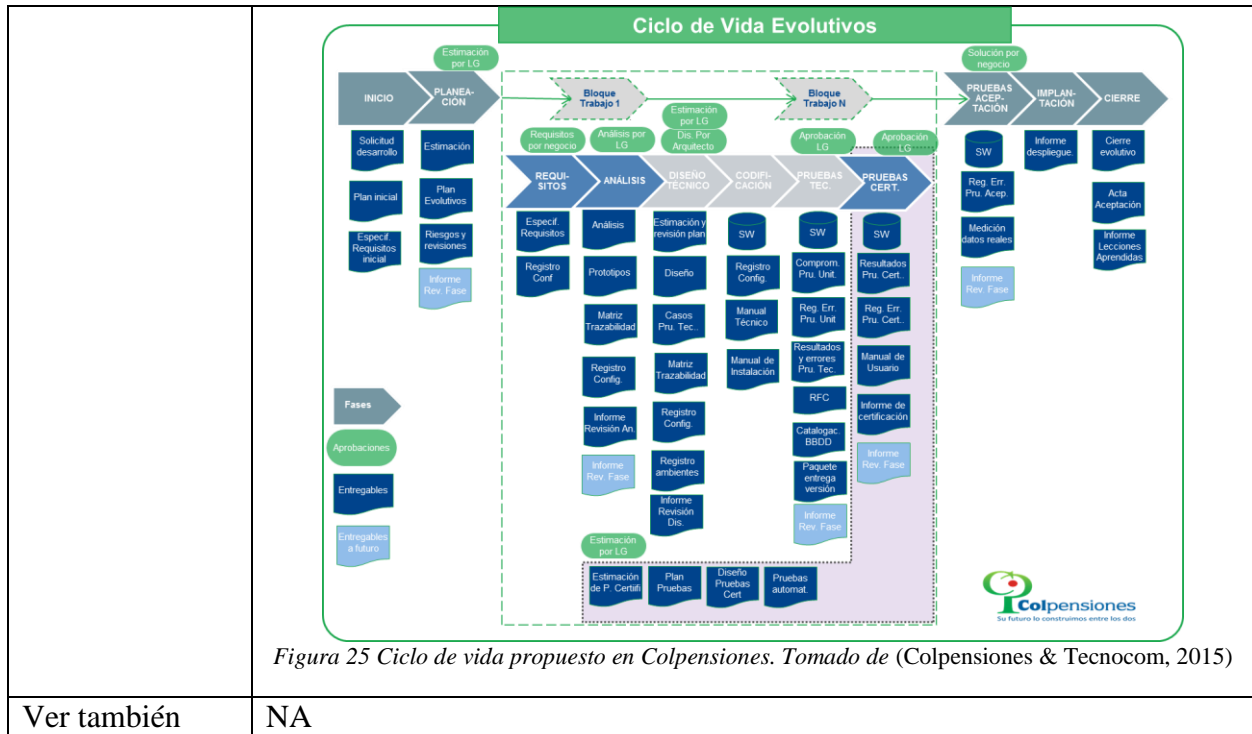
	<ul style="list-style-type: none"> • Distribución geográfica del equipo: NA, es posible trabajar con los dos esquemas. • Uso de fábricas de software: Desarrollo con dos fábricas de software, una de ellas especializada en pruebas.
<p>Solución</p>	<p>Tener una empresa especializada en pruebas que asegure la calidad del software y documentación, y sirva como un filtro necesario para evitar la propagación de defectos, tanto hacia las pruebas de usuario como el despliegue en producción. El tener un filtro separado de calidad ayuda a las empresas a minimizar los costosos defectos en producción, que pueden implicar riesgos operativos, jurídicos y reputacionales.</p> <p>La fábrica de pruebas como lo señala el documento de Infosys(Cooper-brown & Infosys, 2015) actúa como una función independiente en el SDLC y resuelve un problema común al tener una clara delimitación entre las funciones de crear y probar de una organización.</p> <p>Adicionalmente, la fábrica de pruebas se configura como una función de control de calidad centralizada que brinda la adopción de un proceso uniforme y un enfoque de control de calidad a nivel de la empresa con una gobernabilidad más sencilla. La fábrica de pruebas es también la configuración de los atributos de un conjunto más ágil, eficiente y repetible de procesos.</p> <p>La figura 21 muestra el proceso general, y aunque aparece como un Modelo serial, realmente es aplicable en cualquier Modelo de desarrollo. Las cajas en naranja son las labores de la fábrica de pruebas, validación de requerimientos y de los entregables después de la etapa de desarrollo.</p> <div data-bbox="483 1113 1383 1711" style="text-align: center;"> </div> <p style="text-align: center;"><i>Figura 23 Ciclo de desarrollo con Fábrica de pruebas. Fuente propia</i></p>
<p>Consecuencias</p>	<p>Beneficios:</p> <ul style="list-style-type: none"> • Separación de funciones con un grupo especializado en pruebas. • Minimizar riesgos por defectos no detectados en las pruebas

	<ul style="list-style-type: none"> • Mejora la gobernabilidad del proceso al separar la aceptación del producto • Ayuda a tener procesos más estandarizados • Validación de todos los entregables del proceso <p>Desventajas:</p> <ul style="list-style-type: none"> • Implica un costo adicional para el proceso • Si la fábrica de pruebas desconoce el negocio es posible aumentar el tiempo de pruebas y los reprocesos necesarios • Deficiencias en el levantamiento de requerimientos implican sobrecostos adicionales al modificar los casos de prueba <p>Riesgos:</p> <ul style="list-style-type: none"> • Riesgos de proyecto: Si se presentan problemas de calidad fuertes, es posible que las iteraciones de pruebas lleven a ser repetidas completamente y a tener sobrecostos y excesos de tiempo no planeado. • Riesgos de comunicación: Problemas de entendimiento de los defectos detectados con la fábrica de desarrollo, provocan exceso de tiempo para solucionar los problemas. Si el levantamiento de requerimientos no es preciso, se pueden producir retrabajos por generación de casos y escenarios de pruebas erróneos.
Usos conocidos	El uso de fábricas de software para pruebas se ha expandido por el mundo, con empresas tan poderosas como Infosys(2015), en Colombia empresas como Choucair (Choucair, n.d.), GreenSQA (Greensqa, n.d.), SQA S.A.(SQA S.A., n.d.) o Pixel Group Net (Pixel Group Net, n.d.), ofrecen servicios de fábrica de pruebas.
Ver también	NA

Atributo	
Nombre	Cremallera (Zipper)
También conocido como	NA
Autor	Fuente propia
Propósito	Especializar las etapas del proceso con una empresa responsable de usar las mejores prácticas y los recursos más experimentados para cada etapa.
Problema	Se tiene un proceso tercerizado, pero el control de etapas y entregables difícilmente se puede alcanzar sin la vigilancia de otros terceros.
Contexto	Se desea maximizar la fuerza de trabajo usando varias fábricas en el proceso, para utilizar la experiencia y cantidad de recursos calificados que pueden ofrecer, pero el control de los entregables por etapa exige tener un grupo de personas bastante grande y calificado para no quedar a merced de los proveedores. Se requiere una definición de artefactos, estándares de desarrollo y listas de chequeo por artefacto que deben seguirse para poder aceptar los entregables. Se deben establecer mesas de entendimiento donde

	<p>los requerimientos se discuten entre las fábricas de desarrollo, levantamiento y pruebas, y se hace una explicación completa hasta que el entendimiento de la porción de código a desarrollar es compartido por las tres partes, a partir de este entendimiento, se logra un consenso que permite crear el diseño y los casos de prueba.</p> <p>Variables:</p> <ul style="list-style-type: none"> • Número de riesgos: Alto. • Estabilidad y claridad de requerimientos: Vagamente definidos o inestables, la entrada al proceso son requerimientos de alto nivel para definirlos y estabilizarlos dentro del proceso. • Experiencia del equipo de desarrollo: Alta experiencia. Es una de las ganancias al usar una fábrica de software. • La flexibilidad al cambio en el proyecto: Baja flexibilidad. Los sobrecostos de retroceder limitan la flexibilidad. • Involucramiento de usuarios: Bajo o Normal Involucramiento. El uso de fábricas y control de las fábricas por medio de otros proveedores hace que el tiempo de recursos requerido baje. • Distribución geográfica del equipo: NA. La independencia de fábricas permite tener fábricas aisladas. • Uso de fábricas de software: Desarrollo con dos fábricas de software o más, hasta tres es posible: fábrica de requerimientos, fábrica de desarrollo y fábrica de pruebas. En el modelo implementado en Colpensiones la fábrica de requerimientos era la misma de pruebas, para limitar la cantidad de terceros que deben conocer el proyecto.
<p>Solución</p>	<p>Asignar a fábricas especializadas cada etapa del proceso para tener una evaluación rigurosa de los entregables por etapa. En la figura 22 se ve el proceso con etapas propias de la empresa que va a recibir el desarrollo en color azul, las etapas de una fábrica especializada en especificaciones y pruebas están en verde y la fábrica de desarrollo en naranja. La fábrica de desarrollo hace control de la calidad de las especificaciones, así como la fábrica de pruebas hace control de los entregables de la fábrica de desarrollo, entregando al final de las pruebas de fábrica un producto terminado con una calidad apropiada para hacer las pruebas de usuario, disminuyendo la cantidad de retrabajo y las devoluciones en el ciclo.</p> <div data-bbox="446 1522 1421 1648" style="text-align: center;"> </div> <p><i>Figura 24 Proceso de desarrollo Zipper. Fuente propia</i></p> <p>Los modelos apropiados para implementar son los seriales Cascada y Modelo-V, los cíclicos implican sobrecostos muy altos para un modelo donde la información y el control fluye entre fábricas de software.</p>
<p>Consecuencias</p>	<p>Beneficios:</p>

	<ul style="list-style-type: none"> • Uso de grupos de trabajos grandes que permiten paralelización de desarrollos. • Control de entregables entre terceros que aseguran la calidad del software en las pruebas de usuario final. • Posible distribución geográfica de las fábricas y uso de economías de escala. <p>Desventajas:</p> <ul style="list-style-type: none"> • Mayores costos en la devolución dentro del proceso • Si el desarrollo es muy largo o complejo es necesario hacer mesas de entendimiento muy largas, donde los costos se elevan. • Es posible que haya problemas de entendimiento de requerimientos haciendo necesarios reprocesos costosos. <p>Riesgos:</p> <ul style="list-style-type: none"> • Riesgos del proyecto: El pasar la información entre tantos interesados puede retrasar el proyecto. Si la calidad del desarrollo no es la mejor, se puede tomar la decisión de hacer ciclos repetidos de pruebas que retrasan la salida a producción del proyecto generando sobrecostos. • Riesgos técnicos: El manejo de análisis y diseño separado de la especificación puede implicar problemas de entendimiento y por lo tanto reprocesos costosos. • Riesgos de comunicación: Los problemas de comunicación son el mayor riesgo que se corre en este patrón, el traspaso de conocimiento entre el cliente, la fábrica de requerimientos, la fábrica de desarrollo y la fábrica de pruebas hace que se incrementen los posibles conflictos, sobre todo cuando hay ANS de por medio que implican descuentos importantes.
Usos conocidos	Este modelo se ha implementado en entidades públicas en Colombia como UGPP y Colpensiones. A continuación, se encuentra una gráfica con el proceso manejado en Colpensiones, con sus entregables y aprobaciones hechas por los Líderes de Gestión (LG).



Ver también

NA

Atributo	
Nombre	División de producción
También conocido como	NA
Autor	Fuente propia
Propósito	Crear productos con la construcción de componentes separada, permitiendo bajar los costos y el tiempo de desarrollo.
Problema	Es necesario tener un producto en bajos tiempos y costos para salir al mercado, y se hace necesario usar la fuerza de trabajo de más de una fábrica de software para paralelizar o utilizar la experiencia de negocio específica en componentes de algunas fábricas.
Contexto	Es necesario generar un producto de software con una limitante de fecha establecida por el patrocinador, restricciones de negocio o legales, y se necesita utilizar la experiencia de varias fábricas. En ocasiones algunos componentes (e.g. Biometría, Georeferenciación, Componentes de negocio, etc.) requieren de una experiencia específica de la fábrica que los desarrolla, por lo cual la mejor decisión es permitir que un componente con una interfaz bien especificada sea desarrollado por expertos en la materia. Variables: <ul style="list-style-type: none"> Número de riesgos: Alto. El uso de fábricas expertas tiene el objeto de mitigar los riesgos del desarrollo de la aplicación con limitantes de tiempo y componentes específicos.

- Estabilidad y claridad de requerimientos: Claramente definidos y estables. Son requeridos para hacer un diseño y arquitectura guía para todo el desarrollo.
- Experiencia del equipo de desarrollo: Alta experiencia. Es una de las ganancias al usar fábricas de software específicas por componente.
- La flexibilidad al cambio en el proyecto: Baja flexibilidad. El proyecto tiene una limitante de tiempo y es uno de los factores que determina su uso.
- Involucramiento de usuarios: NA. No es un factor decisivo.
- Distribución geográfica del equipo: NA. No es decisiva, pero puede ser geográficamente dispersa para mejorar costos.
- Uso de fábricas de software: Desarrollo con más de dos fábricas de software.

Solución

Separar claramente las etapas de ingeniería de la producción de software, como lo sugiere Royce, W. (1998, p. 75), permite controlar la producción mediante el control de la producción a través del análisis y diseño detallado.

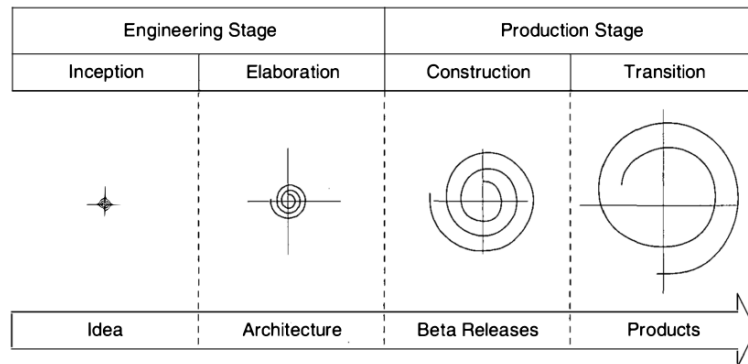


Figura 26 Fases de un ciclo de vida, tomado de (Walker Royce, 1998)

En este patrón, la idea es generar un desarrollo que se pueda dividir, a partir de una distribución de componentes en diferentes proveedores, que se coordinan con unas interfaces claramente establecidas y diseñadas. El tener establecidas las responsabilidades e interfaces de cada componente, de forma precisa, reduce el riesgo de integración, adicionalmente, permite utilizar grandes grupos de desarrollo con bajos riesgos de comunicación o interacción.

A continuación, en la figura 25 se muestra el esquema general del patrón, se llevan a cabo las etapas de Ingeniería para poder separar el trabajo claramente con el fin de integrar los componentes después, y poder hacer las respectivas pruebas de integración y aceptación para su liberación.

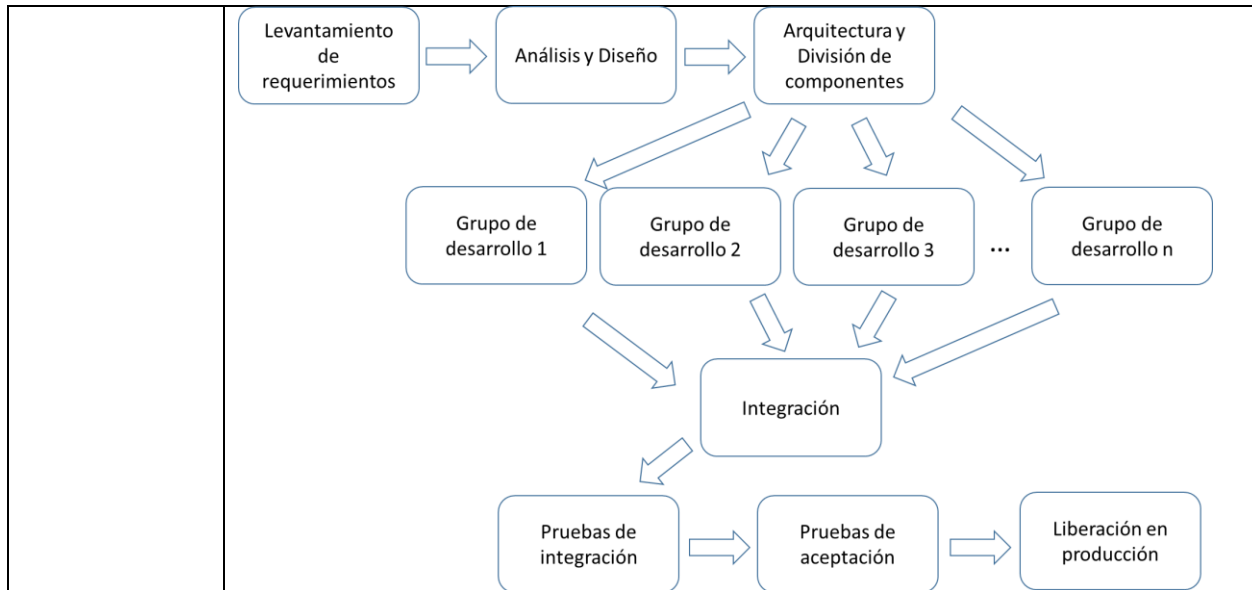
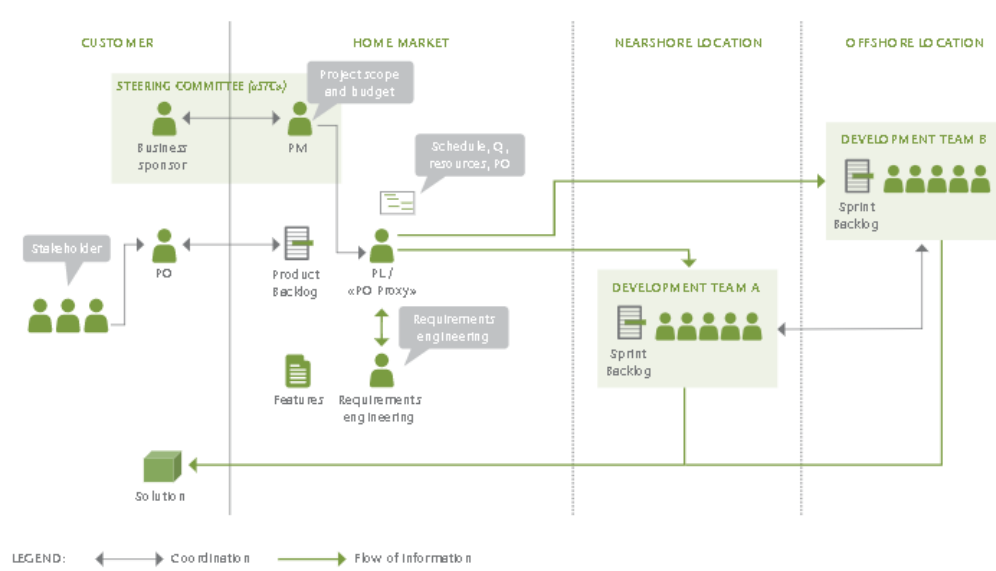


Figura 27 División de trabajo en componentes. Fuente propia

Los grupos de desarrollo pueden ser fábricas diferentes o equipos de trabajo diferentes, incluso separados geográficamente y con zonas horarias diversas. Para este patrón, todos los modelos de SDLC son apropiados, una vez que se ha generado apropiadamente la fase de diseño y arquitectura, la distribución de trabajo es posible.

<p>Consecuencias</p>	<p>Beneficios:</p> <ul style="list-style-type: none"> • Distribución y generación de aplicaciones de gran tamaño en menores tiempos, debido a la coordinación de grandes grupos de trabajo mediante las interfaces y responsabilidades claramente definidas en el diseño y arquitectura. • Delegación del riesgo a diferentes proveedores. • Desarrollo especializado de componentes con fábricas expertas. <p>Desventajas:</p> <ul style="list-style-type: none"> • Distribución de responsabilidades y posibilidad de atrasos generales por el atraso de uno de los subproyectos. • Problemas de comunicación de las especificaciones, sobre todo si se trata de equipos de trabajo distribuidos globalmente, o con diferencias culturales. <p>Riesgos:</p> <ul style="list-style-type: none"> • Riesgos del proyecto: Si existen retrasos en uno de los componentes puede retrasar la integración del proyecto total, causando sobrecostos e incumplimiento de compromisos. • Riesgos técnicos: aunque las interfaces se establecen, defectos en datos pueden retrasar las pruebas de integración. Los defectos en diseño suelen ser muy costosos ya que el reproceso de desarrollo implica devolverse en todos los procesos de integración y pruebas, con dificultades para determinar los orígenes de los problemas, se recomienda el uso de herramientas de debug y profiling para recorrer
----------------------	---

	<p>el software de los componentes, así como exigir estándares de nombramiento y desarrollo en los componentes.</p> <ul style="list-style-type: none"> • Riesgos de comunicación: Las especificaciones incompletas o creadas de forma inconsistente pueden ser un problema con las fábricas de desarrollo creando reprocesos costosos. Especificaciones en idiomas diferentes o generados por culturas diferentes pueden causar inconvenientes grandes, como se pudo evidenciar en la experiencia de TATA en Colombia, donde el desarrollo en la India no fue fácil por las diferencias culturales.
<p>Usos conocidos</p>	<p>La división de producción es un esquema usado por fábricas de software a nivel mundial, en la figura a continuación se muestra cómo este patrón puede ser aplicado con fábricas cercanas <i>Nearshore</i> o remotas <i>Offshore</i>, incluso en el mismo proyecto.</p>  <p>The diagram illustrates the flow of information and coordination across four geographical zones: CUSTOMER, HOME MARKET, NEARSHORE LOCATION, and OFFSHORE LOCATION. In the CUSTOMER zone, a Business Sponsor and a Steering Committee (STC) are involved. In the HOME MARKET, a PM (Project Manager) and a PO (Product Owner) manage the Product Backlog. The PO interacts with Stakeholders and Requirements Engineering. In the NEARSHORE LOCATION, two Development Teams (A and B) work on their respective Sprint Backlogs. The PO acts as a 'PO Proxy' for the teams. In the OFFSHORE LOCATION, another Development Team works on its Sprint Backlog. Information flows from the Business Sponsor to the PM, then to the PO, and finally to the development teams. Coordination is shown between the Business Sponsor and PM, and between the PM and PO. A legend indicates that double-headed arrows represent 'Coordination' and single-headed arrows represent 'Flow of Information'.</p> <p>Figura 28 Proceso mostrando el proceso con Nearshore y offshore. Tomado de (Erni, 2015, p. 24)</p> <p>En Mapfre Seguros se implementa una variable que incluye una fase de presentación de propuestas por parte de las fábricas, y elección de la fábrica por mejores experiencias en proyectos anteriores, costo y tiempo estimados y carga de trabajo en la empresa, aunque en este modelo no se hace diseño antes del desarrollo, sino que es una tarea que realizan las fábricas a partir de la especificación.</p>
<p>Ver también</p>	<p>NA</p>

Atributo	
Nombre	Siguiendo al sol (FTS: Follow The Sun) tomado de (Carmel et al., 2010)
También conocido como	Global Software Development (GSD)
Autor	Carmel, E., Espinoza, J. & Dubinsky, Y.
Propósito	Este patrón proporciona una estructura para hacer desarrollo tomando en cuenta que el cambio de uso horario puede extender el tiempo de trabajo de los equipos, proporcionando una jornada de trabajo más larga y la posibilidad de aumentar las horas de trabajo de los equipos combinados, reduciendo el tiempo de trabajo.
Problema	Es necesario reducir los tiempos de producción, pero los horarios locales imponen una restricción para lograr los desarrollos en los plazos requeridos por el negocio. En empresas globales como IBM o HP el desarrollo combinado de múltiples fábricas en diferentes zonas horarias, se utiliza para generar productos en tiempos reducidos que les permitan competir en el mercado.
Contexto	<p>Se establecen fábricas de software a nivel global, especialmente para empresas multinacionales o aliadas que deben trabajar juntas. Las fábricas se encuentran en zonas horarias diferentes, y esto en lugar de convertirse en un impedimento para trabajar, se puede volver una ventaja con unos procesos de integración y separación de actividades bien establecidas. En un punto específico de tiempo, sólo una de las fábricas tiene el control del producto. La entrega de producto se produce diariamente al final de las tareas planeadas para el día.</p> <p>Variables:</p> <ul style="list-style-type: none"> • Número de riesgos: Alto. El adicionar un componente de comunicación especial al proyecto implica un riesgo adicional. • Estabilidad y claridad de requerimientos: Claramente definidos y estables. Es preferible que todo el equipo los conozca con claridad, así como los casos de prueba esperados. • Experiencia del equipo de desarrollo: Alta. • La flexibilidad al cambio en el proyecto: Alta. Los movimientos de trabajo entre sedes requieren que se tengan márgenes de movilidad. • Involucramiento de usuarios: Bajo o Normal. No es un factor decisivo. • Distribución geográfica del equipo: Requerida. • Uso de fábricas de software: Desarrollo con más de dos fábricas de software.
Solución	Generar un proceso de desarrollo basado en la división de tareas utilizando un método de rotación de trabajo, destinado a reducir la duración del proyecto, en el que el producto de conocimiento es poseído y avanzado por un sitio de producción, y es entonces entregado al final de cada jornada de

trabajo al siguiente sitio de producción, en una zona horaria diferente (Carmel et al., 2010, p. 5). Una configuración óptima de FTS puede llevar a incrementar la eficiencia del proceso en un 71.4%, en la experiencia que muestra Carmel en su ensayo de un FTS de cuatro sitios.

FTS funciona bien con un proceso donde cada sitio ejecuta una fase, por ejemplo, si un sitio se especializa en pruebas mientras otro hace desarrollo, los bugs detectados se reportan en un sistema y el equipo de desarrollo los soluciona, esto implica un desfase de tiempo entre el inicio de trabajo del equipo de desarrollo en comparación con el equipo de pruebas.

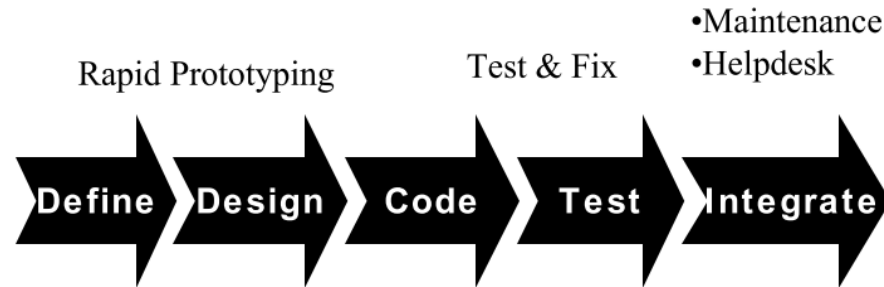


Figura 29 Ciclo de vida FTS. Tomado de (Carmel et al., 2010)

Existen 12 proposiciones que se deben cumplir para un correcto FTS:

1. Comparado con otro proceso de configuración global, FTS incrementa la eficiencia de cronograma substancialmente y esta eficiencia incrementa cuando incrementa el número de sitios.
2. En relación con el trabajo que abarca varias fases del SDLC, el trabajo dentro de una fase SDLC particular, es más adecuado para el desarrollo FTS debido a su especificidad permite traspasos más estructurados y granulares
3. En comparación con las configuraciones globales convencionales, FTS es más adecuado para el desarrollo ágil cuando se utilizan algunas prácticas ágiles básicas: pequeñas iteraciones enmarcadas en tiempo, pruebas automáticas exhaustivas, integración continua, y ritmo sostenible.
4. FTS tendrá más éxito para arquitecturas de productos que dividen el software en componentes más pequeños, relativamente independientes (e.g. características, solicitudes de modificación, módulos).
5. FTS es más adecuado para el desarrollo de componentes de productos que para la integración de componentes.
6. La idoneidad de FTS aumenta cuando un componente de producto es más cohesivo funcionalmente y mejor definido
7. A medida que aumenta la eficiencia de los traspasos de trabajo, también lo hace la velocidad de desarrollo de FTS.
8. Aumentar el número de sitios FTS que tienen baja eficiencia en traspasos conduce a la disminución de mejoras marginales en la velocidad de duración.
9. El potencial de ganancias de velocidad debido a la reducción de coordinación en el sitio, aumenta exponencialmente para grandes

- equipos, si los equipos se dividen en sub-equipos más pequeños a través de múltiples sitios FTS
10. El potencial de ganancias de velocidad debido a la disminución de la coordinación dentro del sitio necesaria en FTS aumenta exponencialmente mientras los equipos se distribuye en más sitios FTS, pero estos beneficios están limitados por el tiempo de tarea diaria.
 11. Enmarcando en el tiempo, un subproducto de cualquier configuración FTS, estimula la productividad individual (en relación con otras configuraciones) debido al rigor añadido, el sentido de fecha límite y la orientación a la meta.
 12. FTS es beneficioso para la velocidad de desarrollo de software cuando la reducción de la duración debido a la disminución de la coordinación en sitio, además de una mayor productividad individual por el enmarcado de tiempo, es mayor que el aumento de la duración debido al aumento de la coordinación de traspasos entre sitios.

La figura a continuación muestra cómo se puede coordinar el trabajo de tres sitios.

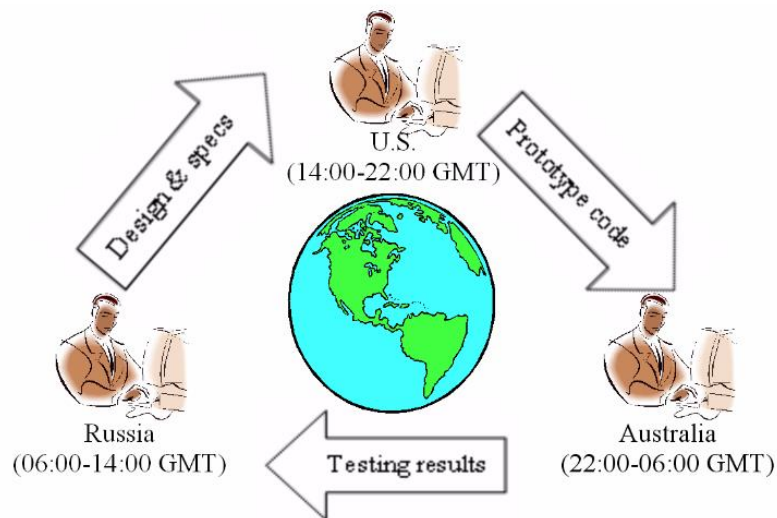


Figura 30 Ciclo de desarrollo con distribución global. Tomado de (Gupta, Sloan School of Management., & Seshasai, 2004)

Consecuencias

Beneficios:

- Mejora los tiempos de entrega de los productos al ampliar la jornada de trabajo.
- Disminución de tiempos de coordinación interna de los grupos de trabajo.

Desventajas:

- La coordinación de trabajo implica el uso de herramientas de integración continua, ya que los traspasos requieren una disciplina exacta para evitar retrabajos y pérdidas de tiempo por malos entendidos o incomprensión del trabajo de los otros sitios.

	<ul style="list-style-type: none"> En ocasiones son necesarias reuniones con otros sitios de trabajo, pero debido a la diferencia horaria, estas se hacen imposibles en horarios de trabajo normales. <p>Riesgos:</p> <ul style="list-style-type: none"> Riesgos de proyecto: Los retrasos en trabajo de un sitio puede impactar el trabajo de otro. Cuando hay días festivos en un país como Colombia, es posible perder tiempo en los otros sitios, especialmente si la tarea específica del país con día festivo es prerequisite de las tareas en otros sitios. Riesgos de negocio: Las reducciones de tiempo esperadas de FTS se pueden perder si hay descoordinación de tareas entre los sitios, produciendo demoras que pueden afectar la salida de productos al mercado. La mitigación que proponen los autores es el desarrollo de paquetes de software más pequeños. Riesgos de comunicación: La descoordinación de horarios hace que las reuniones sean posibles sólo en ventanas de tiempo pequeñas. Es muy probable que haya diferencias culturales que deban ser trabajadas para evitar conflictos que afecten los traspasos de trabajo.
Usos conocidos	En los años 90 IBM lo implemento en cinco locaciones diferentes, pero no fue exitoso. Las primeras implementaciones exitosas las llevaron a cabo en EDS, actualmente HP.
Ver también	24-Hour Knowledge Factory (Gupta et al., 2004)

	Cascada	Modelo-V	Espiral	RUP	XP	Scrum
Desarrollo de sistemas en tiempo real basado en modelos			X	X	X	X
Conjunto de pruebas de Bootstrapping	X	X				
PIP			X	X	X	X
Incremental			X	X		
Evolutivo			X	X		
Entrega incremental			X	X	X	X
Diseño grande	X	X	X	X	X	X
MED			X	X	X	X
Fábrica de pruebas	X	X	X	X	X	X
Zipper	X	X				
División de producción	X	X	X	X	X	X
FTS					X	X

Tabla 3 Patrones aplicables a los diferentes Modelos de SDLC. Tomado de: Fuente Propia

A continuación, se encuentra un cuadro con los valores de las variables propias de cada patrón:

	Número de riesgos	Estabilidad y claridad de requerimientos	Experiencia del equipo de desarrollo	Flexibilidad al cambio en el proyecto	Involucramiento de usuarios	Distribución geográfica del equipo	Uso de fábricas de software
Desarrollo de sistemas en tiempo real basado en modelos	Alto	Vagamente definidos o inestables.	Alta (Es un tema muy específico)	Alta	Normal	Equipo local	Desarrollo <i>inhouse</i> o con una fábrica
Conjunto de pruebas de Bootstrapping	NA	Claramente definidos y estables	Alta	Baja	NA	NA	Desarrollo con dos fábricas de software
PIP	Alto	Vagamente definidos o inestables.	Alta	Alta	NA	NA	NA
Incremental	Bajo o medio	Vagamente definidos o inestables	Alta	Baja	Alto	Equipo local	Desarrollo con una fábrica de software
Evolutivo	Bajo o medio	Vagamente definidos o inestables	Baja	Baja	Alto	NA, es posible con equipo remoto.	Desarrollo con una fábrica de software
Entrega incremental	Bajo o medio	Claramente definidos y estables	Alta	Baja	Bajo o Normal	NA	Desarrollo con una fábrica de software

	Número de riesgos	Estabilidad y claridad de requerimientos	Experiencia del equipo de desarrollo	Flexibilidad al cambio en el proyecto	Involucramiento de usuarios	Distribución geográfica del equipo	Uso de fábricas de software
Diseño grande	Bajo o medio	Claramente definidos y estables.	Alta	Baja	NA	NA	NA
MED	Alto	Vagamente definidos o inestables.	Alta	Alta	Alto	NA	NA
Fábrica de pruebas	NA	Claramente definidos y estables	Alta	Baja	Bajo o Normal	NA	Desarrollo con dos fábricas de software
Zipper	Alto	Vagamente definidos o inestables	Alta	Baja	Bajo o Normal	Permite tener fábricas distribuidas	Dos o tres fábricas
División de producción	Alto	Claramente definidos y estables	Alta	Baja	NA	Puede ser geográficamente dispersa para mejorar costos	Más de dos fábricas de software
FTS	Alto	Claramente definidos y estables	Alta	Alta	Bajo o Normal	Requerida	Desarrollo con más de dos fábricas de software.

Tabla 4 Relación de patrones contra variables. Tomado de: Fuente propia

Árbol de decisión sobre los patrones de SDLC

Como una forma de apoyar el uso de estos patrones de forma rápida, se planteará un árbol de decisión que permitirá orientar al usuario al momento de seleccionar un patrón de acuerdo a las características de su proyecto, por supuesto este es un tema de conocimiento que no ha tenido tanta exploración, por lo cual pueden existir vacíos y es tema de futuras investigaciones determinar los patrones que pueden suplir dichos vacíos. Para generar los puntos de decisión se usarán las variables determinadas y sus posibles valores, y se usó el modelador de procesos del producto Bizagi para hacer la gráfica, tomando en cuenta que es un proceso de decisión lo que se va a trabajar.

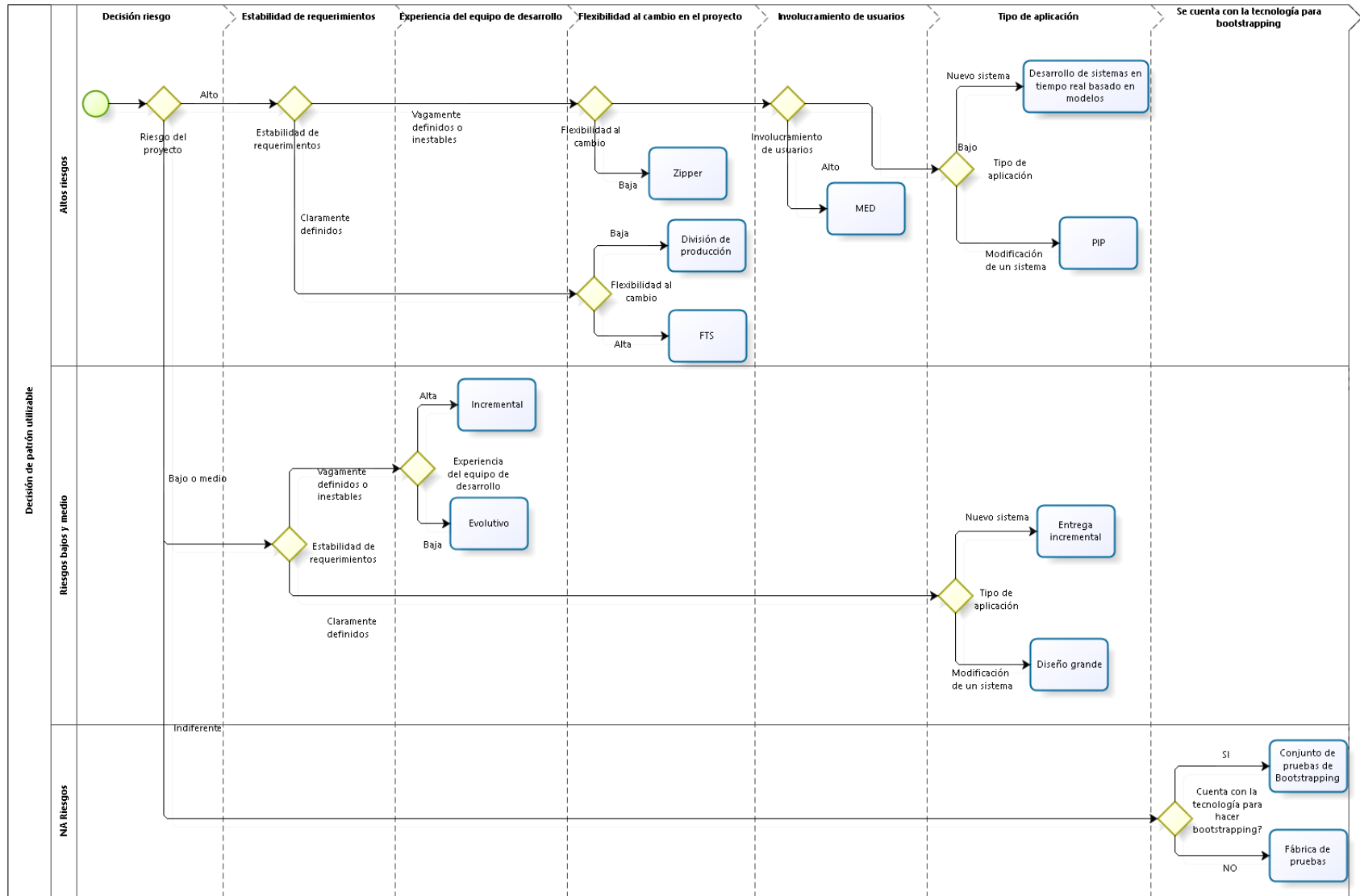


Figura 31 Árbol de decisión en la elección de un patrón de SDLC. Tomado de: Fuente propia

Aplicabilidad en el entorno colombiano

Entorno del medio colombiano

En diferentes estudios se hace énfasis en cifras similares sobre las empresas que conforman el entorno colombiano, por ejemplo, según el Informe de caracterización del sector de software y tecnologías de la información en Colombia (Fedesoft & Ministerio de Tecnologías de la Información y las comunicaciones de Colombia, 2015, p. 27), a nivel nacional el 36,5% de las empresas tienen ventas anuales entre \$50 y \$100 millones de pesos, el 27% entre \$100 y \$500 y sólo el 9,9% superan los \$1.000 millones de pesos y la mitad de la facturación proviene de dos líneas de negocio, Software como servicio 27.09% y Desarrollo/Fábrica de software 23.67%. El 60% de las empresas tienen como patrimonio menos de 294 millones, por lo cual tienen un alto grado de inestabilidad y el 91% de las empresas invierten menos de 294 millones en Investigación y desarrollo, lo cual implica un obstáculo para el crecimiento esperado para el sector por parte del gobierno.

En el estudio de Díaz & Ospina, titulado *Prospectiva 2019 - 2023 para Mipymes dedicadas a desarrollo de software por encargo en Colombia*, las empresas de desarrollo se dividen en: 92 % son micro y pequeñas empresas, el 7 % medianas empresas y el 1 % grandes empresas (2014, p. 77). La mayoría de las microempresas, se crean porque su costo de inversión inicial no es demasiado alto, ya que su mayor activo es el conocimiento, pero así mismo tienen muchos problemas porque sus dueños en la mayoría de casos son ingenieros de sistemas (95%) sin conocimientos administrativos, otro de los problemas es que no se adoptan estándares internacionales para lograr resultados de calidad, precisamente, debido a que no existe un presupuesto para inversión en investigación y desarrollo.

En el estudio de Martínez et al., El crecimiento de la industria del software en Colombia: un análisis sistémico, se destaca que los países que han tenido un crecimiento económico basados en el crecimiento de su industria de software, lo hicieron porque hicieron énfasis en el desarrollo a la medida para nichos de mercado específicos y tuvieron crecimiento al incursionar en mercados de otros países (2015, p. 96). La industria del software en Colombia está alineada con el Programa de transformación productiva, del Ministerio de Comercio Industria y Turismo. El objetivo de dicho programa es impulsar a Colombia como uno de los países más competitivos de la región en el periodo comprendido entre 2007 y 2023, pero realmente el crecimiento sólo ha sido del 8% contra el 17% esperado. Una conclusión importante de este estudio, es que las microempresas de desarrollo de software carecen de un sistema industrial y su dinámica en un mercado altamente competido.

En el reporte de Fúquene, Castellanos, & Fonseca, Bases de la implementación de un modelo de inteligencia para fortalecer el desarrollo tecnológico de la industria del software y servicios asociados en Colombia, si bien es anterior a los artículos e informes mencionados previamente, destaca que en este tiempo dos actividades son de alta relevancia para el sector del software, gestión de la calidad del software y la creación de herramientas de ingeniería de software (Fúquene et al., 2007, p. 188). Aunque en esta investigación se hace una proyección de oportunidades y estrategias para el sector hasta el 2015, viendo los informes correspondientes, se nota que no fueron aplicados y se perdieron 8 años de posible avance. Se destaca que el desarrollo de software a la medida es la actividad más representativa para el posible avance del sector.

En el estudio de Palomino, Estudio del comportamiento de la industria del software en Colombia ante escenarios de capacidades de innovación y ventajas comparativas por medio de dinámica de

sistemas, se llega a conclusiones similares: *“La problemática de la industria del software en Colombia se resume básicamente en: Precios poco competitivos para el mercado internacional, bajos estándares de calidad, inconveniente con el idioma de los países a exportar, poco personal especializado, poca experiencia en mercados internacionales. Todos estos factores apuntan a un bajo nivel de capacidades de innovación que terminan en un bajo nivel de ventajas comparativas, lo cual significa firmas menos competitivas y por ende la industria del software del país se ve afectado por estas falencias.”* (Palomino, 2011, pp. 30–31).

Revisando el estudio de González, Sánchez, & Velandia, Identificación y análisis de factores de éxito en la gerencia proyectos en algunas PYMES del sector TI en Bogotá D.C. Colombia (González et al., 2016), se tuvieron en cuenta a 71 empresas, de las cuales 14 son grandes que fueron entrevistadas y 57 PyMEs que fueron encuestadas, las cuales tienen al desarrollo de software como su principal tipo de proyecto, dentro de las diferencias encontradas se destacan los siguientes puntos:

	Empresas Grandes	PyMEs
Desarrollo de software como principal tipo de proyecto	46%	40%
Porcentaje de proyectos no exitosos	42%	17%
Considera que aporta al éxito la gerencia de proyectos	95%	72%

Tabla 5 Comparación proyectos empresas grandes y PyMEs Bogotá. Tomado de (González et al., 2016, p. 132)

Finalmente, en la tabla a continuación se encuentra la calificación que recibieron los factores de éxito para proyectos de IT, los resultados son que la especificación del proyecto es la más relevante, pero quedan con aspectos influyentes: las competencias adecuadas del gerente del proyecto, la experiencia y capacidades del equipo de trabajo, la gestión adecuada de riesgos y el uso adecuado de una metodología de gestión de proyectos.

ID	Factor de éxito	Muy Influyente	Influyente	Menos Influyente
FE 2.1	Alcance bien definido del proyecto	79.60%	18.50%	1.90%
FE 2.2	Realizar planeación adecuada del proyecto	70.40%	25.90%	3.70%
FE 2.3	Seguimiento y control del proyecto	66.70%	33.30%	1.90%
FE 2.4	Apoyo de la alta dirección	57.40%	38.90%	3.70%
FE 2.5	Comunicación efectiva con los Stakeholders	53.70%	44.40%	1.90%
FE 2.6	Competencias adecuadas del gerente del proyecto	44.40%	51.90%	3.70%
FE 2.7	Experiencia del equipo de trabajo del proyecto	40.70%	50.00%	9.30%
FE 2.8	Gestión adecuada de riesgos	38.90%	50.00%	11.10%
FE 2.9	Conocimiento y capacidades adecuadas del equipo de trabajo del proyecto	33.30%	61.10%	3.70%
FE 2.10	Uso adecuado de una metodología de gestión de proyectos	24.10%	57.40%	18.50%

Tabla 6 Factores de éxito en los proyectos empresas de Bogotá. Tomado de (González et al., 2016, p. 132)

Finalmente, resalta que uno de los factores clave para el éxito de los proyectos es la apropiada planeación, es fundamental porque permite detectar los riesgos involucrados y evita generar problemas más costosos en etapas posteriores, por lo cual para las PyMEs es fundamental que se tome el tiempo para planear y diseñar el proyecto.

Tomando como conclusión de estos informes y estudios, la industria colombiana está conformada en su mayoría por PyMEs, las grandes empresas son más rentables, pero en general todas invierten menos de 294 millones en investigación y certificaciones internacionales de procesos, lo cual genera que las microempresas carezcan de un sistema industrial para su producción y nos lleva a evidenciar fallas en la calidad y problemas para la internacionalización de los servicios. El desarrollo de software es no sólo la actividad más representativa y lucrativa del sector, sino es el tipo de proyecto más frecuente. Adicionalmente, se destaca la necesidad de

tener una fase de planeación para el desarrollo efectivo de los proyectos, ya que permite evitar riesgos y detectar posibles problemas de forma temprana, reduciendo los costos y por lo tanto ayudando a las empresas a ser más eficientes y rentables, es en este punto donde una guía de patrones de SDLC puede ayudar a las empresas sin procesos estandarizados a tener mayor éxito en sus proyectos, al seguir modelos probados a nivel mundial, sin necesidad de hacer inversiones adicionales.

SDLC aplicados en el mercado colombiano

En 2004 el artículo de Casallas & Arboleda(2004), se analiza que las empresas de desarrollo colombianas no tienen en su mayoría el presupuesto para implementar procesos internacionales de calidad, por lo cual se generó un proceso adaptable a las necesidades específicas de los proyectos y basándose en metodologías ágiles. Presenta un aspecto hasta ahora no encontrado en otras investigaciones, y es la alta rotación de recursos, lo cual obliga a mantener un proceso constante de administración de conocimiento para no tener un impacto demasiado alto debido a dicha rotación. No está planteado como un patrón específico, pero una investigación posterior podría generar uno con un análisis de los proyectos que lo utilizaron, sus ventajas y riesgos asociados.

De acuerdo al informe de Arboleda (2002), Modelos de Ciclo de Vida de Desarrollo de Software en el Contexto de la Industria Colombiana de Software, ratifica que el modelo más usado es Cascada, pero muestra como el desarrollo en Espiral (Incremental de Boehm) y el orientado a prototipos tienen buena recepción, pero son mal implementados debido a un bajo conocimiento

de la teoría alrededor de los modelos, así mismo, muestra cómo empezaba en 2002 la curiosidad por el uso de metodologías ágiles como XP, Scrum o Kanban.

En el artículo de Moreno, Rengifo, & Navia, Un acercamiento a las prácticas de calidad de software en las MiPyMESP (Micro, pequeñas y medianas empresas productoras de software) del suroccidente colombiano (2010), describen que para el 60% de las empresas encuestadas, la metodología utilizada en el desarrollo es de alta importancia para el resultado del proyecto, aunque hay desconocimiento de estándares internacionales, dificultando la aplicación de metodologías estándar o de una industrialización.

Finalmente, el estudio de García, Análisis descriptivo del impacto de la aplicación de la metodología cmmi-dev en proyectos de desarrollo de software en empresas colombianas (García, 2014), se describe que el 82% de la industria de software colombiana implementa una metodología de desarrollo de software, teniendo que cerca del 44% de estas usa RUP, mezclado con metodologías ágiles como Scrum y XP. También indica que, de acuerdo a los procesos adelantados con el gobierno colombiano, hay iniciativas para implementar PSP y TSP en las PyMEs, en convenio con Fedesoft y el SENA.

En conclusión, el desarrollo de software en Colombia usa modelos de desarrollo estándar, con bajo uso de estándares internacionales, llevando a las empresas a implementar RUP, XP o Scrum, pero sin tener en cuenta todas las mejores prácticas para llevar a buen término los proyectos de desarrollo de software. Los patrones son totalmente aplicables como complemento a un análisis juicioso de las situaciones del proyecto, permitiendo que se use una guía rápida para implementar mejores prácticas internacionales, sin la necesidad de hacer inversiones cuantiosas

como las que implica una certificación CMMI-DEV 3 o superior. El uso de patrones que involucren varias fábricas de software permitirían las implementaciones de grandes proyectos, con mayor rentabilidad, al permitir la unión de varias PyMEs en un cluster con un margen de riesgo reducido.

Metodologías aplicables

Algunas metodologías o técnicas que pueden ayudar a la implementación de los patrones expuestos en esta monografía, serán presentadas en este capítulo, algunas de ellas ya han sido mencionadas en los mismos patrones, por ejemplo, AOSD, otras son complementarias, a cualquier patrón, como las metodologías usables en levantamiento de requerimientos.

- AOSD: Aspect Oriented Software Development, Desarrollo orientado a aspectos propuesto por Kiczales et al. (1997) en el laboratorio de Xerox en Palo Alto, tiene como principal propósito mejorar la solución que da la programación orientada a objetos, a problemas complejos con varios dominios. Uno de los planteamientos de AOP o AOSD, es que el software puede tener requerimientos no funcionales que dificultan su comprensión, o que si se sigue la creación de aplicaciones orientadas a objetos se mejora su comprensión, pero dificulta cumplir con los requerimientos, especialmente de desempeño. Los aspectos tienden a no ser unidades de descomposición funcional del sistema, sino más bien a ser propiedades que afectan al rendimiento o la semántica de los componentes en formas sistémicas. Los ejemplos

de aspectos, como los patrones de acceso a memoria y la sincronización de objetos concurrentes. La estrategia consiste en generar componentes específicos que servirán como filtros del procesamiento mientras el flujo de datos pasa por los componentes de la aplicación, es necesario usar lenguajes específicos que permiten hacer esta implementación.

- ATDD (Acceptance test–driven development, Desarrollo orientado a pruebas de aceptación): Como lo define Pugh (2011, p. 1), esta metodología se basa en: “Una tríada (el analista de cliente / negocio, desarrollador y probador) colaboran en la producción de estas pruebas para aclarar lo que se debe hacer. En la creación de un producto de alta calidad, ATDD se trata tanto de esta producción como de la prueba real.”. Se basa en que una de las tareas más efectivas para lograr buen código es desarrollar las pruebas de aceptación antes de codificar, esto mejora el entendimiento de los requerimientos antes de su desarrollo.

El ciclo propuesto se encuentra descrito en la figura a continuación. En el primer paso el equipo (tríada) discute las historias de usuario y genera los casos de prueba; en el segundo paso se destilan los casos de prueba y se llenan los formatos respectivos; en el tercer paso se hace el desarrollo y se generan los prototipos a probar para su implementación en producción.

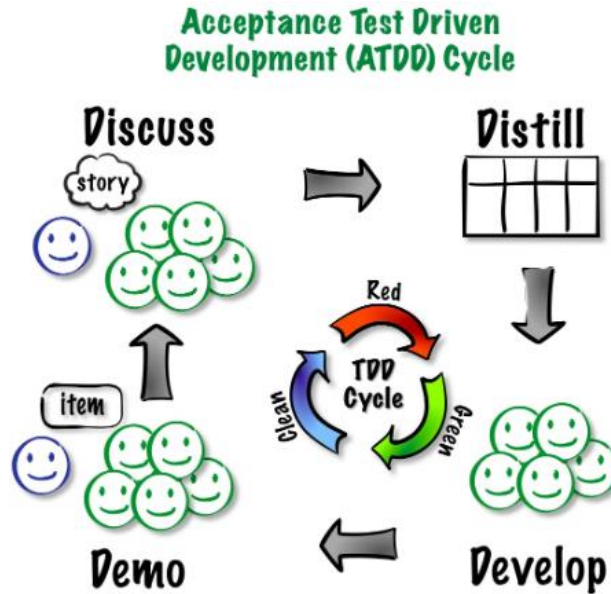


Figura 32 Ciclo de ATDD Tomado de (Hendrickson, 2008)

- BDD: Behavior-driven development, Desarrollo orientado a comportamiento. BDD pretende crear una relación entre el lenguaje de usuario funcional o de negocio y la parte técnica, y que sea desde esta relación desde donde arranque la definición de pruebas y, desde estas, el desarrollo.

En BDD, las pruebas se crean usando las especificaciones en el formato de Historias de usuario, en el esquema: “Como [rol] quiero [característica] para que [los beneficios].”. El núcleo de BDD es "conseguir las palabras correctas", es decir, la producción de un vocabulario que es preciso, accesible, descriptivo y consistente (Duarte & Fernandes, n.d.). Una implementación en Java, JBehave (“JBehave,” n.d.), funciona usando el formato de especificación Gherkin, en el cual se determinan escenarios funcionales explicando quién ejecuta, cuando y qué acciones.

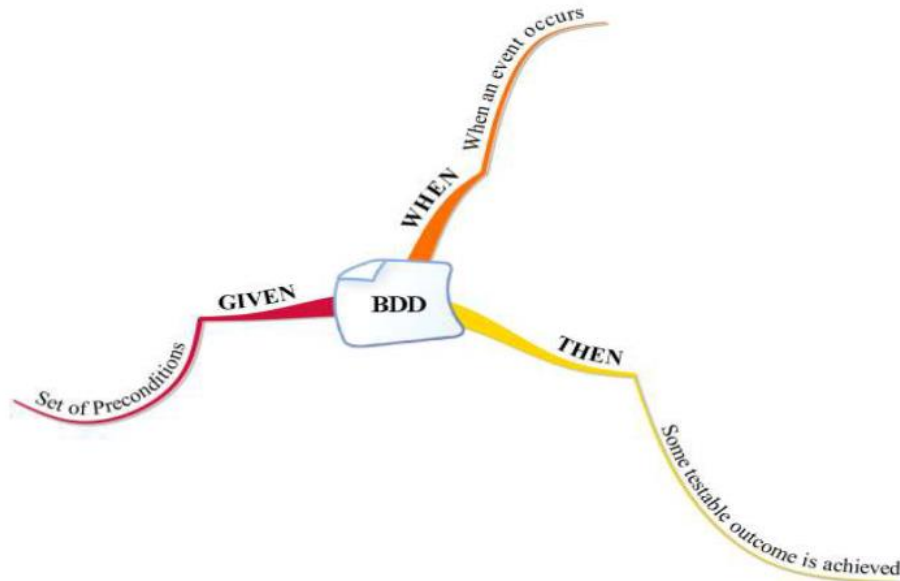


Figura 33 Escenarios de BDD. Tomado de (Duarte & Fernandes, n.d.)

Escribiendo los escenarios en la notación mencionada previamente, se logran crear unos casos de prueba consistentes, con sus criterios de aceptación, a los cuales se asignan etiquetas para llevar estas al código fuente, de forma que se crean casos de prueba automatizados que permiten verificar que el código cumple con la especificación escenario por escenario. El ciclo básico de BDD es el siguiente:

- 1 – Escribir las historias de usuario.
 - 2 – Escribir los escenarios.
 - 3 – Automatizar las pruebas de aceptación.
- **MDS**: Model-driven software development, Desarrollo de software orientado a modelos. En esta metodología, los diseños y modelos no son sólo documentación, sino que están al mismo nivel del software (Stahl et al., 2006). Se basa en un lenguaje

de modelado estándar como UML, y herramientas que transforman los modelos en código, o base para que este se cree a partir del esquema creado automáticamente.

Una ventaja de los modelos es que son independientes de plataforma, así que pueden generar el código base para cualquier lenguaje o tecnología específica. Los modelos independientes de plataforma se transforman en modelos específicos de plataforma, y luego estos en código.

En la gráfica a continuación, se muestra el proceso para un ejemplo específico en JEE donde se hace una especificación genérica, luego se transforma en elementos propios de la plataforma y finalmente en código.

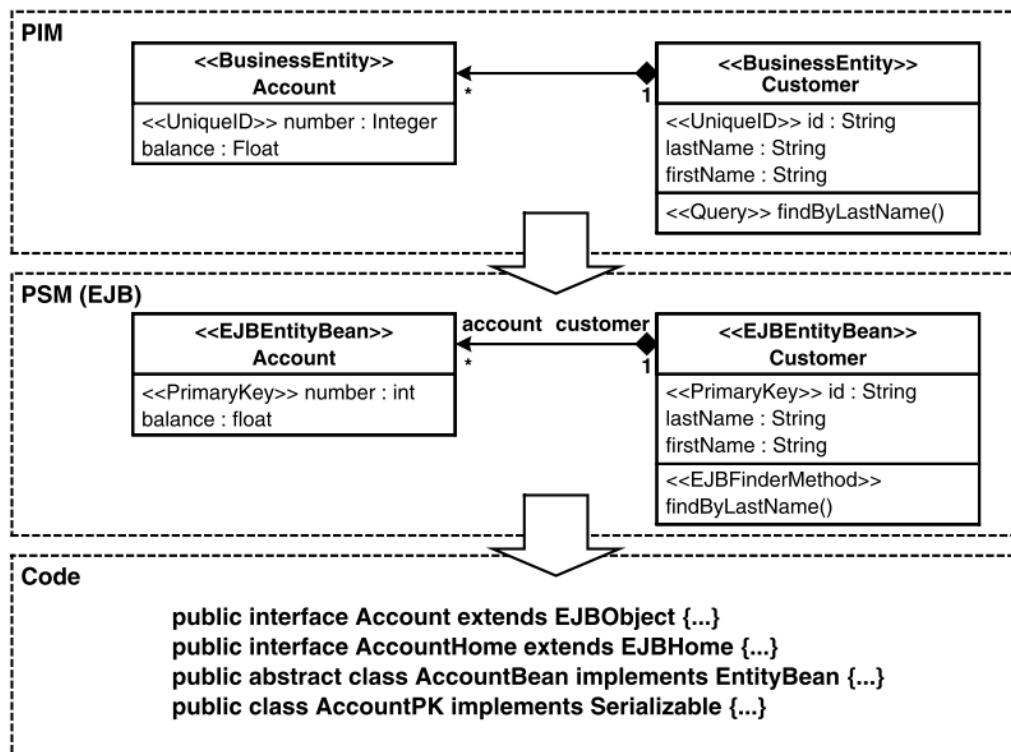


Figura 34 Proceso de transformación de MDS. Tomado de (Stahl et al., 2006)

- TDD: Test-driven development, Desarrollo orientado a Pruebas. En TDD, al contrario de ATDD, las pruebas son generadas por los desarrolladores (Schroeder & Rothe, 2005), pero tienen un ámbito totalmente técnico que permite su automatización y repetitividad, tienen el mismo sentido de desarrollar las pruebas antes de desarrollar código, lo cual puede hacer sentir algo incómodos a los desarrolladores antes de tomar la costumbre de trabajar con esta metodología. Una buena herramienta para este tipo de prueba (En tecnología Java) es JUnit, que permite definir escenarios y casos de prueba automatizables. TDD es totalmente compatible con la integración continua, ya que permite conocer de forma automática si el código integrado cumple con las pruebas unitarias definidas para cada módulo.
- Volère: Esta metodología de requerimientos usa formatos particulares y un ciclo de levantamiento específico que permite validar los requerimientos antes de solicitar su desarrollo, el manejo de prototipos se hace mediante lenguajes de diseño como UML o BPMN, tiene su propio grupo de formatos que pueden ser obtenidos del sitio web <http://www.volere.co.uk/>. El ciclo propuesto se muestra en la gráfica a continuación.

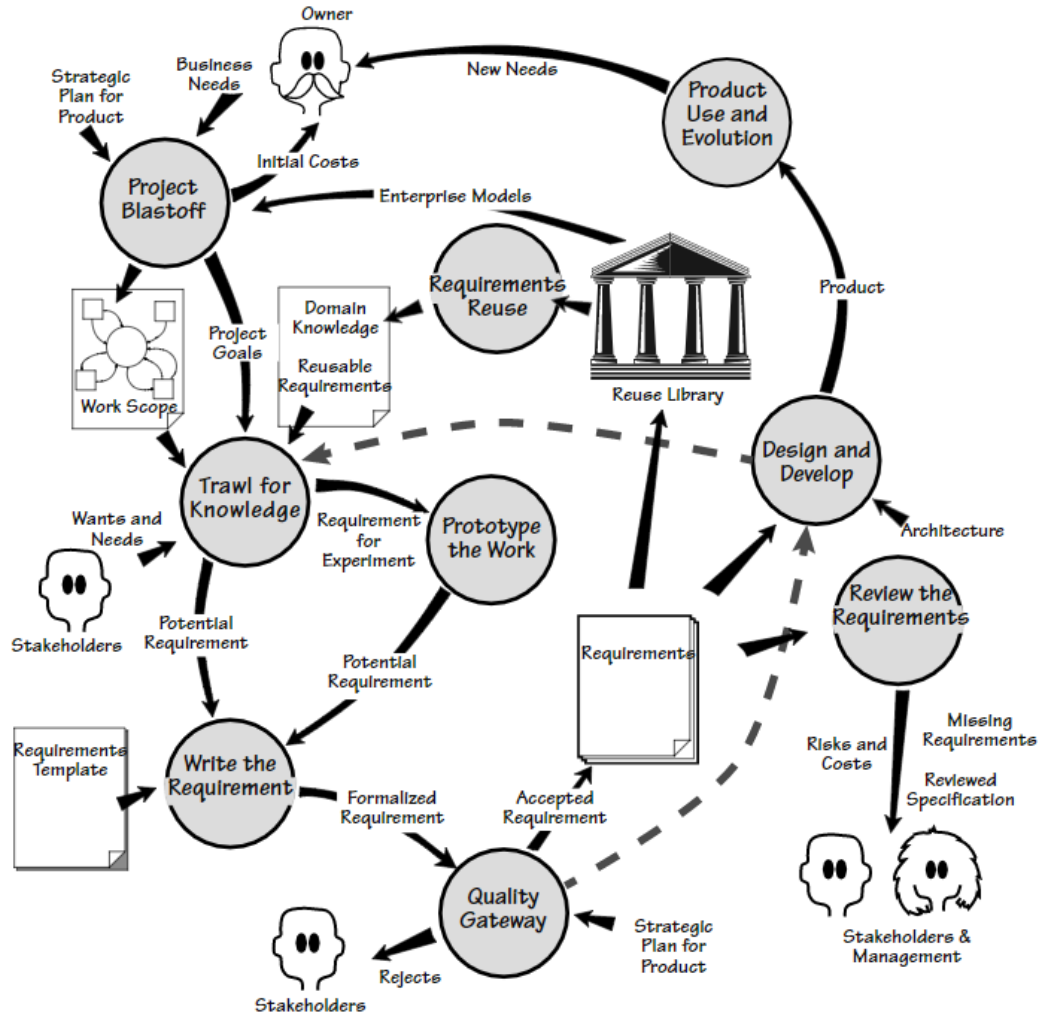


Figura 35 Ciclo de levantamiento Volère. Tomado de (Robertson & Robertson, 2013)

Conclusiones

Los patrones de proceso de desarrollo de software se han desarrollado en diferentes círculos de trabajo, pero no han tenido la difusión que tuvieron otra clase de patrones como los patrones de diseño.

Esta serie de patrones es útil cuando se hace un análisis de las variables propias de un proyecto y se determina su aplicabilidad, tomando en cuenta sus beneficios, desventajas y riesgos, no hay una receta mágica que sea compatible con todos los problemas existentes, es por esto que se hace necesario recopilarlos y estudiar su aplicabilidad, en esta monografía se generó una guía rápida que permite determinar el patrón adecuado, de acuerdo a las características del proyecto, pero es posible encontrar con la práctica, nuevos patrones adecuados a otras situaciones de proyecto, y puede ser una investigación adecuada para trabajos posteriores o un proceso de compilación constante a cargo de una entidad como Fedesoft.

En la monografía se compilaron las investigaciones de entidades públicas, especialmente del gobierno de los Estados Unidos de América y de la Universidad de Múnich, así como los trabajos de múltiples investigadores acerca de los riesgos y variables fundamentales a tener en cuenta al planear el ciclo de vida de un proyecto de desarrollo de software.

Los patrones seleccionados tienen el respaldo de su aplicación en múltiples proyectos, y tuvieron un esquema que permite claramente determinar sus ventajas, el problema que solucionan y sus consecuencias, así como un esquema claro y los modelos de desarrollo aplicables.

La aplicabilidad en el mercado colombiano, se ve ligada al hecho de que la mayoría de empresas de desarrollo de software en el medio son PyMEs, con bajo capital para invertir en procesos de calidad, o de gestión del conocimiento que les permita tener una mejora continua en la planeación de proyectos, por esto el usar patrones que permitan determinar maneras óptimas de planear las fases de un proyecto, de acuerdo a sus características y riesgos específicos, es una ventaja sin costo adicional, se hace necesario dar difusión a esta clase de patrones para los ingenieros, que en su mayoría son los generadores de esta clase de emprendimientos. En el entorno colombiano, el uso de fábricas de pruebas, que es ampliamente extendido, e incluso el uso de trabajo tercerizado con modelos como *nearshore* u *offshore* no han tenido toda la documentación y estudio que deberían tener, si consideramos que esta forma de trabajo le permite al país incursionar en mercados de alta rentabilidad y valor agregado, como lo exigen los cambios económicos globales de esta era. Adicionalmente, se usan modelos de desarrollo estándar, por lo cual la aplicabilidad de los patrones es totalmente compatible con las primeras etapas de los proyectos de desarrollo.

Las variables determinadas como fundamentales en los procesos de desarrollo de software, de acuerdo a las investigaciones consultadas fueron: número de riesgos, estabilidad de requerimientos, experiencia del equipo de desarrollo, flexibilidad al cambio en el proyecto y el involucramiento de usuarios funcionales. En esta investigación se determinaron los posibles valores de las variables para facilitar la elección de los patrones, y se relacionó el uso de los patrones a las variables, generando un cuadro con esta asociación para facilitar su consulta.

En este estudio se revisaron los riesgos más representativos para esta clase de proyectos, con lo cual se determinó que los tipos de riesgo aplicables son: De proyecto, técnicos, de negocio y de comunicación. Al igual que las variables, cada patrón tuvo un análisis sobre los riesgos con mayor probabilidad de ocurrencia.

El uso de otras metodologías complementarias permite tener un paso adelante en la implementación de los patrones, en la mayoría de los expuestos en esta monografía se enfocan en la etapa más riesgosa que afrontamos en el desarrollo de proyectos de software, el levantamiento de requerimientos, varios de ellos hacen énfasis en una parte importante que es la integración con las pruebas y el desarrollo necesario.

En investigaciones posteriores, se puede profundizar en los patrones de desarrollo con múltiples fábricas, así como la aplicación de modelos industrializados de automatización de desarrollo. En Colombia es fundamental entender la posibilidad de usar patrones de desarrollo que permitan generar cluster de empresas PyMEs, con capacidad de desarrollar proyectos de mayor riesgo y presupuesto para obtener las rentabilidades que tienen las empresas grandes.

Referencias

- Ambler, S. (1998). *Process Patterns Building Large-Scale Systems Using Object Technology*. *Software science*. Cambridge: Cambridge University Press. <http://doi.org/10.1007/s13398-014-0173-7.2>
- Arboleda, H. (2002). Modelos de Ciclo de Vida de Desarrollo de Software en el Contexto de la Industria Colombiana de Software.
- Azanza, M., Díaz, O., & Trujillo, S. (2008). *Software Factories: Describing the Assembly Process*. Mondragón, San Sebastián. España.
- Balduino, R. (2005). Basic Unified Process: A Process for Small and Agile Projects. *Rational Unified Process Content Developer, IBM*, 1–7. Retrieved from <http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Basic+Unified+Process:+A+Process+for+Small+and+Agile+Projects#0>
<http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Basic+Unified+Process:+A+Process+for+Small+and+Agile+Projects#0>
- Bartelt, A. C., Bauer, O., Beneken, G., Bergner, K., Birowicz, U., Bliß, T., ... Bayern, F. (2006). *V-Modell XT Das deutsche Referenzmodell für Systementwicklungsprojekte*. Bundesrepublik Deutschland.
- Beck, K., Andres, C., & Gamma, E. (2005). *Extreme Programming Explained: Embrace Change*. Pearson Education Inc.
- Beck, K., Beedle, M., Bennekum, A. van, Cockburn, A., Cunningham, W., Fowler, M., ... Thomas, D. (2001). Agile Manifesto. Retrieved from <http://agilemanifesto.org/>
- Bergner, K., & Rausch, A. (2002). Process Pattern Test Suite Bootstrapping, 1–5.
- Bieg, D. (Project M. I. (2014). PMI's Pulse of the Profession: Requirements Management — A

- Core Competency for Project and Program Success. *PMI's Pulse of the Profession*, 1–20.
- Boehm, B., Lane, J., Koolmanojwong, S., & Turner, R. (2014). *The incremental commitment Spiral Model* (Primera ed). Addison-Wesley Professional.
- Boehm, B. W. (1988). A Spiral Model of Software Development and Enhancement. *IEEE Computer*. <http://doi.org/10.1109/2.59>
- Booch, G., Rumbaugh, J., & Jacobson, I. (2005). *The Unified Modeling Language User Guide*. Addison-Wesley.
- Borges, P., Monteiro, P., & Machado, R. J. (2011). Tailoring RUP to small software development teams. In *Proceedings - 37th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2011* (pp. 306–309). <http://doi.org/10.1109/SEAA.2011.55>
- Boyde, J. (2012). *A Down-To-Earth Guide To SDLC Project Management: Getting your system / software development life cycle project successfully across the line using PMBOK adaptively*. (Vol. 1). <http://doi.org/10.1017/CBO9781107415324.004>
- Brooks, F. (1995). *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley.
- Bundesrepublik Deutschland. (2008). *V-Modell XT Complete*.
- Carmel, E., Espinosa, J. A., & Dubinsky, Y. (2010). “Follow the Sun” Workflow in Global Software Development. *Journal of Management Information Systems*, 27(1), 17–38. <http://doi.org/10.2753/MIS0742-1222270102>
- Casallas, R., & Arboleda, H. (2004). QualDev Process: Procesos Adaptables de Desarrollo de Software para Proyectos Ágiles, (2), 64–75.
- Centers for Medicare & Medicaid Services. (2008). Selecting a development approach. *Centers for Medicare & Medicaid Services*, 1–10. Retrieved from <http://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information->

Technology/XLC/Downloads/SelectingDevelopmentApproach.pdf

Choucair. (n.d.). Choucairtesting.com. Retrieved from <https://www.choucairtesting.com/>

Clarotesting.com. (2008). The seductive and dangerous V Model. Retrieved March 28, 2016, from <http://www.clarotesting.com/page11.htm>

Colpensiones, & Tecnocom. (2015). *Capacitación Modelo de Gobierno*. Bogotá.

Cooper-brown, B., & Infosys. (2015). *Test Factory Setup for SAP Applications*.

Díaz, M., & Ospina, M. (2014). Prospectiva 2019 - 2023 para Mipymes dedicadas al desarrollo de software por encargo en Colombia. *El Hombre Y La Máquina*, 44(1), 75–91.

Duarte, C., & Fernandes, A. (n.d.). Behavior-Driven Development.

Erni. (2015). Creating WIN-WIN situations. *Experience Reports on Management, Processes and Technology*, (65), 19.

Eveleens, J. L., & Verhoef, C. (2010). The rise and fall of the Chaos report figures. *IEEE Software*, 27(1), 30–36. <http://doi.org/10.1109/MS.2009.154>

Fedesoft, & Ministerio de Tecnologías de la Información y las comunicaciones de Colombia. (2015). *Informe De Caracterización Del Sector De Software Y Tecnologías De La Información En Colombia*. Retrieved from <http://fedesoft.org/noticias-fedesoft/disponible-estudio-de-caracterizacion-de-la-industria-del-software-colombiano/>

Fúquene, A., Castellanos, O., & Fonseca, S. (2007). Bases de la implementación de un modelo de inteligencia para fortalecer el desarrollo tecnológico de la industria del software y servicios asociados en Colombia. *Ingeniería E Investigación*, 27(3), 182–192. Retrieved from www.redalyc.org/articulo.oa?id=64327321

García, D. (2014). Metodología Cmmi-Dev En Proyectos De Desarrollo De. *Universidad Militar Nueva Granada*. Retrieved from <http://repository.unimilitar.edu.co/handle/10654/12482>

Gasca-Hurtado, P., & Manrique, B. (2013). Taxonomía de riesgos de outsourcing de software

Software outsourcing risk taxonomy. *Ingeniare, Revista Chilena de Ingeniería*, 21, 41–53.

Gilb, T. (1988). *Principles of Software Engineering Management*. Addison Wesley Longman.

González, J., Sánchez, S., & Velandia, D. (2016). *Identificación y análisis de factores de éxito en la gerencia proyectos en algunas PYMES del sector TI en Bogotá D.C. Colombia*.

ESCUELA COLOMBIANA DE INGENIERÍA JULIO GARAVITO.

Greenfield, J., & Short, K. (2003). Software Factories Assembling Applications with Patterns, Models, Frameworks and Tools. *Management*, 16–27.

Greensqa. (n.d.). greensqa.com/. Retrieved from <http://greensqa.com/>

Guntamukkala, V., Wen, H. J., & Tarn, J. M. (2006). An empirical study of selecting software development life cycle models. *Human Systems Management*, 25(4), 265–278. Retrieved from <http://content.ebscohost.com/ContentServer.asp?T=P&P=AN&K=22122914&S=R&D=bth&EbscoContent=dGJyMNHX8kSeqK44zdnyOLCmr0qeprZSr6e4S7CWxWXS&ContentCustomer=dGJyMPGosk+xq65QuePfgex44Dt6fIA\http://content.ebscohost.com/ContentServer.asp?T=P&P=AN&K=23065925&S=R>

Gupta, A., Sloan School of Management., & Seshasai, S. (2004). Toward the 24-hour knowledge factory. *Working Paper*, (4455–4), 11 leaves. <http://doi.org/10.2139/ssrn.1012847>

Hanafiah, M., & Kasirun, Z. M. (2007). Using rule-based technique in developing the tool for finding suitable software methodology. *Malaysian Journal of Computer Science*, 20(2), 209–224. Retrieved from <http://www.scopus.com/inward/record.url?eid=2-s2.0-38049100985&partnerID=tZOtx3y1>

Hendrickson, E. (2008). Driving development with tests: ATDD and TDD. *Quality Tree*

Software, Inc. Http://www. Qualitytree. Com, 1–9.

Humphrey, W. S. (2000). The Team Software Process (TSP). *Management*, (November), 32–34.

<http://doi.org/10.1002/0471028959.sof352>

IBM Corporation. (2005). Ciclo vital de RUP Planificar las fases Planificar las estrategias.

Retrieved from <https://catalyst->

epfwiki.csc.com/wikis/crup/core.base_rup/customcategories/rup_lifecycle_100BF298.html

Iida, H. (1999). Pattern-Oriented Approach to Software Process Evolution. *Proceedings of the*

International Workshop on the Retrieved from

<http://www.tarrani.net/PatternOrientedApproachSWEvolution.pdf>

Inteco, E. (Instituto N. de T. de la C. (2009). Ingeniería del Software: Metodologías y Ciclos de

Vida. *Laboratorio Nacional de Calidad Del Software*, 83. Retrieved from

https://www.incibe.es/file/N85W1ZWfHifRgUc_oY8_Xg

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The Unified Software Development Process*.

Addison-Wesley Object Technology.

JBehave. (n.d.). Retrieved from <http://jbehave.org/>

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. V., Loingtier, J.-M., & Irwin, J.

(1997). Aspect-Oriented Programming. *ACM Computing Surveys*, 28(June), 220–242.

<http://doi.org/10.1145/242224.242420>

Kroll, P., & Kruchten, P. (2003). *The Rational Unified Process Made Easy: A Practitioner's*

Guide to the RUP: A Practitioner's Guide to the RUP. Addison-Wesley Professional.

Maheshwari, S., Jain, D. C., Maheshwari, M. S., & Jain, D. C. (2012). A Comparative Analysis

of Different types of Models in Software Development Life Cycle. *International Journal of*

Advanced Research in Computer Science and Software Engineering, 2(5), 285–290.

Martínez, S., Arango, S., & Robledo, J. (2015). El crecimiento de la industria del software en Colombia: un análisis sistémico. *Revista EIA*, 12(23), 95–106.

<http://doi.org/10.14508/reia.2015.12.23.95-106>

McConnell, S. (1996). *Rapid Development*. Redmond, Washington: Microsoft Press.

Miler, J., & Gorski, J. (2004). Risk identification patterns for software projects. *Foundations of Computing and Decision Sciences*, 29(1–2), 115–132.

Mintic, & Fedesoft. (2015). *Estudio de salarios del sector de software y TI de Colombia 2015*.

Mishra, A., & Dubey, D. (2013). A Comparative Study of Different Software Development Life Cycle Models in Different Scenarios. *International Journal of Advance Research in Computer Science and Management Studies*, 1(5), 64–69.

Moreno, J., Rengifo, L., & Navia, M. (2010). Un acercamiento a las prácticas de calidad de software en las MiPyMES del suroccidente colombiano. *Revista Lasallista de Investigación*, 7(1), 17–24. Retrieved from <http://search.ebscohost.com/login.aspx?direct=true&db=aph&AN=54861616&lang=es&site=ehost-live>

Nehmer, J., Gotzhein, R., Avenhaus, J., Härder, T., Litz, L., Madlener, K., ... Zimmermann, G. (2000). Development of large systems with generic methods. Retrieved from <http://www.sfb501.uni-kl.de/>

Nomura, L., Tonini, A. C., Hikage, O. K., & Tonini, A. C. (2007). A model for defining software factory processes.

Ozturk, V. (2013). Selection of appropriate software development life cycle using fuzzy logic. *Journal of Intelligent & Fuzzy Systems*, 25(1), 797–810. <http://doi.org/10.3233/IFS-120686>

Palomino, K. (2011). *Estudio del comportamiento de la industria del software en Colombia ante*

- escenarios de capacidades de innovación y ventajas comparativas por medio de dinámica de sistemas*. UNIVERSIDAD NACIONAL DE COLOMBIA. Retrieved from <http://www.bdigital.unal.edu.co/5411/1/200802180-2011.pdf>
- Pixel Group Net. (n.d.). Pixel Group Net. Retrieved from <http://pixelgroup.net/categoria/servicios/qa/>
- Pressman, R. (2010). *Software Engineering: A Practitioner's Approach* (Seventh). New York: Mc Graw Hill.
- Pries-Heje, J., Baskerville, R., & Hansen, G. I. (2005). Strategy models for enabling offshore outsourcing: Russian short-cycle-time software development. *Information Technology for Development, 11*(1), 5–30. <http://doi.org/10.1002/itdj.20001>
- Prikladnicki, R., & Audy, J. (2012). Managing Global Software Engineering: A Comparative Analysis of Offshore Outsourcing and the Internal Offshoring of Software Development. *Information Systems Management, 29*, 216–232. <http://doi.org/10.1080/10580530.2012.687313>
- Pugh, K. (2011). *Lean - Agile Acceptance Test-Driven Development*. Addison-Wesley.
- Rastogi, V. (2015). Software Development Life Cycle Models- Comparison , Consequences. *International Journal of Computer Science and Information Technologies, 6*(1), 168–172.
- Robertson, S., & Robertson, J. (2013). *Mastering the Requirements Process: Getting Requirements Right*. Pearson Education Inc.
- Royce, W. (1970). Managing the Development of large Software Systems. *IEEE Wescon*, (August), 1–9.
- Royce, W. (1998). *Software Project Management*. Addison-Wesley.
- Satpathy, T. (2016). *Una guía para el Conocimiento de SCRUM (Guia SBOK)*. Statewide

- Agricultural Land Use Baseline 2015* (Vol. 1). Retrieved from www.scrumstudy.com
- Schroeder, P. J., & Rothe, D. (2005). Teaching Unit Testing using Test-Driven Development Workshop on Teaching Software Testing 2005. *4th Workshop on Teaching Software Testing, Melbourne, Florida.*
- Schwaber, K., & Sutherland, J. (2011). The scrum guide. *Scrum. Org, October, 2(July)*, 17. <http://doi.org/10.1053/j.jrn.2009.08.012>
- Seema, & Malhotra, S. (2015). Analysis and tabular comparison of popular SDLC models. *International Journal of Advanced Research in Computer Science and Software Engineering*, 6(2277), 168–172. <http://doi.org/10.6088/ijacit.12>.
- Soares, S., & Borba, P. (2002). PIP: Progressive implementation pattern. *1st Workshop on Software Development Process Patterns (OOPSLA02)*, 1–6.
- SQA S.A. (n.d.). <http://www.sqasa.com/>.
- Stahl, T., Völter, M., Bettin, J., Haase, A., & Helsen, S. (2006). *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, Ltd.
- Takeuchi, H., & Nonaka, I. (1986). The new product development game. *Journal of Product Innovation Management*, 3(3), 205–206. [http://doi.org/10.1016/0737-6782\(86\)90053-6](http://doi.org/10.1016/0737-6782(86)90053-6)
- Tao, H. (2006). *Mastering Software Project Management: Managing Software Projects According to The Art Of Business*. Lulu.com.
- Technische Universität München. (2002). Proceedings of the 1st Workshop on Software Development Patterns (SDPP'02). In *TUM. München: Institut für Informatik*.
- Tecnosfera. (2015). Crece la industria de “software” 100% colombiano. *El Tiempo*. Retrieved from <http://www.eltiempo.com/tecnosfera/novedades-tecnologia/aumento-de-la-industria-de-software-colombiano/15445677>

- Tesanovic, A. (2001). *What is a pattern? Linkoping University*. Retrieved from <http://www.cs.duke.edu/courses/cps108/compsci308/cps108/fall04/notes/06patterns.pdf>
- The Standish Group. (1995). The Standish group: the chaos report. *Project Smart*, 16.
- The Standish Group International. (2013). CHAOS MANIFESTO 2013: Think Big, Act Small. *The Standish Group International*, 1–52. Retrieved from <http://www.standishgroup.com>
- The US Department of Justice. (2003). Systems Development Life Cycle Guidance Document. Retrieved from <http://www.justice.gov/archive/jmd/irm/lifecycle/table.htm>
- Thomas, C. (2010). Software Commoditization and Open Source Strategies. *Strategies*, (July), 1–25.
- Unctad, & Ki-moon, B. a N. (2012). *Information economy report 2012: the software industry and developing countries*.
- United States of America Department of Defense. (1994). *Software Development and Documentation. Military Standard* (Vol. 1994). Retrieved from [ftp://121.241.180.136/Copy of rpm2/Uploads/_634405360523877505_498-std.pdf](ftp://121.241.180.136/Copy%20of%20rpm2/Uploads/_634405360523877505_498-std.pdf)
- Vaghela, R. K. (2015). A Comparative Analysis of Software development life cycle Models. *INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH*, 4(6), 512–515. <http://doi.org/10.17148/IJARCCE.2016.5246>

LICENCIA DE USO – AUTORIZACIÓN DE LOS AUTORES

Actuando en nombre propio identificado (s) de la siguiente forma:

Nombre Completo Jorge Iván Alvarez Garcia

Tipo de documento de identidad: C.C. T.I. C.E. Número: 79691279

Nombre Completo _____

Tipo de documento de identidad: C.C. T.I. C.E. Número: _____

Nombre Completo _____

Tipo de documento de identidad: C.C. T.I. C.E. Número: _____

Nombre Completo _____

Tipo de documento de identidad: C.C. T.I. C.E. Número: _____

El (Los) suscrito(s) en calidad de autor (es) del trabajo de tesis, monografía o trabajo de grado, documento de investigación, denominado:

Catálogo de Patrones de ciclo de vida en desarrollo de
Software aplicables en la Industria Colombiana

Dejo (dejamos) constancia que la obra contiene información confidencial, secreta o similar: SI NO
(Si marqué (marcamos) SI, en un documento adjunto explicaremos tal condición, para que la Universidad EAN mantenga restricción de acceso sobre la obra).

Por medio del presente escrito autorizo (autorizamos) a la Universidad EAN, a los usuarios de la Biblioteca de la Universidad EAN y a los usuarios de bases de datos y sitios webs con los cuales la Institución tenga convenio, a ejercer las siguientes atribuciones sobre la obra anteriormente mencionada:

- A. Conservación de los ejemplares en la Biblioteca de la Universidad EAN.
- B. Comunicación pública de la obra por cualquier medio, incluyendo Internet
- C. Reproducción bajo cualquier formato que se conozca actualmente o que se conozca en el futuro
- D. Que los ejemplares sean consultados en medio electrónico
- E. Inclusión en bases de datos o redes o sitios web con los cuales la Universidad EAN tenga convenio con las mismas facultades y limitaciones que se expresan en este documento
- F. Distribución y consulta de la obra a las entidades con las cuales la Universidad EAN tenga convenio

Con el debido respeto de los derechos patrimoniales y morales de la obra, la presente licencia se otorga a título gratuito, de conformidad con la normatividad vigente en la materia y teniendo en cuenta que la Universidad EAN busca difundir y promover la formación académica, la enseñanza y el espíritu investigativo y emprendedor.

Manifiesto (manifestamos) que la obra objeto de la presente autorización es original, el (los) suscritos es (son) el (los) autor (es) exclusivo (s), fue producto de mi (nuestro) ingenio y esfuerzo personal y la realizó (zamos) sin violar o usurpar derechos de autor de terceros, por lo tanto la obra es de exclusiva autoría y tengo (tenemos) la titularidad sobre la misma. En vista de lo expuesto, asumo (asumimos) la total responsabilidad sobre la elaboración, presentación y contenidos de la obra, eximiendo de cualquier responsabilidad a la Universidad EAN por estos aspectos.

En constancia suscribimos el presente documento en la ciudad de Bogotá D.C.,

NOMBRE COMPLETO: <u>Jorge Iván Álvarez G.</u>	NOMBRE COMPLETO: _____
FIRMA: <u>[Firma]</u>	FIRMA: _____
DOCUMENTO DE IDENTIDAD: <u>79691279</u>	DOCUMENTO DE IDENTIDAD: _____
FACULTAD: <u>Estudios en Ambientes Virtuales</u>	FACULTAD: _____
PROGRAMA ACADÉMICO: <u>Maestría en Gerencia de Sistemas de Información y proyectos tecnológicos</u>	PROGRAMA ACADÉMICO: _____
NOMBRE COMPLETO: _____	NOMBRE COMPLETO: _____
FIRMA: _____	FIRMA: _____
DOCUMENTO DE IDENTIDAD: _____	DOCUMENTO DE IDENTIDAD: _____
FACULTAD: _____	FACULTAD: _____
PROGRAMA ACADÉMICO: _____	PROGRAMA ACADÉMICO: _____

Fecha de firma: 11-11-2016