UNIDAD
1

INGENIERÍA DE SOFTWARE I

TEMA 2:
Modelos del Proceso de Software

FR0018/ v1.0 / 18-11-2019

UNEMI ONLINE

ÍNDICE

1. Unidad 1: La ingeniería de Software y los modelos del proceso

Tema 2: Modelos del Proceso de Software

Objetivo:

Comprender las actividades de ingeniería que componen el proceso del software y sus modelos de proceso.

Introducción:

El desarrollo de software comprende actividades debidamente establecidas por una metodología propia de su desarrollo, dichas actividades forman parte del llamado proceso de software, el cual está compuesto por algunos modelos genéricos con el fin de simplificar los procesos orientados a la producción de software de calidad.

Los modelos de procesos, también conocidos como paradigmas de proceso, son caracterizados desde diferentes visiones o perspectivas arquitectónicas de otros modelos, sin embargo, es necesario precisar que dichos modelos no presentan descripciones definitivas de los procesos de software. Es preciso señalar que no existe un proceso ideal, depende del tipo de sistema y desde un punto de vista perfectible es posible dirigir un proyecto de software con nuevos métodos y técnicas.

2. Información de los subtemas

2.1 Subtema 1: Modelos de proceso prescriptivo

Antes de exponer cada uno de los tipos modelos de proceso prescriptivo, llamados también tradicional, es necesario señalar el origen, o dicho de mejor forma, la motivación que originó la propuesta de estos modelos para desarrollo de un proyecto de software.

> Los modelos de proceso prescriptivo fueron propuestos originalmente para poner orden en el caos del desarrollo de software. La historia indica que estos modelos tradicionales han dado cierta estructura útil al trabajo de ingeniería de software y que constituyen un mapa razonablemente eficaz para los equipos de software. Sin embargo, el trabajo de ingeniería de software y el producto que genera siguen "al borde del caos". (Pressman, 2010, pág. 33)

En ese contexto, se describen a continuación los modelos de proceso de software, cada uno con actividades estructurales y diferentes tipos de énfasis:

Modelo de la cascada

A continuación, se detallan algunas definiciones para el modelo en mención:

"Toma las actividades fundamentales del proceso de especificación, desarrollo, validación y evolución y, luego, los representa como fases separadas del proceso, tal como especificación de requerimientos, diseño de software, implementación, pruebas, etcétera" (Sommerville, 2011, pág. 29).

El Laboratorio Nacional de Calidad del Software (2009) señala: "es un enfoque metodológico que ordena rigurosamente las etapas del ciclo de vida del software, de forma que el inicio de cada etapa debe esperar a la finalización de la inmediatamente anterior" (pág. 25).

"El modelo en cascada es un ejemplo de un proceso dirigido por un plan; en principio, usted debe planear y programar todas las actividades del proceso, antes de comenzar a trabajar con ellas" (Sommerville, 2011, pág. 30).

Pressman (2010) añade lo siguiente del modelo en cascada:

> Sugiere un enfoque sistemático y secuencial para el desarrollo del software, que comienza con la especificación de los requerimientos por parte del cliente y avanza a través de planeación, modelado, construcción y despliegue, para concluir con el apoyo del software terminado. (pág. 34)

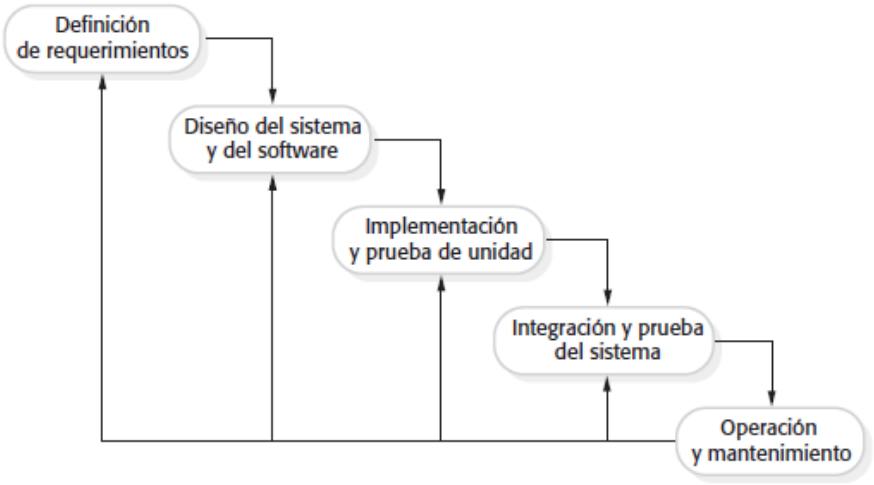

Figura 1. El modelo en cascada
Fuente: Ingeniería de software, (Sommerville, 2011, pág. 30)

Por su parte Sommerville (2011) destaca al modelo en cascada como "un ejemplo de un proceso dirigido por un plan; en principio, usted debe planear y programar todas las actividades del proceso, antes de comenzar a trabajar con ellas" (pág. 30), puntualizando de la siguiente manera las etapas de este modelo:

1. Análisis y definición de requerimientos. Los servicios, las restricciones y las metas del sistema se establecen mediante consulta a los usuarios del sistema. Luego, se definen con detalle y sirven como una especificación del sistema.
2. Diseño del sistema y del software. El proceso de diseño de sistemas asigna los requerimientos, para sistemas de hardware o de software, al establecer una arquitectura de sistema global. El diseño del software implica identificar y describir las abstracciones fundamentales del sistema de software y sus relaciones.

3. Implementación y prueba de unidad. Durante esta etapa, el diseño de software se realiza como un conjunto de programas o unidades del programa. La prueba de unidad consiste en verificar que cada unidad cumpla con su especificación.
4. Integración y prueba de sistema. Las unidades del programa o los programas individuales se integran y prueban como un sistema completo para asegurarse de que se cumplan los requerimientos de software. Después de probarlo, se libera el sistema de software al cliente.
5. Operación y mantenimiento. Por lo general (aunque no necesariamente), ésta es la fase más larga del ciclo de vida, donde el sistema se instala y se pone en práctica. El

mantenimiento incluye corregir los errores que no se detectaron en etapas anteriores del ciclo de vida, mejorar la implementación de las unidades del sistema e incrementar los servicios del sistema conforme se descubren nuevos requerimientos.

Modelos de proceso incremental

"El desarrollo incremental se basa en la idea de diseñar una implementación inicial, exponer ésta al comentario del usuario, y luego desarrollarla en sus diversas versiones hasta producir un sistema adecuado" (Sommerville, 2011, pág. 32).

El Laboratorio Nacional de Calidad del Software (2009) sostiene:

> El modelo incremental combina elementos del modelo en cascada con la filosofía interactiva de construcción de prototipos. Se basa en la filosofía de construir incrementando las funcionalidades del programa. Este modelo aplica secuencias lineales de forma escalonada mientras progresa el tiempo en el calendario. Cada secuencia lineal produce un incremento del software.

Siendo así, Pressman asegura: "El modelo incremental aplica secuencias lineales en forma escalonada a medida que avanza el calendario de actividades. Cada secuencia lineal produce "incrementos" de software susceptibles de entregarse de manera parecida a los incrementos producidos en un flujo de proceso evolutivo" (Pressman, 2010, pág. 35).

> Cuando se utiliza un modelo incremental, es frecuente que el primer incremento sea el producto fundamental. Es decir, se abordan los requerimientos básicos, pero no se proporcionan muchas características suplementarias (algunas conocidas y otras no). El cliente usa el producto fundamental (o lo somete a una evaluación detallada). (Pressman, 2010, pág. 35)

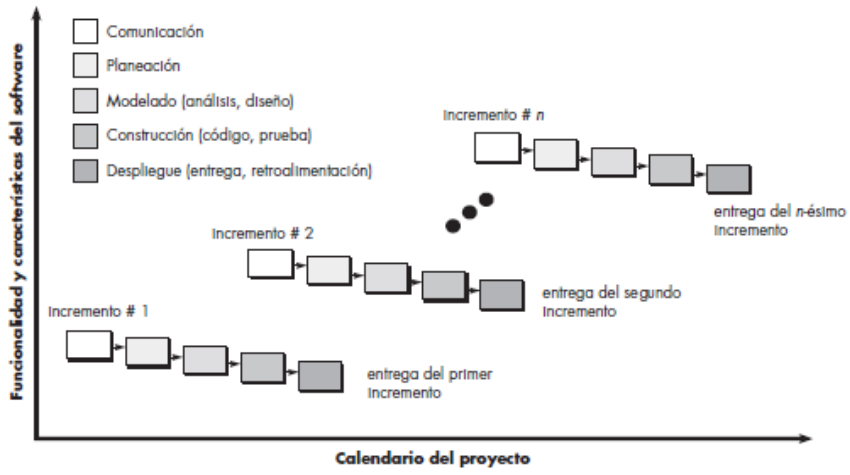

Figura 2. El modelo incremental
Fuente: Ingeniería de software: Un enfoque práctico, (Pressman, 2010, pág. 36)

> El desarrollo incremental es útil en particular cuando no se dispone de personal para la implementación completa del proyecto en el plazo establecido por el negocio. Los primeros incrementos se desarrollan con pocos trabajadores. Si el producto básico es bien recibido, entonces se agrega más personal (si se requiere) para que labore en el siguiente incremento. Además, los incrementos se planean para administrar riesgos técnicos. (Pressman, 2010, pág. 36)

Modelos de proceso evolutivo

"Los modelos evolutivos son iterativos. Se caracterizan por la manera en la que permiten desarrollar versiones cada vez más completas del software" (Pressman, 2010, pág. 36).

Se presentan a continuación dos modelos del proceso evolutivo:

Hacer prototipos

"Un prototipo es una versión inicial de un sistema de software que se usa para demostrar conceptos, tratar opciones de diseño y encontrar más sobre el problema y sus posibles soluciones" (Sommerville, 2011, pág. 45).

> El paradigma de construcción de prototipos comienza con la recolección de requisitos. El desarrollador y el cliente encuentran y definen los objetivos globales para el software, identifican los requisitos conocidos y las áreas del esquema en donde es obligatoria más definición. Entonces aparece un diseño rápido. (Laboratorio Nacional de Calidad del Software, 2009, pág. 31)

En similar idea, Pressman (2010) afirma:

> El paradigma de hacer prototipos comienza con comunicación (figura 3). Usted se reúne con otros participantes para definir los objetivos generales del software, identifica cualesquiera requerimientos que conozca y detecta las áreas en las que es imprescindible una mayor definición. Se planea rápidamente una iteración para hacer el prototipo, y se lleva a cabo el modelado (en forma de un "diseño rápido"). Éste se centra en la representación de aquellos aspectos del software que serán visibles para los usuarios finales (por ejemplo, disposición de la interfaz humana o formatos de la pantalla de salida). (Pressman, 2010, pág. 37)

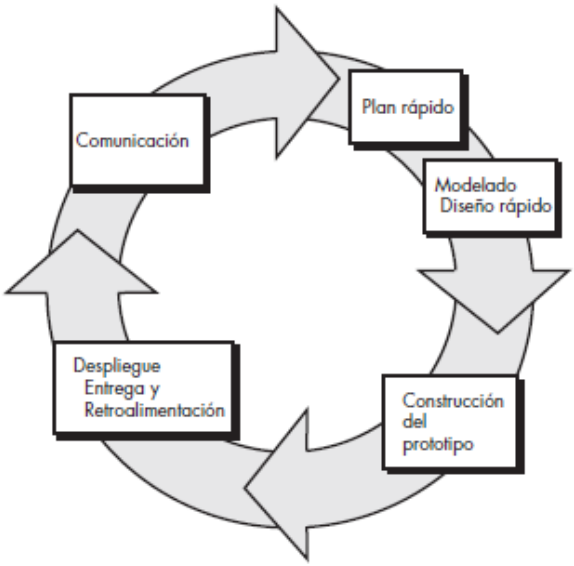

Figura 3. El paradigma de hacer prototipos
Fuente: Ingeniería de software: Un enfoque práctico, (Pressman, 2010, pág. 37)

> El diseño rápido lleva a la construcción de un prototipo. Éste se entrega y es evaluado por los participantes, que dan retroalimentación para mejorar los requerimientos. La iteración ocurre a medida de que el prototipo es afinado para satisfacer las necesidades de distintos participantes, y al mismo tiempo le permite a usted entender mejor lo que se necesita hacer. (Pressman, 2010, pág. 37)

El paradigma de hacer prototipos es apetecido por los ingenieros de software y los participantes involucrados en el proyecto, la sensación del sistema real es tomada por los usuarios, mientras que los desarrolladores logran construir de inmediato, sin embargo, la elaboración de prototipos tiende a ocasionar problemas por las causas siguientes:

1. Los participantes ven lo que parece ser una versión funcional del software, sin darse cuenta de que el prototipo se obtuvo de manera caprichosa; no perciben que en la prisa por hacer que funcionara, usted no consideró la calidad general del software o la facilidad de darle mantenimiento a largo plazo. Cuando se les informa que el producto debe rehacerse a fin de obtener altos niveles de calidad, los participantes gritan que es usted un tonto y piden “unos cuantos arreglos” para hacer del prototipo un producto funcional. Con demasiada frecuencia, el gerente de desarrollo del software cede.
2. Como ingeniero de software, es frecuente que llegue a compromisos respecto de la implementación a fin de hacer que el prototipo funcione rápido. Quizá utilice un sistema operativo inapropiado, o un lenguaje de programación tan sólo porque cuenta con él y lo conoce; tal vez implementó un algoritmo ineficiente sólo para demostrar capacidad. Después de un tiempo, quizá se sienta cómodo con esas elecciones y olvide todas las razones por las que eran inadecuadas. La elección de

algo menos que lo ideal ahora ha pasado a formar parte del sistema. (Pressman, 2010, pág. 38)

El modelo espiral

Según Boehm el modelo espiral es un modelo evolutivo del proceso del software y se acopla con la naturaleza iterativa de hacer prototipos con los aspectos controlados y sistémicos del modelo de cascada. Tiene el potencial para hacer un desarrollo rápido de versiones cada vez más completas. Boehm describe el modelo del modo siguiente:

El modelo de desarrollo espiral es un generador de modelo de proceso impulsado por el riesgo, que se usa para guiar la ingeniería concurrente con participantes múltiples de sistemas intensivos en software. Tiene dos características distintivas principales. La primera es el enfoque cíclico para el crecimiento incremental del grado de definición de un sistema y su implementación, mientras que disminuye su grado de riesgo. La otra es un conjunto de puntos de referencia de anclaje puntual para asegurar el compromiso del participante con soluciones factibles y mutuamente satisfactorias.

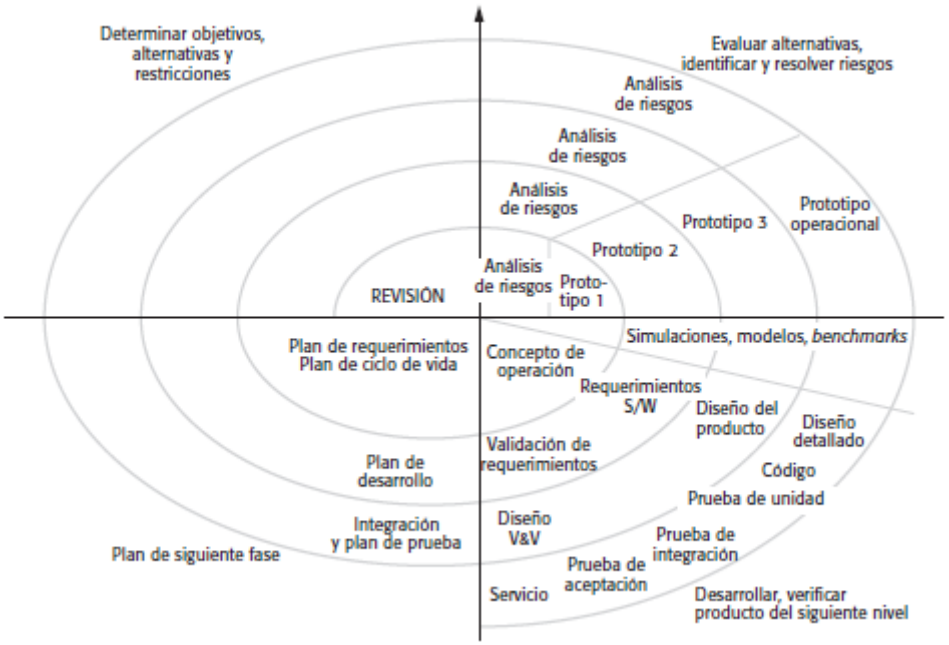

Figura 4. Modelo en espiral de Boehm del proceso de software
Fuente: Ingeniería de software: Un enfoque práctico, (Sommerville, 2011, pág. 49)

"Las actividades de este modelo se conforman en una espiral, cada bucle representa un conjunto de actividades (…) no están fijadas a priori, sino que las siguientes se eligen en función del análisis de riesgos, comenzando por el bucle anterior". (Laboratorio Nacional de Calidad del Software, 2009, pág. 29)

> Al ser un modelo de ciclo de vida orientado a la gestión de riesgos se dice que uno de los aspectos fundamentales de su éxito radica en que el equipo que lo aplique tenga la necesaria experiencia y habilidad para detectar y catalogar correctamente riesgos. (Laboratorio Nacional de Calidad del Software, 2009, pág. 29)

Según Sommerville (2011), cada ciclo en la espiral se divide en cuatro sectores:

1. Establecimiento de objetivos. Se definen objetivos específicos para dicha fase del proyecto. Se identifican restricciones en el proceso y el producto, y se traza un plan de gestión detallado. Se identifican los riesgos del proyecto. Pueden planearse estrategias alternativas, según sean los riesgos.

2. Valoración y reducción del riesgo. En cada uno de los riesgos identificados del proyecto, se realiza un análisis minucioso. Se dan acciones para reducir el riesgo. Por ejemplo, si existe un riesgo de que los requerimientos sean inadecuados, puede desarrollarse un sistema prototipo.

3. Desarrollo y validación. Después de una evaluación del riesgo, se elige un modelo de desarrollo para el sistema. Por ejemplo, la creación de prototipos desechables sería el mejor enfoque de desarrollo, si predominan los riesgos en la interfaz del usuario. Si la principal consideración son los riesgos de seguridad, el desarrollo con base en transformaciones formales sería el proceso más adecuado, entre otros. Si el principal riesgo identificado es la integración de subsistemas, el modelo en cascada sería el mejor modelo de desarrollo a utilizar.

4. Planeación. El proyecto se revisa y se toma una decisión sobre si hay que continuar con otro ciclo de la espiral. Si se opta por continuar, se trazan los planes para la siguiente fase del proyecto.

> El primer circuito alrededor de la espiral da como resultado el desarrollo de una especificación del producto; las vueltas sucesivas se usan para desarrollar un prototipo y, luego, versiones cada vez más sofisticadas del software. Cada paso por la región de planeación da como resultado ajustes en el plan del proyecto. El costo y la programación de actividades se ajustan con base en la retroalimentación obtenida del cliente después de la entrega. (Pressman, 2010, pág. 40)

Modelos concurrentes

"El modelo de desarrollo concurrente, en ocasiones llamado ingeniería concurrente, permite que un equipo de software represente elementos iterativos y concurrentes de cualquiera de los modelos de proceso" (Pressman, 2010, pág. 40).

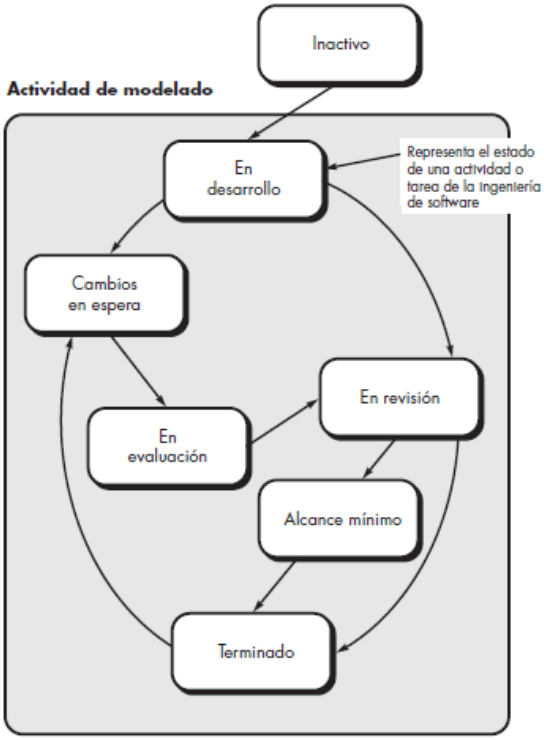

Figura 5. Modelo de proceso concurrente
Fuente: Ingeniería de software: Un enfoque práctico, (Pressman, 2010, pág. 41)

La figura 5 muestra la representación esquemática de una actividad de ingeniería de software
dentro de la actividad de modelado con el uso del enfoque de modelado concurrente. La actividad —modelado— puede estar en cualquiera de los estados mencionados en un momento dado. En forma similar, es posible representar de manera análoga otras actividades, acciones o tareas (por ejemplo, comunicación o construcción). Todas las actividades de ingeniería de software existen de manera concurrente, pero se hallan en diferentes estados. (Pressman, 2010, pág. 41)

"El modelado concurrente define una serie de eventos que desencadenan transiciones de un estado a otro para cada una de las actividades, acciones o tareas de la ingeniería de software" (Pressman, 2010, pág. 42).

El modelado concurrente es aplicable a todos los tipos de desarrollo de software y proporciona un panorama apropiado del estado actual del proyecto. En lugar de confinar las actividades, acciones y tareas de la ingeniería de software a una secuencia de eventos, define una red del proceso. Cada actividad, acción o tarea de la red existe simultáneamente con otras actividades, acciones o tareas. Los eventos generados en cierto punto de la red del proceso desencadenan transiciones entre los estados. (Pressman, 2010, pág. 42)

2.2 Subtema 2: Modelos de proceso especializado

"Los modelos de proceso especializado tienen muchas de las características de uno o más de los modelos tradicionales (…) Sin embargo, dichos modelos tienden a aplicarse cuando se elige un enfoque de ingeniería de software especializado" (Pressman, 2010, pág. 43).

Desarrollo basado en componentes

"El concepto de componentes para el desarrollo de software no es un concepto nuevo; para muchos autores simplemente es la evolución de la metodología orientada a objetos" (Troya, Vallecillo, & Fuentes, 2017)

"El modelo de desarrollo basado en componentes incorpora muchas de las características del modelo espiral. Es de naturaleza evolutiva y demanda un enfoque iterativo para la creación de software (…) construye aplicaciones a partir de fragmentos de software prefabricados" (Pressman, 2010, pág. 43).

> El DSBC trata de sentar las bases para el diseño y desarrollo de aplicaciones distribuidas basadas en componentes software reutilizables. Dicha disciplina cuenta actualmente con un creciente interés, tanto desde el punto de vista académico como desde la industria, en donde la demanda de mecanismos y herramientas de desarrollo basados en componentes es cada día mayor. (Bertoa, Troya, & Vallecillo, 2002)

Pressman (2010) menciona que el modelo de desarrollo basado en componentes incorpora las etapas siguientes:

1. Se investigan y evalúan, para el tipo de aplicación de que se trate, productos disponibles
basados en componentes.
2. Se consideran los aspectos de integración de los componentes.
3. Se diseña una arquitectura del software para que reciba los componentes.
4. Se integran los componentes en la arquitectura.
5. Se efectúan pruebas exhaustivas para asegurar la funcionalidad apropiada.

> El modelo del desarrollo basado en componentes lleva a la reutilización del software, y eso da a los ingenieros de software varios beneficios en cuanto a la mensurabilidad. Si la reutilización de componentes se vuelve parte de la cultura, el equipo de ingeniería de software tiene la posibilidad tanto de reducir el ciclo de tiempo del desarrollo como el costo del proyecto. (Pressman, 2010, pág. 43)

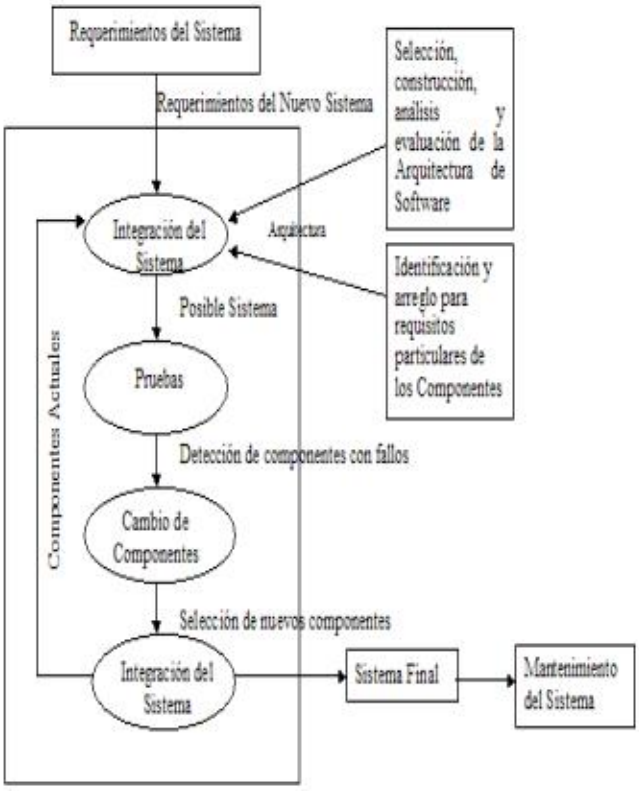

Figura 6. Modelo del Ciclo de vida de DSBC
Fuente: Aspectos de calidad en el desarrollo de software basado en componentes, (Bertoa, Troya, & Vallecillo, 2002)

Según Rojas & García (2004), las aactividades que involucra el modelo del ciclo de vida para DSBC (figura 6), son las siguientes:

Análisis de Requerimientos: En esta etapa del ciclo de vida los procesos y las necesidades del negocio se descubren y se expresan en los casos de uso.
Selección, construcción, análisis y evaluación de la Arquitectura de Software: "La arquitectura del software define un sistema en términos de componentes computacionales y la interacción entre ellos".
Identificación y arreglo para requisitos particulares del Componente: En esta actividad, los componentes deben ser seleccionados por los requerimientos funcionales y de calidad que satisfaga cada componente.
Integración del Sistema: En esta actividad se debe examinar, evaluar y determinar cómo va a ser la comunicación y la coordinación entre los componentes que harán parte del sistema.
Pruebas: Posterior a la integración de los componentes se debe proceder con las pruebas, esto implica evaluar el funcionamiento de los componentes que fueron integrados en el sistema, si algún componente demuestra no estar funcionando de forma correcta se debe pensar en la posibilidad de reemplazarlo o modificarlo para luego proceder con la re-integración.

Mantenimiento: En el período del mantenimiento, se lleva a cabo un proceso similar al desarrollado en la POO, esto es vigilar el correcto funcionamiento del sistema, corregir fallas en el comportamiento, etc.

El modelo de métodos formales

Agrupa actividades que llevan a la especificación matemática formal del software de cómputo. Los métodos formales permiten especificar, desarrollar y verificar un sistema basado en computadora por medio del empleo de una notación matemática rigurosa. (Pressman, 2010, pág. 44)

> En Ingeniería de Software, un método formal es un proceso que se aplica para desarrollar programas, y que explota el poder de la notación y de las pruebas matemáticas. Además, los requisitos, la especificación y el sistema completo deben validarse con las necesidades del mundo real. (Kuhn, Chandramouli, & Butler, 2002)

> Los métodos formales permiten representar la especificación del software, verificación y diseño de componentes mediante notaciones matemáticas. El uso de métodos formales permite plantear de manera clara la especificación de un sistema, generando modelos que definen el comportamiento en términos del "qué debe hacer" y no del "cómo lo hace". (Fernández y Fernández, 2011)

"Cuando durante el desarrollo se usan métodos formales, se obtiene un mecanismo para eliminar muchos de los problemas difíciles de vencer con otros paradigmas de la ingeniería de software" (Pressman, 2010, pág. 44).

Aunque el modelo de los métodos formales no es el más seguido, promete un software libre de defectos. Sin embargo, se han expresado preocupaciones acerca de su aplicabilidad en un ambiente de negocios:

- El desarrollo de modelos formales consume mucho tiempo y es caro.
- Debido a que pocos desarrolladores de software tienen la formación necesaria para aplicar métodos formales, se requiere mucha capacitación.
- Es difícil utilizar los modelos como mecanismo de comunicación para clientes sin complejidad técnica. (Pressman, 2010, pág. 44)

Desarrollo de software orientado a aspectos

Sin importar el proceso del software que se elija, los constructores de software complejo implementan de manera invariable un conjunto de características, funciones y contenido de información localizados. Estas características localizadas

del software se modelan como componentes (clases orientadas a objetos) y luego se construyen dentro del contexto de una arquitectura de sistemas. (Pressman, 2010, pág. 44)

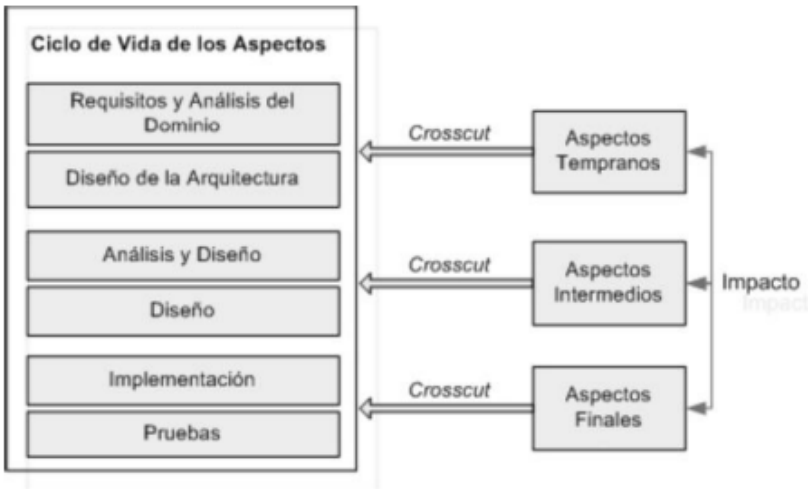

Figura 7. Ciclo de vida del Desarrollo de Software Orientado a Aspectos
Fuente: El desarrollo de software Orientado a Aspectos (Tabares, Alferez Salinas, & Alferéz Salinas, 2008)

El desarrollo de software orientado a aspectos (DSOA), conocido también como programación orientada a aspectos (POA), es un paradigma de ingeniería de software relativamente nuevo que proporciona un proceso y enfoque metodológico para definir, especificar, diseñar y construir aspectos: "mecanismos más allá de subrutinas y herencia para localizar la expresión de una preocupación global" (Pressman, 2010, pág. 44)

La figura 7 ilustra la forma como los aspectos evolucionan a través de todo el ciclo de vida de desarrollo de software. Los aspectos se clasifican de la siguiente forma: (1) aspectos tempranos, los cuales se especifican desde los requisitos hasta la arquitectura; (2) aspectos intermedios, los cuales se especifican en estructuras aspectuales al nivel del diseño; (3) aspectos finales, los cuales se especifican en lenguajes de programación especializados. (Tabares, Alferez Salinas, & Alferéz Salinas, 2008)

2.3 Subtema 3: El Proceso unificado

"El proceso unificado es un intento por obtener los mejores rasgos y características de los modelos tradicionales del proceso del software, pero en forma que implemente muchos de los mejores principios del desarrollo ágil de software" (Pressman, 2010, pág. 46). Rumbaugh, Booch y Jacobson trabajaron juntos para crear un "método unificado", con el objeto de combinar la mejor parte de cada uno de los modelos de software, es decir sus métodos individuales de análisis y diseño orientado a objetos. El resultado fue el lenguaje de modelado unificado (UML), el cual incluye una notación robusta para el modelado y desarrollo de los sistemas orientados a objetos.
Pressman (2010) agrega: "El UML brinda la tecnología necesaria para apoyar la práctica de la ingeniería de software orientada a objetos, pero no da la estructura del proceso que guíe a los equipos del proyecto cuando aplican la tecnología" (pág. 46).

"El proceso unificado (PU) y el UML se usan mucho en proyectos de toda clase orientados a objetos. El modelo iterativo e incremental propuesto por el PU puede y debe adaptarse para que satisfaga necesidades específicas del proyecto" (Pressman, 2010, pág. 46).

Fases de proceso unificado

Teniendo en cuenta los aspectos mencionados previamente, Rational que recientemente fue comprada por IBM, elaboró un marco de referencia para el proceso de desarrollo de software basado en el modelo en espiral. Este método se conoce como RUP "Rational Unified Process" (figura 9). Para una mejor organización, el RUP agrupa las iteraciones en etapas y fases que facilitan la administración del proyecto. (Gil, 2004)

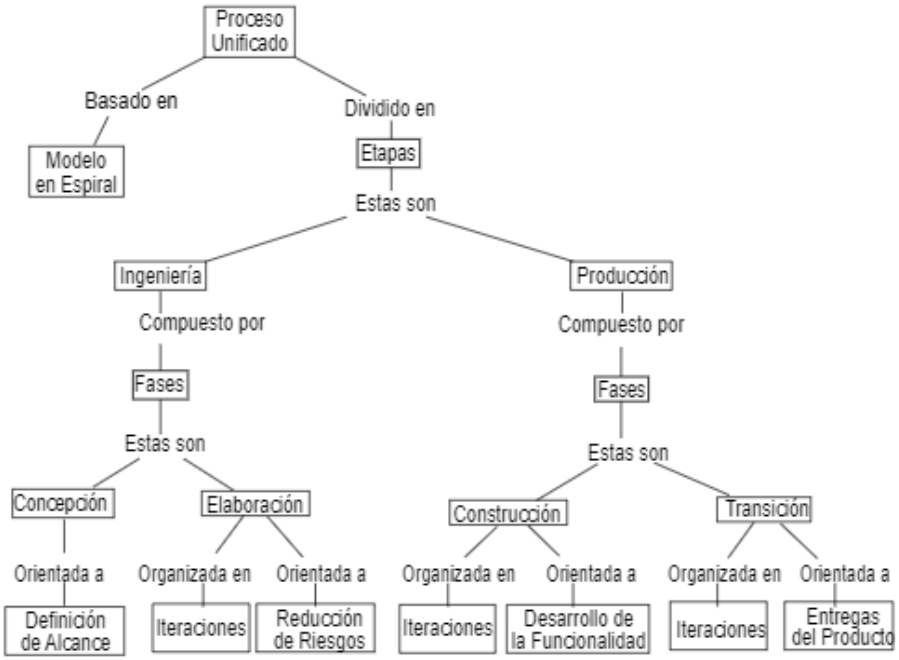

Figura 9. Mapa conceptual del Proceso Unificado de Rational
Fuente: Ingeniería de software: Un enfoque práctico (Pressman, 2010, pág. 47)

Etapa de ingeniería

Esta etapa agrupa las fases de concepción y de elaboración, lo que básicamente le da por objetivos la conceptualización del sistema y el diseño inicial de la solución del problema.

Fase de concepción

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

Fase de elaboración

Los casos de uso seleccionados para desarrollarse en esta fase permiten definir la arquitectura del sistema, se realiza la especificación de los casos de uso seleccionados y el primer análisis del dominio del problema, se diseña la solución preliminar del problema y comienza la ejecución del plan de manejo de riesgos, según las prioridades definidas en él.

Etapa de producción

En esta etapa se realiza un proceso de refinamiento de las estimaciones de tiempos y recursos para las fases de construcción y transición, se define un plan de mantenimiento para los productos entregados en la etapa de ingeniería, se

implementan los casos de uso pendientes y se entrega el producto al cliente, garantizan-do la capacitación y el soporte adecuados.

Fase de construcción

El propósito de esta fase es completar la funcionalidad del sistema, para ello se deben clarificar los requerimientos pendientes, administrar el cambio de los artefactos construidos, ejecutar el plan de administración de recursos y mejoras en el proceso de desarrollo para el proyecto.

Fase de transición

El propósito de esta fase es asegurar que el software esté disponible para los usuarios finales, ajustar los erro-res y defectos encontrados, capacitara los usuarios y proveer el soporte técnico necesario. Se debe verificar que el producto cumpla con las especificaciones entregadas por las personas involucradas en el proyecto al inicio del mismo.

2.4 Subtema 4: Modelos del proceso personal y del equipo

> El mejor proceso del software es el que está cerca de las personas que harán el trabajo. Si un modelo del proceso del software se ha desarrollado en un nivel corporativo u organizacional, será eficaz sólo si acepta una adaptación significativa para que cubra las necesidades del equipo de proyecto que en realidad hace el trabajo de ingeniería de software. (Pressman, 2010, pág. 48)

Proceso personal del software (PPS)

Watts Humphrey sugiere que, a fin de cambiar un proceso personal ineficaz, un individuo debe pasar por las cuatro fases, cada una de las cuales requiere capacitación e instrumentación cuidadosa.

"El proceso personal del software (PPS) pone el énfasis en la medición personal tanto del producto del trabajo que se genera como de su calidad. Además, el PPS responsabiliza al profesional acerca de la planeación del proyecto" (Pressman, 2010, pág. 48).

Vargas & Soto Durán (2010) afirman: "Este modelo está enfocado al desarrollo profesional del ingeniero, fomentando una adecuada administración de calidad de los proyectos de desarrollo, reducción de defectos del producto, estimación y planeación del trabajo".

Como sostiene Fernández (2009), El PSP enseña a los ingenieros lo siguiente:

- Cómo manejar la calidad de sus proyectos.
- Hacer las cosas simples para dar soluciones.
- A mejorar tiempos de estimación y planeación.
- Reducir los defectos de los productos.

> El 70% de los costos de desarrollo en un proyecto, lo constituyen las habilidades y los hábitos del trabajo de los ingenieros, los cuales determinan en gran parte el resultado del desarrollo de software. El proceso personal de software PSP (el cómo) puede ser utilizado por los ingenieros como una guía a un acercamiento disciplinado y estructurado al desarrollo de software. (Fernández, 2009, pág. 37)

Según Fernández (2009), el PSP puede aplicar a muchas partes del desarrollo de software, incluyendo:

- Desarrollo de programas.

- Definición de requisitos.
- Estructura de la documentación.
- Pruebas del sistema.
- Mantenimiento de los sistemas.
- Desarrollo de sistemas de software grandes.

Pressman declara que el modelo del PPS define cinco actividades estructurales:

Planeación. Esta actividad aísla los requerimientos y desarrolla las estimaciones tanto del tamaño como de los recursos. Además, realiza la estimación de los defectos (el número de defectos proyectados para el trabajo). Todas las mediciones se registran en hojas de trabajo o plantillas. Por último, se identifican las tareas de desarrollo y se crea un programa para el proyecto.
Diseño de alto nivel. Se desarrollan las especificaciones externas para cada componente que se va a construir y se crea el diseño de componentes. Si hay incertidumbre, se elaboran prototipos. Se registran todos los aspectos relevantes y se les da seguimiento.
Revisión del diseño de alto nivel. Se aplican métodos de verificación formal para descubrir errores en el diseño. Se mantienen las mediciones para todas las tareas y resultados del trabajo importantes.
Desarrollo. Se mejora y revisa el diseño del componente. El código se genera, revisa, compila y prueba. Las mediciones se mantienen para todas las tareas y resultados de trabajo de importancia.
Post mórtem. Se determina la eficacia del proceso por medio de medidas y mediciones obtenidas (ésta es una cantidad sustancial de datos que deben analizarse con métodos estadísticos). Las medidas y mediciones deben dar la guía para modificar el proceso a fin de mejorar su eficacia.

Proceso del equipo de software (PES)

De acuerdo con Watts Humphrey “el objetivo de éste es construir un equipo “autodirigido” para el proyecto, que se organice para producir software de alta calidad”.

Watts Humphrey desarrolla el proceso de software de equipo TSP para la unidad operacional más pequeña de las organizaciones de software, el equipo del proyecto. TSP fue diseñado para ser un proceso de nivel 5 CMM, para los equipos del proyecto. (Fernández, 2009, pág. 37)

> TSP es la fase posterior de PSP, está diseñado para el trabajo de equipos de desarrollo de software autodirigidos, que se orienta al desarrollo de productos con

el mínimo de defectos en tiempo y costos estimados. Cuenta con planes detallados y procesos como revisiones personales, inspecciones e índices de desempeño de calidad, y el fomento de la integración del equipo. (Mondragón Campos, 2011)

Humphrey define los objetivos siguientes para el PES:

- Formar equipos autodirigidos que planeen y den seguimiento a su trabajo, que establezcan metas y que sean dueños de sus procesos y planes. Éstos pueden ser equipos de software puros o de productos integrados (EPI) constituidos por 3 a 20 ingenieros.
- Mostrar a los gerentes cómo dirigir y motivar a sus equipos y cómo ayudarlos a mantener un rendimiento máximo.
- Acelerar la mejora del proceso del software, haciendo del modelo de madurez de la capacidad (CMM), nivel 5, el comportamiento normal y esperado.
- Brindar a las organizaciones muy maduras una guía para la mejora.
- Facilitar la enseñanza universitaria de aptitudes de equipo con grado industrial.

Según Fernández (2009) el proceso del software del equipo (TSP), junto con el proceso personal del software (PSP), ayuda al ingeniero de alto rendimiento a:

- Aseguramiento de la calidad de los productos de software.
- Crear productos de software seguros.
- Mejora el proceso de gerencia en una organización.

Adicionalmente, Fernández (2009) agrega que los grupos de la ingeniería que utilizan el TSP aplican conceptos integrados del desarrollo de sistemas orientados al software, llevando al equipo a:

- Establecer metas.
- Definir papeles del equipo.
- Determinación de riesgos.
- Producir un plan del equipo.

Igual que el PPS, el PES es un enfoque riguroso para la ingeniería de software y proporciona beneficios distintivos y cuantificables en productividad y calidad. El equipo debe tener un compromiso total con el proceso y recibir capacitación completa para asegurar que el enfoque se aplique en forma apropiada. (Pressman, 2010, pág. 50)

3. Preguntas de Comprensión de la Unidad

1. ¿Con qué objeto fueron propuestos los modelos prescriptivos?

Los modelos de proceso prescriptivo fueron propuestos originalmente para poner orden en el caos del desarrollo de software. La historia indica que estos modelos tradicionales han dado cierta estructura útil al trabajo de ingeniería de software y que constituyen un mapa razonablemente eficaz para los equipos de software.

2. ¿Qué se entiende por el modelo de cascada?

Es un enfoque metodológico que ordena rigurosamente las etapas del ciclo de vida del software, de forma que el inicio de cada etapa debe esperar a la finalización de la inmediatamente anterior.

3. ¿Cómo funciona el modelo de prototipos?

El paradigma de construcción de prototipos comienza con la recolección de requisitos. El desarrollador y el cliente encuentran y definen los objetivos globales para el software, identifican los requisitos conocidos y las áreas del esquema en donde es obligatoria más definición.

4. ¿Qué pretende el desarrollo basado en componentes (DSBC)?

El DSBC trata de sentar las bases para el diseño y desarrollo de aplicaciones distribuidas basadas en componentes software reutilizables. Dicha disciplina cuenta actualmente con un creciente interés, tanto desde el punto de vista académico como desde la industria, en donde la demanda de mecanismos y herramientas de desarrollo basados en componentes es cada día mayor.

5. ¿Con qué objeto se creó el proceso unificado?

El proceso unificado se originó con el objeto de combinar la mejor parte de cada uno de los modelos de software, es decir sus métodos individuales de análisis y diseño orientado a objetos. El resultado fue el lenguaje de modelado unificado (UML), el cual incluye una notación robusta para el modelado y desarrollo de los sistemas orientados a objetos.

4. Material Complementario

Los siguientes recursos complementarios son sugerencias para que se pueda ampliar la información sobre el tema trabajado, como parte de su proceso de aprendizaje autónomo:

Videos de apoyo:

Producción de software
https://www.youtube.com/watch?v=FAhdlq0Zytk&t=265s

Bibliografía de apoyo:

Gamma , E., Helm, R., Johnson, R., & Vlissides, J. (2003). Patrones de Diseño. Addison-Wesley.

Schach, S. (2006). Ingeniería de Software Clásica y Orientada a Objetos. *Sexta edición*. McGraw-Hill.

Links de apoyo:

Problemas con el desarrollo incremental
http://www.SoftwareEngineering-9.com/Web/IncrementalDev/

Ingeniería de software de cuarto limpio
https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Web/Cleanroom/

Instituto de Ingeniería de Software
http://www.sei.cmu.edu

Object Management Group (OMG)
http://www.omg.org

5. Bibliografía

- Bertoa, M., Troya, J., & Vallecillo, A. (2002). Aspectos de calidad en el desarrollo de software basado en componentes.
- Fernández y Fernández, C. (2011). Métodos formales aplicados a la industria del software. REPOSITORIO NACIONAL CONACYT.
- Fernández, H. (2009). Procesos de ingeniería de software. Obtenido de https://revistas.udistrital.edu.co/index.php/vinculos/article/view/4141/5806
- Gamma , E., Helm, R., Johnson, R., & Vlissides, J. (2003). Patrones de Diseño. Addison-Wesley.
- Gil, R. (2004). Estructura básica del proceso unificado de desarrollo de software. Sistemas y Telemática.
- Kuhn, D., Chandramouli, T., & Butler, R. (2002). Cost effective use of formal methods in verification and validation. Maryland, USA.
- Laboratorio Nacional de Calidad del Software. (2009). Curso de Introducción a la Ingeniería de Software. España: Instituto Nacional de Tecnologías de la comunicación (INTECO).
- Mondragón Campos, O. (2011). Integrando TSP y CMMI: Lo mejor de dos mundos. *Software Guru, 50*.
- Pressman, R. (2010). *Ingeniería del software: Un enfoque práctico.* Mc Graw Hill.
- Schach, S. (2006). Ingeniería de Software Clásica y Orientada a Objetos. *Sexta edición*. McGraw-Hill.
- Sommerville, I. (2011). *Ingeniería de Software.* México: Pearson Educación.
- Tabares, M., Alferez Salinas, G., & Alferéz Salinas, E. (2008). El desarrollo de software Orientado a Aspectos: Un Caso Práctico para un Sistema de Ayuda en Línea. Avances en Sistemas e Informática.
- Troya, J., Vallecillo, A., & Fuentes, L. (2017). Desarrollo de sodtware basado en componentes.