Ingeniería en Desarrollo de Software
3er semestre

Programa de la asignatura
Introducción a la ingeniería de software

Unidad 1. Ingeniería de software

Clave:

Ingeniería:	TSU:
15142318	**16142318**

Universidad Abierta y a Distancia de México

Índice

Unidad 1. Ingeniería de software

Presentación de la unidad

En esta unidad aprenderás la definición, características y elementos que conforman la ingeniería de software, analizarás los diferentes métodos de desarrollo para comprender cómo se relacionan con los elementos del ciclo de vida del software. Todo esto con la finalidad de que entiendas el proceso que involucra la creación de un software, el cual abarca desde la concepción de la idea hasta su implantación y mantenimiento.

Ejemplo de software

También comprenderás varios procesos de desarrollo de software, cada uno con sus ventajas y desventajas, diferentes enfoques y características. Cada uno apropiado para un tipo de proyecto en específico. Entender esto te servirá para aplicar o recomendar alguno en un momento dado en tu vida profesional.

Propósito

Al término de esta unidad lograrás:

- Identificar los conceptos fundamentales de la ingeniería del software.
- Comparar las características de los métodos de desarrollo de software.
- Identificar los tipos de técnicas de recolección
- Identificar los requerimientos de un caso de estudio
- Identificar diagramas del dominio y de interacción
- Analizar los lineamientos del diseño de la interfaz
- Analizar los lineamientos de la codificación
- Analizar los tipos de pruebas y el proceso de mantenimiento

Competencia específica

- Analizar los diferentes métodos de desarrollo de software para comprender cómo se relacionan con los elementos del ciclo de vida del software, identificando las características de cada método en un caso de estudio.

1.1. Introducción a la ingeniería de software

Entender el concepto de ingeniería de software es muy importante, ya que es el fundamento de todas las metodologías, modelos, teorías, estándares, etc. que se han generado a través del tiempo, con el fin de hacer el proceso de desarrollo de software más exacto y predecible, de tal manera que se generen acciones de mejora que lleven a la masificación del producto de manera industrial y económicamente redituable.

Antes de comenzar a explicar la ingeniería de software como tal, deberás conocer qué es la ingeniería y el software por separado, entendiendo las características y elementos que los constituyen, para que posteriormente comprendas por qué al unir ambas definiciones conforman una amplia área de estudio.

1.1.1. Ingeniería

Partir de la definición de ingeniería nos permite entender por qué surge la necesidad de generar procesos, para aplicar el ingenio, métodos, modelos, estándares y conocimiento científico, para aplicarlo en algo práctico y redituable, económicamente hablando; así como la sistematización y mejora de los procesos que permitan la producción más rápida y abundante de producto siendo éste un bien o servicio.

Ingeniería

Pero entonces, ¿qué es la ingeniería?

Ingeniería

(1) Conjunto de conocimientos y técnicas que permiten aplicar el saber científico a la utilización de la materia y de las fuentes de energía.

(2) Conjunto de conocimientos y técnicas cuya aplicación permite la utilización racional de los materiales y de los recursos naturales, mediante invenciones, construcciones u otras realizaciones provechosas para el hombre.

(3) Profesión y ejercicio del ingeniero.

Ingeniero

(1) Persona que profesa o ejerce la ingeniería (Pressman, 2010, p. xxi).

Por lo anterior, se puede afirmar que la ingeniería busca la aplicación de la ciencia en los procesos industriales para su perfeccionamiento y las personas encargadas de aplicar estos procesos son los especialistas llamados **ingenieros.**

1.1.2. Software

Referente al software podemos decir que es un elemento de la computadora que abarca la lógica de la misma, el cual es necesario para ejecutar una tarea. Tiene la propiedad de ser intangible y se compone de sistemas operativos y de aplicación. A continuación se presentan tres definiciones:

Software

(1) Instrucciones (programas de computadora) que cuando se ejecutan proporcionan la función y el rendimiento deseados.

(2) Estructuras de datos que permiten a los programas manipular adecuadamente la información.

(3) Documentos que describen la operación y el uso de programas (Pressman, 2010, p. 7).

Como notarás en las tres definiciones se refieren al software como un conjunto de programas. Entonces, podemos definir al software como un código escrito en un lenguaje específico para un procesador. El código es escrito en un lenguaje de programación. Existen tres tipos de lenguajes:

1. Bajo nivel o lenguaje máquina (ceros y unos)
2. Ensamblador formado por mnemónicos (palabras fáciles de recordar)
3. Alto nivel que se acerca más al lenguaje humano.

Los lenguajes ensamblador y el de alto nivel requieren ser traducidos a lenguaje máquina para ser ejecutados.

1.1.3. Ingeniería de software

Tomando en cuenta los conceptos anteriores tenemos elementos para deducir una definición de la ingeniería de software y para ello podemos considerar que es un área de estudio que se constituye por una serie de métodos y técnicas para desarrollar software. Sin embargo, es preciso que se analicen algunas definiciones como las que presenta el siguiente autor:

“La ingeniería del software es el establecimiento y uso de principios robustos de la ingeniería a fin de obtener económicamente software que sea fiable y que funcione eficientemente sobre máquinas reales” (Pressman, 2010, p. 17).

“Ingeniería del software: la aplicación de un enfoque sistemático, disciplinado y cuantificable hacia el desarrollo, operación y mantenimiento del software; es decir, la aplicación de ingeniería al software” (Pressman, 2010, p. 17).

Como puedes observar la primera definición habla sobre los principios robustos, lo cual hace referencia a procesos mejorados y confiables. Respecto al aspecto económico, sugiere que debido a la aplicación de la ingeniería de software se logrará establecer un proceso donde se puedan estimar mejor los costos y obtener beneficios económicos, que sean redituables para quien produce el software y más accesible para quien lo compra.

En la definición también se menciona el producto, que es el software, éste simplemente debe poder ser utilizado y la información que genere deberá ser fiable, eso encierra un gran sentido de calidad en el proceso de construcción de software, la cuestión no es sólo

construir en volumen, sino garantizar que el producto cubra los propósitos para los que fue creado, es decir, los requerimientos que definió el cliente para su desarrollo.

La segunda definición también tiene un enfoque claro hacia los procesos bien establecidos y cuantificables, es decir medibles, esto para revisar cómo se realizan los procesos y de esta manera poder mejorar cuando los resultados no son los esperados. Todo esto se contempla desde la creación del software hasta su instalación y mantenimiento.

Sin embargo, la ingeniería de software se confronta con su enemigo principal: "el desánimo" por aplicar las prácticas y modelos que ésta propone. Es decir, construir software con apego a un proceso y el tiempo que se necesita para administrarlo son factores que no todos los desarrolladores están dispuestos a invertir en sus proyectos. Otro factor es que el tiempo nunca parece ser suficiente, lo que lleva a los desarrolladores de software a dedicarse por completo a la codificación, haciendo a un lado procesos como el análisis y el diseño. Esto es equivalente a la construcción de un edificio, sin planos o maquetas, entonces, ¿el ingeniero se arriesga a construir algo que no ha planeado ni definido previamente? La respuesta es clara, no se arriesga, tiene que planear, analizar y diseñar antes de construir algo. En la ingeniería de software es muy similar, **antes de construir el software se debe planear, analizar y diseñar lo que se va a construir**, de lo contrario se toma un gran riesgo de no construir el producto esperado en tiempo y forma. La ingeniería de software contempla todo el proceso de desarrollo de sistemas de información, software de aplicación o de sistemas. En el siguiente tema podrás descubrir en qué consiste dicho proceso.

1.2. El proceso de desarrollo

El proceso de desarrollo de software consta de una serie de actividades necesarias para construir un producto de software. Existen diferentes modelos que proponen su propio estilo de proceso de desarrollo, pero todos comparten por lo menos las siguientes etapas:

1. Especificación: se definen los requerimientos y restricciones de operación.
2. Diseño e implementación: actividades necesarias para construir el software abarcando los requerimientos de la especificación.

3. Validación: revisión con el cliente para su aprobación.
4. Evolución: capacidad de adaptación a cambios y actualizaciones del software (Sommerville, 2011, p. 28).

No existe un modelo de construcción de software ideal, algunas empresas incluso han diseñado su propio proceso de desarrollo partiendo de la idea de alguno ya existente y haciendo las mejoras que se adapten a su realidad.

En los temas siguientes se analizarán algunos modelos de proceso de desarrollo de software de manera general resaltando las etapas, así como sus ventajas y desventajas.

1.2.1. Métodos de desarrollo: alternativas

Para la construcción de cualquier tipo de producto de software se desarrollan una serie de actividades partiendo desde la visión del proyecto que el cliente tiene, hasta el producto final.

Un modelo de desarrollo establece el orden en que se ejecutarán esas actividades en forma de procesos. Cada proyecto es único y no existe un modelo que pueda aplicarse cien por ciento a todos los proyectos. Por lo cual existen varias alternativas de modelos de desarrollo para ser utilizadas dependiendo de cada tipo de proyecto. Pero los modelos solamente se describen como marcos del proceso, esto significa que no se detallan las actividades, sino que dependen de la selección de mejores prácticas de desarrollo que cada organización utilice para cada etapa del proceso.

Además de las actividades, las descripciones de los procesos deben contener:

- Productos: son el resultado de ejecutar cada proceso, por ejemplo, un producto del proceso del análisis pueden ser las descripciones y diagramas del modelado de requerimientos.
- Roles: representan la responsabilidad que las personas tienen con el proyecto. Estos roles pueden ser líder o administrador del proyecto, analistas, diseñadores, programadores, encargados de pruebas, de calidad, etc.

- Condiciones: declaraciones válidas que restringen de alguna manera las actividades que tengan relación con el proceso o con el producto. Por ejemplo, una precondición puede ser que el cliente haya aprobado todos los requerimientos (Pressman, 2010, p. 28).

A continuación se explican diferentes modelos y sus características.

Modelo en cascada

Es el modelo más antiguo de desarrollo de software y ha servido como base para la generación de otros modelos de ciclo de vida; propone la construcción de software a través de una secuencia simple de fases. Cada fase contiene una serie de actividades para lograr un objetivo. Cada fase depende de la meta lograda en la anterior y por ello se representa con una flecha hacia abajo. Las flechas de regreso representan retroalimentación.

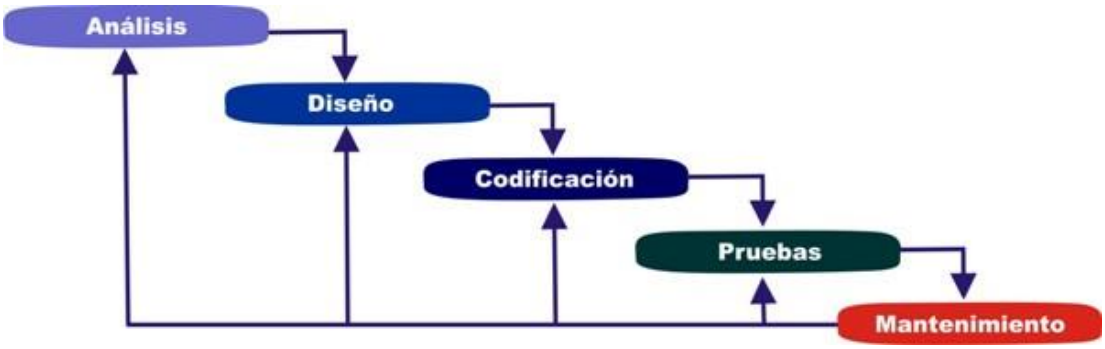


Diagrama del Ciclo de vida cascada.
Tomada de Pressman (2010, p. 30).

Etapas del modelo cascada:

- Análisis: analista y cliente definen todos los requerimientos del sistema y su especificación detallada.
- Diseño: a partir del análisis se diseñan las estructuras de datos, bases de datos, interfaces y procedimientos para dar solución a los requerimientos del cliente.
- Codificación: el diseño se traduce a código para la generación del software.
- Pruebas: se hace la revisión del código para comprobar que no tenga errores y realice lo esperado de acuerdo al diseño y al detalle del requerimiento.
- Mantenimiento: esta etapa se realiza después de la entrega del software y sirve para asegurar que el sistema siga funcionando y se da seguimiento a las mejoras que el cliente solicite.

Ventajas del modelo cascada:

- Es más sencillo planear las actividades del ciclo completo.
- La calidad del producto es alta.
- Permite trabajar con personal inexperto.
- Es el modelo más conocido.
- Es fácil de aprender.

Desventajas del modelo cascada:

- Los proyectos reales no siempre siguen el orden que propone este modelo, los cambios pueden causar confusión al equipo del proyecto por la poca interacción que propone el modelo.
- Es difícil que el cliente (quién solicitó la construcción del proyecto) exponga todos los requerimientos y en este modelo se requiere que desde el comienzo del proyecto sean definidos.
- El cliente deberá ser paciente, ya que verá alguna versión del trabajo hasta que el proyecto esté muy avanzado y cualquier error que detecte será cada vez más difícil y costoso de corregir.
- Este tipo de modelo genera estados de "bloqueo", cuando una etapa sufre algún retraso, algunos miembros del equipo deberán esperar a otros para completar su actividad.

Modelo de construcción de prototipos

Este modelo fue creado como una herramienta para identificar los requerimientos del software. Facilita al equipo de desarrollo entender los requerimientos del cliente y también ayuda al cliente a detallar más claramente las necesidades que tiene respecto a la construcción del software.

Existen varias formas de hacer un prototipo, esto depende de la naturaleza del proyecto, algunas de ellas son:

1. Un borrador de historia, tal y como fue escuchado por parte del cliente, se plantea a manera de historieta las funcionalidades del sistema, por ejemplo una interfaz con las opciones del menú, el área de despliegue de resultados, si es necesario la imagen corporativa. Apoyándose de la descripción de la historia podemos representar la

interactividad del sistema con los usuarios. Para construir este tipo de modelo puede ser utilizando un block de notas de papel o herramientas de software que faciliten la construcción de interacción hombre-máquina y permitan simular el proceso que se solicita. De esta manera el cliente puede hacerse a la idea de cómo va a funcionar el sistema final sin tener que construirlo, y así discutirlo con el analista.

2. Un modelo que implementa alguna función importante del sistema. Por ejemplo, se puede construir una aplicación rápida sin aspectos de diseño ni con la base de datos completa, lo mínimo requerido para simular un proceso interno del sistema o la validación de alguna fórmula. De esta manera se obtiene el entendimiento del requerimiento por parte del equipo de desarrollo.

Estos prototipos pueden servir como documentación de apoyo para especificar bien los requerimientos; sin embargo, no se recomienda que se utilicen como primera versión del sistema, pues seguramente se han construido de una manera rápida y con pocos detalles de calidad, y por lo tanto no representa el sistema real que el cliente necesita.

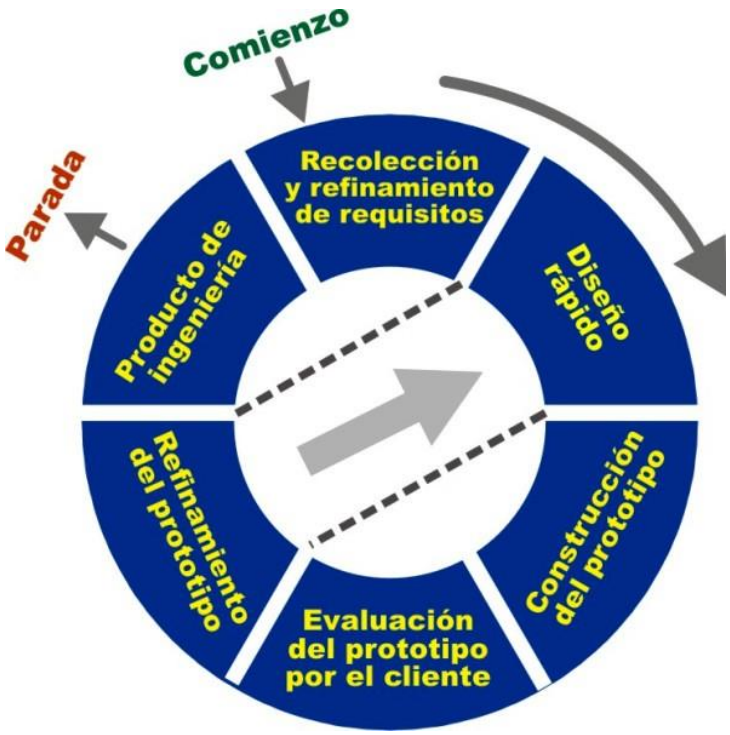


Figura 2. Diagrama del modelo de construcción de prototipos.
Tomada de Pressman (2010, p. 24).

Como podrás observar en la figura el modelo de construcción de prototipos tiene una serie de etapas que se describen a continuación.

La flecha que indica el comienzo señala la recolección de requisitos, así que presta atención a las siguientes descripciones:

1. Recolección de requisito: analista y cliente definen la especificación de requerimientos.
2. Diseño rápido: se hace el diseño del prototipo.
3. Construcción del prototipo: en cualquiera de las herramientas seleccionadas.
4. Evaluación del prototipo: cliente y usuario revisan el prototipo y generan observaciones.
5. Refinamiento del prototipo: las observaciones del cliente y usuarios sirven para mejorar el prototipo que nuevamente es construido y regresa al paso 2.
6. El ciclo concluye cuando el cliente y el usuario no tienen más observaciones, y además el prototipo es claro para el analista y el equipo de desarrollo.

Ventajas del modelo de construcción de prototipos:

- No modifica el flujo del ciclo de vida.
- Reduce el riesgo de construir productos que no satisfagan las necesidades de los usuarios.
- Reduce costos y aumenta la probabilidad de éxito.
- Exige disponer de las herramientas adecuadas.

Desventajas del modelo de construcción de prototipos:

- El cliente puede confundir las primeras versiones del prototipo con el producto final.
- El cliente puede desilusionarse al saber que los prototipos no son el producto y que este aún no se ha construido.
- Se requiere compromiso y trabajo del cliente para estar revisando los prototipos.
- No se sabe exactamente cuánto será el tiempo de desarrollo, ni cuántos prototipos se tienen que desarrollar.
- Si un prototipo fracasa, el costo del proyecto puede resultar muy caro.

- El desarrollador puede caer en la tentación de utilizar un prototipo, por ejemplo la interfaz gráfica de un módulo y continuarlo para construir el sistema sin tomar en cuenta aspectos de calidad necesarios para el proyecto.

Modelo incremental

El modelo incremental combina elementos del modelo cascada (aplicados repetidamente) con la filosofía interactiva de construcción de prototipos. Cada secuencia cascada produce un **incremento**. Cuando se utiliza este modelo, el primer incremento a menudo es un producto esencial (núcleo). Es decir, se afrontan requisitos básicos, para muchas funciones suplementarias (algunas conocidas, otras no) que quedan sin extraer. El cliente utiliza el producto central (o sufre la revisión detallada). Como resultado de utilización y/o de evaluación, se desarrolla un plan para el incremento siguiente. El plan afronta la modificación del producto central a fin de cumplir mejor las necesidades del cliente y la entrega de funciones, y características adicionales. Este proceso se repite siguiendo la entrega de cada incremento, hasta que se elabore el productivo completo (Pressman, 2010, p. 26).

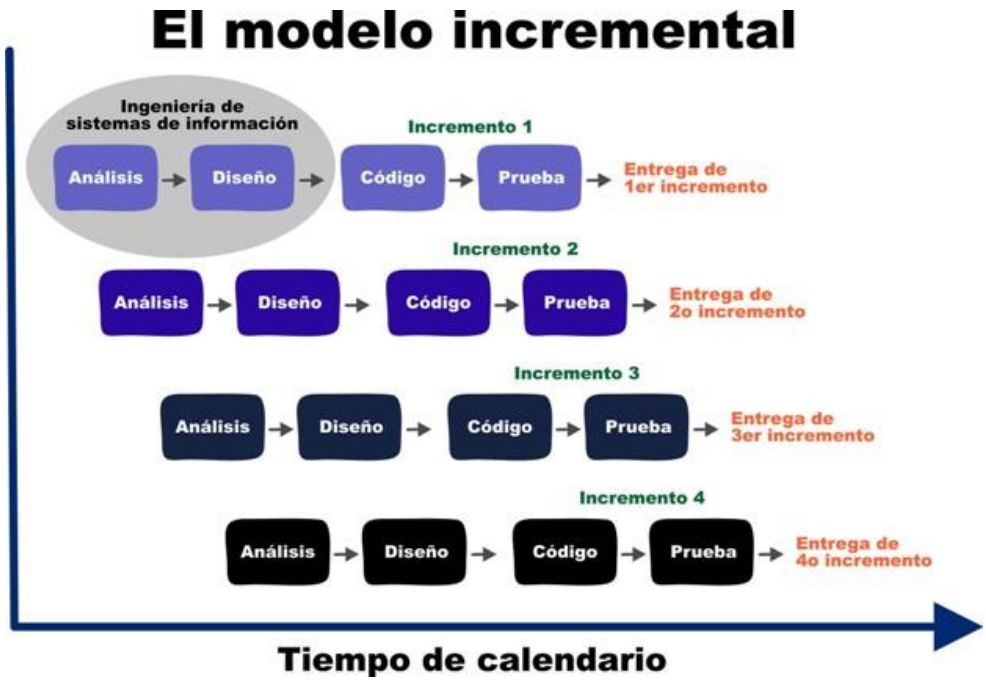


Diagrama del modelo incremental.
Tomada de Pressman (2010, p. 27).

Ventajas del modelo incremental:

- Construir un sistema pequeño es menos riesgoso que construir un sistema grande.
- Si un error es detectado, sólo la última iteración necesita ser descartada.
- Al desarrollar parte de las funcionalidades, es más fácil determinar si los requerimientos planeados para los niveles subsiguientes son correctos.

Desventajas del modelo incremental:

- Se puede suponer que los requerimientos han sido definidos desde el inicio del proyecto.
- Se necesita experiencia para definir los incrementos de forma de distribuir las tareas de manera proporcional.
- Se corre el riesgo de que el desarrollo se prolongue demasiado.

Modelo vida espiral

El modelo en espiral es un modelo de proceso de software evolutivo que acompaña la naturaleza interactiva de construcción de prototipos con los aspectos controlados y sistemáticos del modelo cascada. Se proporciona el potencial para el desarrollo rápido de versiones incrementales del software. Durante las primeras iteraciones, la versión incremental podría ser un modelo en papel o un prototipo. Durante las últimas iteraciones, se producen versiones cada vez más completas de ingeniería del sistema (Pressman, 2010, p. 28).

Se incorpora un nuevo elemento en el proceso de desarrollo del software, el “análisis de riesgos”, y define entre 3 y 6 regiones de tareas. Por ejemplo las que se muestran en la figura son:

Regiones del modelo espiral:

1. Comunicación con el cliente: cliente y analista establecen las políticas de comunicación.
2. Planificación: determinan tiempos, objetivos, restricciones, etc. del proyecto.
3. Análisis de riesgos: identificación y gestión del riesgo.
4. Ingeniería: desarrollo y verificación del producto del siguiente nivel.

5. Construcción y adaptación: actividades para desarrollar, realizar pruebas, instalación y mantenimiento al software.
6. Evaluación del cliente: validación del cliente hasta su aprobación y liberación.

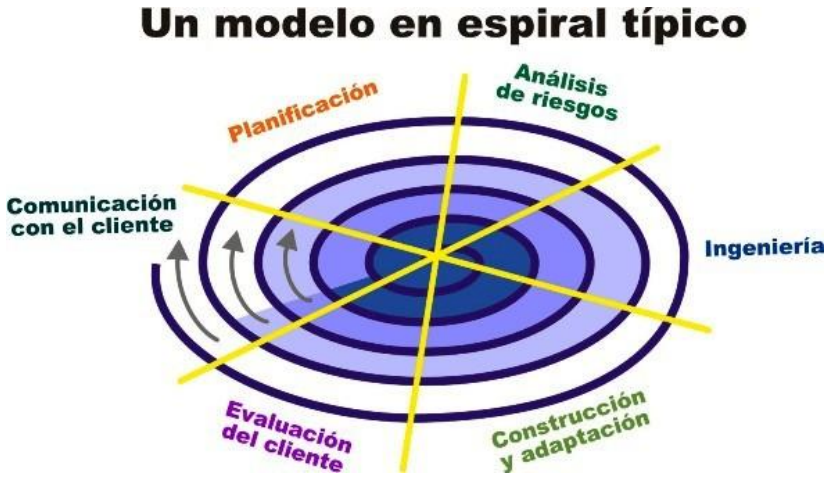


Figura 4. Diagrama del ciclo de vida espiral.
Tomada de Pressman (2010, p. 28).

Ventajas del ciclo de vida espiral:

- No se requiere tener todos los requerimientos definidos desde el inicio del proyecto.
- Es evolutivo, lo cual permite al cliente ir definiendo mejor sus requerimientos y también junto con el equipo ir gestionando los riesgos.
- Se pueden construir prototipos para disminuir el riesgo del proyecto, cuando no se cuenta con requerimientos claros.
- Representa de mejor manera un proceso de desarrollo real, con respecto al modelo cascada.
- Los requerimientos se van validando con cada iteración.

Desventajas del ciclo de vida espiral

- Genera la apariencia de ser interminable.
- Es muy complejo.
- Se requiere el trabajo y participación del cliente de manera continua.
- El modelo es relativamente nuevo por lo que no se cuenta con los casos suficientes para demostrar su efectividad.

1.2.2. El proceso unificado de desarrollo

El Proceso Unificado Racional (RUP, por las siglas de *Rational Unifed Process*) Es un ejemplo de un modelo de proceso que se derivó del trabajo sobre UML y el proceso asociado de desarrollo de software unificado. Conjunta elementos de todos los modelos de proceso genéricos. RUP se describe desde tres perspectivas (Sommerville, 2011):

1. Dinámica: muestra las fases del modelo a través del tiempo.
2. Estática: presenta actividades del proceso que se establecen.
3. Práctica: sugiere buenas prácticas a usar durante el proceso.

RUP es un modelo en fases que identifica 4 fases discretas en el proceso de software:

1. Inicio: define el alcance y objetivos del proyecto.
2. Elaboración: plan del proyecto, especificación de características y arquitectura base.
3. Construcción: construye y opera el producto.
4. Transición: transición del producto a la comunidad del usuario.

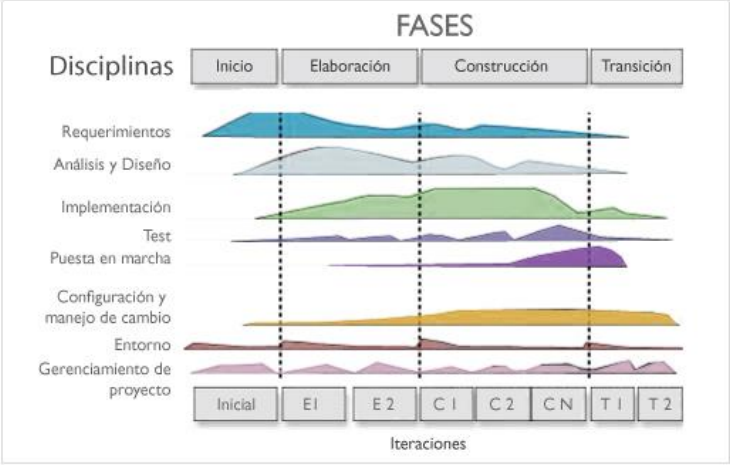


Figura 5. Diagrama del proceso unificado de desarrollo.
Tomada de Mornacco (2006, p.1).

Ventajas del Proceso Unificado Racional

- Identificación y gestión del riesgo en etapas tempranas del proyecto.
- El conocimiento adquirido en una iteración puede aplicarse de iteración a iteración.
- Involucramiento continúo de los usuarios.

Desventajas del Proceso Unificado Racional

- Por el grado de complejidad puede no resultar muy adecuado.
- El RUP es generalmente mal aplicado en el estilo cascada.
- Requiere conocimientos del proceso y de UML.
- El RUP no es un proceso adecuado para todos los tipos de desarrollo, por ejemplo, para el desarrollo de software embebido.

Independientemente del proceso de desarrollo que se seleccione, éste debe adaptarse para utilizarse por el equipo de desarrollo del proyecto de software. Si el proceso es débil, tendrá efecto sobre el producto final (el software). Además, todos los procesos deberán incluir actividades para gestionar el cambio, ya sea que implique la generación de prototipos para hacer más claros los cambios o bien que los procesos se estructuren para desarrollo y entrega de interactivos y de esta forma los cambios no afecten al sistema como un todo.

Existen otras metodologías que proponen procesos más rápidos de construcción de software para atender proyectos donde se requiere hacer un uso inmediato del mismo. En el siguiente tema se verán algunos procesos ágiles de desarrollo.

1.2.3. Métodos ágiles

Debido a que muchas empresas operan en ambientes económicos internacionales y cambiantes, deben responder rápidamente a las nuevas oportunidades, mercados, situaciones económicas, surgimiento de productos y servicios competitivos. El software forma una parte importante en casi todas las operaciones empresariales, por lo que los procesos de desarrollo deberán también construir software de una manera más rápida para que éste pueda seguir estando disponible como una herramienta confiable y actualizada con las nuevas demandas organizacionales.

El software es un elemento clave en casi todos los procesos y operaciones industriales, por ello se requiere que sea desarrollado de una manera rápida y efectiva para aprovechar las oportunidades y permanecer dentro del mercado (Sommerville, 2011).

Los procesos de desarrollo de software que se han visto en el tema anterior no están orientados al desarrollo rápido del software. A medida que los requerimientos cambian, o se descubren errores, el diseño tiene que reelaborarse y probarse de nuevo, provocando que el proceso se prolongue, y entregando el producto al cliente después de lo que se planeó originalmente. Además, estos procesos involucran un importante tiempo de gestión para no perder el control en el tiempo que lleva construir un software. Éstas son algunas de las razones por las cuales han surgido nuevos modelos ágiles de desarrollo de software.

Los procesos de desarrollo ágil se desarrollan para generar software útil de forma rápida. Esto se hace de manera incremental aumentando funcionalidades al sistema. En este tema veremos dos opciones de metodologías ágiles, que a continuación se explicarán:

Programación Extrema XP

La programación extrema (XP) es uno de los métodos ágiles más conocidos y utilizados. El término se generó por haber llevado los métodos tradicionales a niveles “extremos” como lo es el desarrollo iterativo.

Los involucrados en estos proyectos principalmente son el cliente y el equipo de desarrollo. El cliente es quien toma diferentes roles y puede ser una o varias personas, además es quien contrata al equipo de desarrollo, y será el usuario del software quién apruebe y libere el proyecto. El equipo de desarrollo generalmente está conformado por programadores que son los encargados de realizar el análisis, el plan y el desarrollo del sistema hasta liberarlo completamente.

En el XP muchas versiones de un sistema pueden desarrollarse por distintos programadores, integrarse y ponerse a prueba en un sólo día. Asimismo, los requerimientos se expresan como escenarios del usuario que posteriormente son traducidos a tareas. Los programadores trabajan en pares y antes de escribir el código desarrollan pruebas para cada tarea (Sommerville, 2011, p. 65).

El proceso de liberación de la programación extrema consiste en los siguientes pasos:

1. El usuario describe la “historia” del proceso que quiere automatizar, con sus propias palabras.

2. El equipo traduce la “historia” en tareas para construir las funciones del sistema.
3. El equipo y el usuario eligen cuál historia y actividades se pueden traducir más rápidamente en una función del sistema y planean liberarla en dos semanas aproximadamente.
4. El equipo se encarga de codificar, integrar y probar la funcionalidad que se ha planeado liberar.
5. Liberación del software, al principio con la primera entrega la funcionalidad es mínima, pero como las liberaciones son frecuentes se va agregando más funciones al sistema.
6. Se evalúa el sistema con el cliente de la iteración comprobando que la funcionalidad realmente opere de acuerdo a las especificaciones de la “historia” del usuario (Sommerville, 2011, p. 65).

Para poder llevar a cabo los pasos anteriores será necesario tomar en cuenta los siguientes puntos:

Realizar una **planeación incremental** basada en las tarjetas de historia, éstas se deben ordenar de acuerdo a la prioridad que se les haya asignado, se obtienen las tareas necesarias para su desarrollo y se les asigna un tiempo. Es incremental porque cada avance será un incremento en las funciones del sistema.

Cada **liberación será pequeña,** pero deben ser frecuentes de tal manera que se vayan integrando al sistema hasta terminarlo.

No se cuenta con mucho tiempo para **diseños detallados**, por lo cual éste debe ser rápido cubriendo únicamente los requerimientos planeados en cada entrega (iteración).
Las **primeras pruebas** son realizadas desde antes que se escriba el código y durante su desarrollo de tal manera que al terminar el código ya esté también probado.

De manera rápida el equipo de programación realiza una **refactorización** que consiste en remover el código duplicado, revisión de nombre de atributos y métodos, substitución de código, etc. con la finalidad de mejorar el código.

Los **programadores** trabajan **en pares** para apoyarse en el análisis, diseño, desarrollo y prueba del software.

Todos los programadores del **equipo** se responsabilizan por todo el código y todos lo conocen, de tal manera que cualquiera de ellos puede realizar cambios cuando sea necesario
El código que se genera en cada iteración se **integra** inmediatamente al sistema y se realizan inmediatamente las pruebas de integración.

No es recomendable que se **presione** al equipo desarrollando de forma exhaustiva en horas extras, ya que esto baja el estándar de calidad del código y la productividad se reduce.

Como parte del equipo de desarrollo debe integrarse a **un representante del cliente** (usuario) para proveer los requerimientos del sistema, o sea las historias que representarán las funcionalidades del sistema y también apoyara en la revisión de la iteración y su liberación (Sommerville, 2011, p. 66).

Metodología Scrum

Scrum tiene como objetivo gestionar y controlar los procesos de creación de software utilizando un modelo ágil, iterativo e incremental aplicando métodos como RUP y métodos ágiles como XP.

Scrum es un conjunto de reglas, procedimientos y prácticas relacionadas entre sí, que trabajan en conjunto para mejorar el entorno de desarrollo, reduce los gastos generales de la organización y se asegura que coincidan las iteraciones con los entregables a usuarios finales.

El proceso de desarrollo Scrum

Los proyectos se realizan en bloques cortos y fijos (repeticiones de 30 días o menos). Cada iteración debe generar un resultado completo. El proceso comienza con la lista de objetivos y requerimientos del producto que pueden funcionar como el plan del proyecto. En la lista

de objetivos y requerimientos el cliente agrupa los requerimientos en distintas iteraciones y entregas, tomando en cuenta el valor que aportan con respecto a su costo (los prioriza).

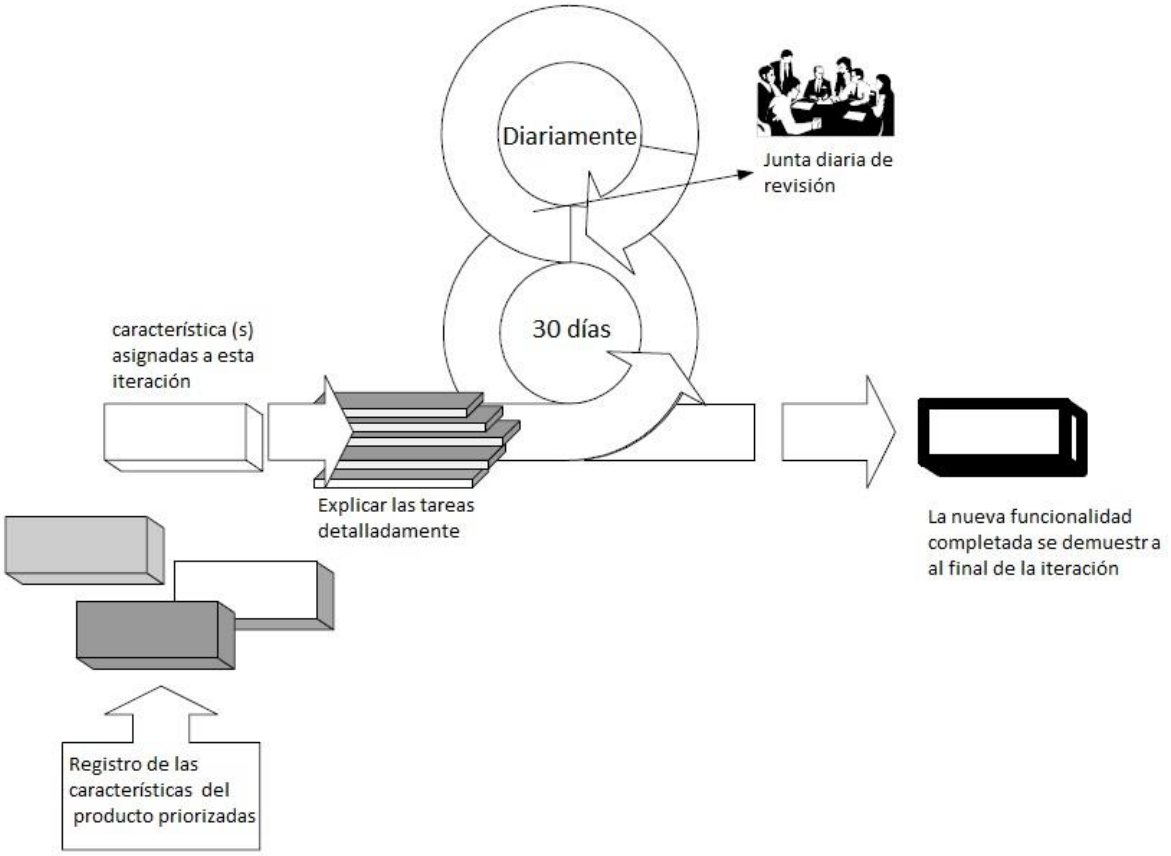
El primer día se realiza la junta de planificación de la iteración y se realiza lo siguiente:

1. Selección de requerimientos. El cliente presenta la lista de requerimientos del proyecto previamente priorizada. El equipo pregunta las dudas que hayan surgido y selecciona los requerimientos prioritarios que se compromete a completar en la iteración.
2. Planificación de la iteración. El equipo desarrolla una lista de actividades de la iteración para desarrollar los requerimientos a los que se ha comprometido, se asignan responsables para cada actividad y la estimación del tiempo se realiza tomando la opinión de cada integrante.
3. Ejecución de la iteración. Diariamente el equipo realiza una reunión de 15 minutos, para revisar el trabajo que han realizado y se hacen adaptaciones necesarias que permitan cumplir con el compromiso adquirido. Cada integrante del equipo debe contestar las siguientes preguntas:
    - ¿Qué han realizado los miembros del equipo desde la última reunión?
    - ¿Hay algún problema?
    - ¿Hay obstáculos para concluir las tareas?
    - ¿Qué va a hacer cada miembro del equipo antes de la próxima reunión?

En cada iteración debe existir un **facilitador,** que es quien se encarga de que el equipo pueda realizar su trabajo eliminando los obstáculos y evitando interrupciones externas que puedan afectar el desempeño de los miembros del equipo.
Inspección y adaptación. Cuando termina la iteración se realiza una junta de revisión que consta de:

1. Demostración: el equipo presenta al cliente los requerimientos que se desarrollaron en la iteración, lo cual representa el incremento de producto y debe estar preparado para integrarse sin el mínimo esfuerzo.
2. Retrospectiva: el equipo analiza sus lecciones aprendidas en este proyecto.

Este proceso se representa en el siguiente gráfico.



Scrum.
Tomada de Hunt (2006, p. 26).

El método Scrum tiene los siguientes beneficios:

- La gestión y control se desarrolla de manera ágil.
- Se reconoce explícitamente que los requerimientos pueden cambiar rápidamente en su enfoque iterativo e incremental de desarrollo del producto.
- Es posible utilizar todavía prácticas de ingeniería existentes dentro de Scrum para facilitar la introducción a los métodos ágiles en una organización.
- Es un enfoque basado en el equipo y ayuda a mejorar las comunicaciones y cooperación.
- Útil para proyectos pequeños y grandes.
- Ayuda a identificar y luego eliminar cualquier obstáculo para el buen desarrollo del producto final.

Muchas organizaciones han utilizado con éxito Scrum en miles de proyectos para gestionar y controlar el trabajo, al parecer con importantes mejoras en la productividad. Una observación interesante relacionada con Scrum es que puede ser visto como un proceso que ayuda a concluir que existen las prácticas de ingeniería dentro de un proceso iterativo controlado. De este modo, proporciona los valores, procedimientos y reglas que pueden ayudar a introducir un proceso de desarrollo más dinámico y flexible (Hunt, 2006, p. 25).

Cierre de la unidad

Durante esta primera unidad del curso, revisaste los conceptos de la Ingeniería de software y algunos de los diferentes métodos del proceso de desarrollo del software que existen, esto te ayudará a identificar proyectos por sus características y con base en éstas seleccionar la forma en que convendría desarrollarlos.

Es recomendable investigar en internet otras metodologías que no se cubren en esta unidad, para que puedas tener un criterio más amplio al identificar un repertorio mayor de opciones de desarrollo.
Es importante que resuelvas los casos de estudio, ya que de esta manera te forzarás a realizar un análisis de la información contenida en esta unidad.

Para saber más

Si deseas saber más acerca de metodologías ágiles consulta la siguiente dirección electrónica:

Por medio de este texto obtendrás una crítica más detallada de los métodos ágiles, ya que examina sus fortalezas y debilidades; está escrito por un ingeniero de software con vasta experiencia.

Fuentes de consulta

Fuentes básicas:

- Hunt, J. (2006). *Agile Software Construction*. United States: Springer.
- Kendall, K. E., y Kendall, J. E. (2011). *Análisis y diseño de sistemas*. México: Pearson Educación.
- Pressman, R. (2010). *Ingeniería de software.* España: McGraw-Hill/Interamericana.
- Sommerville, I. (2011). *Ingeniería de software*. México: Pearson Educación.

Fuentes complementarias:

- *Diccionario Larousse de la lengua española.* (2011). México: Larousse. Recuperado de http://www.diccionarios.com
- Real Academia Española de la Lengua. (2001). *Diccionario de la lengua española*. Recuperado de http://www.rae.es/rae.html