INSTITUTO SUPERIOR
TECNOLÓGICO
VICENTE LEÓN

Guía general de estudio de la asignatura

METODOLOGÍAS DE DESARROLLO DE SOFTWARE

Lorena Maricela Paucar Coque

Carrera de Tecnología Superior Desarrollo de Software
Asignatura: Metodologías de Desarrollo de Software
Código de la asignatura: DSW11-3P2
Nivel: Segundo

www.istvicenteleon.edu.ec

INSTITUTO SUPERIOR TECNOLÓGICO VICENTE LEÓN
Dirección Matriz, Latacunga, Cotopaxi
Matriz

ASIGNATURA
Lorena Maricela Paucar Coque

MSc. Ángel Velásquez Cajas Editor

Directorio editorial institucional
Mg. Omar Sánchez Andrade Rector
Mg. Fabricio Quimba Herrera Vicerrector
Mg. Milton Hidalgo Achig Coordinador de la Unidad de Investigación

Diseño y diagramación
Mg. Alex Zapata Álvarez
Mtr. Leonardo López Lidioma

Revisión técnica de pares académicos
– Adolfo Joel Moya Esparza
Instituto Superior Tecnológico Bolivar
a.moya@institutos.gob.ec
– Luis Gonzalo Borja Almeid
Universidad de las Fuerzas Armadas ESPEL
lgborja2@espe.edu.ec

ISBN: 978-9942-676-38-2
Primera edición
Agosto 2024

RIMANA EDITORIAL

Índice

DESARROLLO GUÍA DE ESTUDIO

1. Datos informativos

Lorena Maricela Paucar Coque, titular del número de Cédula de Identidad N. 1803040995, de nacionalidad ecuatoriana. Sus estudios de pregrado los cursó en el Colegio Experimental Ambato obteniendo el título de Bachiller en Informática. La preparación de pregrado los realizó en la Escuela Politécnica del Ejército, recibiendo el título de Ingeniera en Sistemas e Informática. En la universidad Pontificia Católica del Ecuador se graduó en la maestría Gerencia en Informática.Egresada del programa doctoral en Gerencia de la universidad de Yacambú, Venezuela. Su experiencia laboral se ha desempeñado en la empresa Textiles Rio Blanco como Auxiliar del Departamento de Mantenimiento (2000). Auxiliar en Sistemas Informáticos en la empresa López Torres Industrial S.A (2001).Docente de secundaria en el Colegio República de Argentina (2002).Desarrollador de software en Omnisoft (2001-2004), Solsoft (2003).En la empresa Computronic's se desempeñó como Administrador Red de Datos y Soporte Técnico. Experiencia como docente universitario por más de 13 años en la Universidad Técnica de Cotopaxi. Actualmente labora en el Instituto Superior Tecnológico Vicente León desempeñándose como docente, miembro de la Unidad de TICS. Ha contribuido en congresos y seminarios como ponente. Respecto a publicaciones: Autora y coautora de artículos científicos y libros.

2. Presentación de la Asignatura

El avance de la Informática a destacado muchos cambios dentro de la sociedad convirtiendo procesos manuales en automáticos mediante el Desarrollo de Software, a más de eso el avance de la tecnología del hardware donde las computadores día a día tienen procesadores, discos duros, memorias de última generación condición que permite alojar software de mayor capacidad, utilizando programas que facilitan el desarrollo de los mismos en menor tiempo, pero en esta transición nos vamos encontrando en la necesidad de implementar metodologías que permite organizar ordenadamente acciones que fundamentan la creación de una nueva automatización de un evento a más de eso la documentación de los mismos que en lo posterior nos servirá de referencia para una actualización. Desarrollar software implica muchas

cosas, desde su planificación hasta la puesta en marcha se deben de seguir un sinnúmero de pasos o actividades. Hoy en día existen diversas metodologías para hacerlo, sin embargo es necesario definir primero la naturaleza del software antes de elegir un determinado ciclo de vida.

3. Introducción de los Temas

Esta guía presenta los siguientes temas de estudio:

La unidad uno de la presente asignatura contiene información relevante sobre la introducción de las metodologías tradicionales de desarrollo de software, descripción de los tipos de metodologías tradicionales para desarrollo de software, historia de la metodología y características de la metodología RUP, mejoras de prácticas para el desarrollo del software y estructura del proceso.

La presente guía de aprendizaje encaminará al estudiante a gestionar un proyecto software a través de una metodología tradicional. Para ello, se entenderán los fundamentos de los principios de las metodologías de desarrollo de software, así como el conocimiento de las metodologías ágiles de desarrollo más comunes en la vida real.

4. Objetivos de Aprendizaje de la Asignatura

– Identificar las formas de estructurar esquemas de desarrollo tradicionales y las técnicas de gestión para el control de un proyecto.

– Aplicar una metodología de desarrollo de software durante el ciclo de vida de una aplicación desarrollada.

5. Unidad y Subunidades

1. Metodologías de Desarrollo de Software
1.1. Definición
1.2. Evolución Histórica

1.3. Tipos de Metodologías
1.3.1. Metodologías Tradicionales
1.3.1.1. Características
1.3.2. Metodologías Ágiles
1.3.2.1. Características
1.3.2.2. Origen de las Metodologías Ágiles
1.3.2.3. Manifiesto Ágil
1.4. Rational Unified Process (RUP)
1.4.1. Descripción de la Metodología
1.4.2. Historia
1.4.3. Características de la Metodología RUP
1.4.4. Ciclo de Vida
1.4.5. Elementos
1.4.6. Fases
– Inicio
– Elaboración
– Construcción
– Transición
1.4.7. Actores – Roles
1.5. Diferencias entre Metodologías Tradicionales y Ágiles

6. Resultados de Aprendizaje de la Unidad

Identifica las formas de estructurar esquemas de desarrollo ágiles y las técnicas de gestión para el control de un proyecto.

7. Estrategias Metodológicas

Se buscará que el aprendizaje se base en el análisis y solución de problemas, usando información en forma significativa, favoreciendo la retención, comprensión, el uso o aplicación de la información, los conceptos, ideas, principios y las habilidades en la resolución de problemas.

Se realizará la resolución de casos para favorecer la realización de procesos de pensamiento complejo, tales como: análisis, razonamientos,

argumentaciones, revisiones y profundización de diversos temas. Se realizarán prácticas de laboratorio para desarrollar las habilidades proyectadas en función de las competencias y el uso de software que permitirá llevar la teoría a la práctica.

Se desarrollará ejercicios orientados a la carrera y otros propios del campo de estudio. La evaluación cumplirá con las tres fases: diagnóstica, formativa y sumativa, valorando el desarrollo del estudiante en cada tarea y en especial en las evidencias del aprendizaje de cada unidad.

Se organizará actividades y proyectos que requieran que los estudiantes trabajen juntos para alcanzar metas comunes. Esto promueve habilidades de trabajo en equipo, comunicación, liderazgo y resolución de conflictos.

8. Criterios de Evaluación

Fases	Instrumentos	Primer Parcial %(Puntos)	Segundo Parcial %(Puntos)	Promedio %(Puntos)
Fase 1: Trabajos Prácticos	Trabajos Individual	2 2 2	2 2 2	2 2 2
Fase 2: Lecciones	Trabajo de clase o colaborativo	2	2	2
Fase 3:				
Evaluación	Cuestionario	2	2	2
	Total	10	10	10

Horas Docencia= 32
Horas de Trabajo Autónomo= 10
Horas de prácticas experimentales= 32

9. Desarrollo de las Subunidades

Metodologías de Desarrollo de Software

1.1. Definición

La metodología para el desarrollo de software implica abordar de manera sistemática la ejecución, gestión y administración de un proyecto con el propósito de llevarlo a cabo de manera altamente exitosa. Al respecto, Morales (2015) lo define como "el conjunto de técnicas y procedimientos que se implementan a lo largo del ciclo de vida del software con el fin de cumplir con objetivos" (p. 17). En sí, esta metodología comprende los procesos a seguir sistemáticamente para idear, implementar y mantener un producto software desde que surge la necesidad del producto hasta que cumplimos el objetivo por el cual fue creado.

1.2. Evolución Histórica

En la década de 1950, surgieron los primeros lenguajes de programación de alto nivel. Debido a las restricciones en la capacidad y velocidad de las computadoras de la época, los programadores se enfocaban en minimizar la cantidad de líneas de códigos y optimizar el uso de recursos. Espinoza (2017), manifiesta alrededor de 1968, con la introducción de las mini-computadoras, se observó un incremento en la capacidad de memoria y velocidad de cálculo. La diversidad de lenguajes de alto nivel, en conjunto con el sistema IBM/360, generaba dificultades para transferir programas entre diferentes sistemas.

Los costos asociados al software superaban a los del hardware, lo que llevó a reconsiderar la durabilidad de los programas. Se establecieron criterios para el éxito en el desarrollo de software, tales como un bajo costo inicial, facilidad de mantenimiento, portabilidad, satisfacción de requisitos del cliente y cumplimiento estándares de calidad. En la década de 1970, la Programación Estructurada surgió como respuesta a estos desafíos mediante la aplicación de la estrategia Top Down, formalizando el uso de constructores y proponiendo técnicas de modelado como diagramas de flujo y pseudocódigo.

Este enfoque sentó las bases para el surgimiento de la Ingeniería de Software como disciplina. Metodologías como SA/SD, que distinguían entre el modelado de procesos y datos en el análisis y empleaban cartas de estructura en el diseño, comenzaron a surgir. Se promovieron modelos de ciclo de vida con el objetivo de reducir costos y aumentar la confiabilidad, destacando el modelo cascada, aunque su rigidez condujo a la búsqueda de alternativas como los modelos prototipo, Rapid Application Development y espiral. En los años 80, herramientas como CASE e IDE brindaron apoyo, pero no estaban alineadas completamente con el concepto original de metodología buscado desde sus inicios.

La programación orientada a objetos, en contraste con la estructurada, simula un entorno real donde los elementos interactúan. Se consolidó principalmente entre las décadas de los 70 y 80, prometiendo mayor flexibilidad y reutilización de objetos. Inicialmente diseñada para la programación, luego para el diseño y finalmente para el análisis, muchas metodologías basadas en esta filosofía no cubren todo el ciclo de vida del desarrollo de software. En los años 90, se logra la unificación y estandarización de una notación con la creación del Lenguaje de Modelamiento Unificado (UML), impulsado por Rational. Esto permitió que, independientemente de la metodología utilizada, todos pudieran comprender y comunicar sus diseños e ideas.

En 1998, Rational Unified Process (RUP), también de Rational, fue lanzada al mercado. RUP integra elementos de diversas metodologías en un enfoque universal adaptable a las particularidades específicas de cada proyecto. Hacia finales de la década, surgieron las metodologías ágiles como alternativa a las predictivas, vistas como burocráticas y lentas. Las metodologías ágiles se caracterizan por la ausencia de documentación y la colaboración de equipos multidisciplinarios. Ejemplos de estas metodologías son XP (Programación Extrema), Crystal Clear, Método de Desarrollo de Sistemas Dinámicos (DSDM) y el Proceso Unificado Ágil.

1.3. Tipos de Metodologías

La elección de una metodología de desarrollo varía según los objetivos, el tiempo y los recursos disponibles para el equipo de desarrollo. Considerando estas características, desde la década de 1970 existen antecedentes de los primeros modelos estandarizados para todo el proceso de desarrollo.

Examinando las características fundamentales de las metodologías más prominentes en la industria, podemos clasificarlas en dos categorías distintas que son las más comunes: Metodologías Tradicionales y Metodologías Ágiles.

1.3.1. Metodologías Tradicionales

Las metodologías tradicionales de desarrollo de software se basan en la planificación de proyectos. Inicia el desarrollo de un proyecto mediante un riguroso proceso de definición de requisitos, antes de abordar las fases de análisis y diseño, con el objetivo de lograr resultados de alta calidad dentro de un cronograma establecido. Pérez (2011) plantea que estas metodologías tradicionales establecen una planificación rigurosa y detallada del proyecto, respaldada por herramientas y una carga de trabajo considerable en las fases de planificación, diseño y documentación.

Esto se hace con el propósito de lograr un desarrollo predecible dentro de límites temporales y costos específicos; recordando que estas metodologías surgieron para abordar los desafíos surgidos en la crisis del software experimentada décadas atrás. Según Navarro et al.(2013)

Estas metodologías, se concibe un único proyecto de gran envergadura con una estructura bien definida. Se sigue un proceso secuencial unidireccional sin posibilidad de retroceso; el proceso es inflexible y no experimenta cambios. Los requisitos se acuerdan de manera integral para todo el proyecto, lo que implica extensos períodos de planificación inicial y una comunicación limitada con el cliente una vez que esta fase ha concluido.(p. 31)

Esto implica que estas metodologías tradicionales a menudo enfrentan desafíos cuando se trata de adaptarse a cambios en los requisitos o de manejar proyectos grandes y complejos.

1.3.1.1. Características. Las metodologías tradicionales en el ámbito de gestión de proyectos de desarrollo de software suelen compartir algunas características, al respecto Espinoza (2017) menciona las siguientes:

– División de etapas ordenadas y secuenciales, cada una con objetivos y reglas prefijadas.

– Administración de documentación específica y detallada de la planificación del proyecto de software, elaborada antes de iniciar las actividades.

– Al final de cada etapa o sub- etapa definida del proyecto (hito) se materializa un entregable y se realiza retroalimentación.

– Predomina el control del proceso. Específica roles, funciones, tareas y herramientas que pueden ser completadas por personal idóneo. No es importante quién, ni cómo lo hace, sino qué rol cumple.

– Las comunicaciones entre el equipo y con los interesados son planeadas de manera formal y mayormente sirven para anunciar avances. Prefiere la interacción con el cliente, para recolección de requisitos y sugerencias, al inicio del proyecto.

– Se asume que las estimaciones de alcance, tiempo y costos, no son variables, y que se deben respetar y cumplir.

1.3.2. Metodologías Ágiles

Los avances en las Tecnologías de la Información y en los contextos empresariales plantearon un desafío para que las metodologías tradicionales se adapten a los cambios. Sin embargo, su capacidad para responder de manera ágil a las variaciones a lo largo de todo el ciclo de desarrollo resultó limitada, lo que contribuyó en muchos casos al fracaso del producto final. "Estas metodologías sugieren iniciar proyectos sin estimaciones precisas basadas en supuestos, reconociendo que la estimación inicial es aproximada. Este enfoque permite ganar experiencia rápidamente y mejorar la certeza de las estimaciones prescindiendo de supuestos" (Alaimo, 2013, p. 29).

Las metodologías ágiles se centran en ciclos cortos y equipos pequeños de desarrollo, destacando el trabajo colaborativo y la participación activa del cliente. Utilizan enfoques mejorados para el desarrollo de software, priorizando la calidad sin requerir documentación exhaustiva. La integración directa del cliente en el proceso y la colaboración efectiva entre los miembros del equipo contribuyen a la satisfacción del cliente.

1.3.2.1. Características. Este tipo de metodologías ágiles se guían en ciclos cortos de desarrollo y pequeños grupos y como una de las principales características es el trabajo en equipo y se involucra de una forma activa al cliente. Estas metodologías emplean enfoques más efectivos en el desarrollo de software, obteniendo un producto de calidad sin una documentación exhaustiva, clientes satisfechos ya que están integrados directamente con el desarrollo del producto, la colaboración entre los miembros del equipo de desarrollo.

Las metodologías presentan varias características como las siguientes:

– Capacidad de respuesta a cambios de requisitos a lo largo del desarrollo.

– Entrega continua y en plazos breves de software funcional.

– Trabajo en conjunto entre el cliente y el equipo de desarrollo.

– Importancia de la simplicidad, eliminando el trabajo innecesario.

– Atención continua a la excelencia técnica y al buen diseño.

– Mejora continua de los procesos y el equipo de desarrollo (Amaro & Valverde, 2007).

1.3.2.2. Origen de las Metodologías Ágiles. En la década de 1990, surgieron las Metodologías Livianas, como Extreme Programming(XP), Scrum y Lean Software Development. En febrero de 2001, diecisiete profesionales se reunieron en Utah (EEUU) para establecer los valores y principios de la filosofía ágil. Crearon la Agile Alliance, una organización sin fines de lucro, y redactaron el Manifiesto Ágil, buscando ofrecer una alternativa a los procesos tradicionales de desarrollo de software, caracterizados por la rigidez y la documentación excesiva. La Agile Alliance se dedica a promover estos valores y principios, facilitando su adopción en las organizaciones (Alaimo, 2013).También se declaró la piedra angular del movimiento ágil, conocida como Manifiesto Ágil (Agile Manifiesto).

1.3.2.3. Manifiesto Ágil. El Manifiesto Ágil constituye un conjunto de valores y principios que orienta el desarrollo de software ágil. Fue formulado

por un grupo de profesionales en desarrollo de software conocidos como los "Manifestantes Ágiles" durante una reunión en 2001.

El manifiesto Ágil se compone de 4 valores y 12 principios.

Valores

1. Valorar a las personas y las interacciones entre ellas por sobre los procesos y las herramientas.

2. Valorar el software funcionando por sobre la documentación detallada.

3. Valorar la colaboración con el cliente por sobre la negociación de contratos.

4. Valorar la respuesta a los cambios por sobre el seguimiento estricto de los planes.

Principios

1. Nuestra mayor prioridad es satisfacer al cliente a través de entregas tempranas y frecuentes de software con valor.

2. Aceptar el cambio incluso en etapas tardías del desarrollo. Los procesos ágiles aprovechan los cambios para darle al cliente ventajas competitivas.

3. Entregar software funcionando en forma frecuente, desde un par de semanas a un par de meses, prefiriendo el periodo de tiempo más corto.

4. Expertos del negocio y desarrolladores deben trabajar juntos diariamente durante la ejecución del proyecto.

5. Construir proyectos en torno a personas motivadas, generándoles el ambiente necesario, atendiendo sus necesidades y confiando en que ellos van a poder hacer el trabajo.

6. La manera más eficiente y efectiva de compartir la información dentro de un equipo de desarrollo es la conversación cara a cara.

7. El software funcionando es la principal métrica de progreso.

8. Los procesos ágiles promueven el desarrollo sostenible. Los sponsors, desarrolladores y usuarios deben poder mantener un ritmo constante indefinidamente.

9. La atención continua a la excelencia técnica y buenos diseños incrementan la agilidad.

10. La simplicidad el arte de maximizar la cantidad de trabajo no hecho- es esencial.

11. Las mejores arquitecturas, requerimientos y diseños emergen de equipos auto-organizados.

12. A intervalos regulares, el equipo reflexiona acerca de cómo convertirse en más efectivos, luego mejora y ajusta su comportamiento adecuadamente (Alaimo, 2013).

1.4. Rational Unified Process (RUP)

1.4.1. Descripción de la Metodología

El Proceso Unificado Racional (RUP), es un proceso de Ingenieria de Software desarrollado y mantenido por Rational Software Corporation. Se constituye en una metodología estándar más utilizada para el anáisis, implementación y documentación en el desarrollo de software. Según Garcia (2010), es un "método de desarrollo de software pesado.

Se requiere un grupo grande de programadores para trabajar. Es un marco del proyecto que describe una clase los procesos que son iterativos e incrementales. Proporciona una estructura completa para guiar el desarrollo de software"(p. 4).

En esta metodología los pasos no están firmemente establecidos, eso quiere decir que se trata de un conjunto de metodologías adaptables al contexto y requerimientos de cada organización, donde el software es organizado como una colección de unidades denominados objetos, que están conformados por datos y funciones que interactúan entre sí.

1.4.2. Historia

La metodología RUP fue creada por Ivar Jacobson, Grady Booch y James Rumbaugh en los años 90.

Estos tres especialistas en ingeniería de software colaboraron para desarrollar un enfoque unificado que integrará las mejores prácticas del desarrollo de software y proporcionará una metodología completa y estructurada.

RUP se fundamenta en un modelo incremental y evolutivo, fomentando la flexibilidad y adaptabilidad en el desarrollo de software. Inicialmente, fue parte del conjunto de herramientas Rational de IBM, pero posteriormente se independizó como una metodología propia.

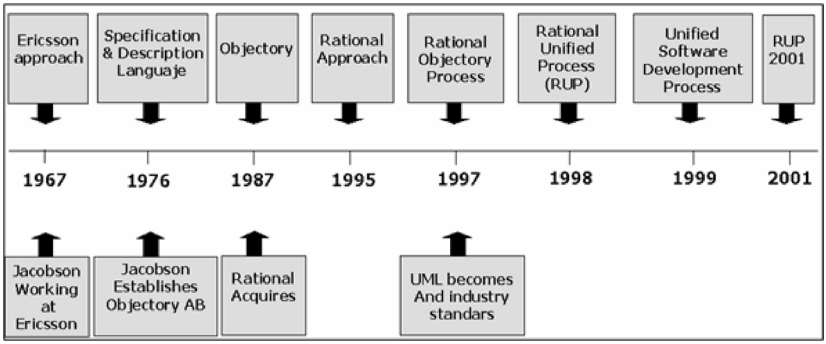
El antecedente más importante se ubica en 1967 con la Metodología Ericsson (Ericsson Approach) elaborada por Ivar Jacobson, una aproximación de desarrollo basada en componentes, que introdujo el concepto de Caso de Uso.

Entre los años de 1987 a 1995 Jacobson fundó la compañía Objectory AB y lanza el proceso de desarrollo Objectory (abreviación de Object Factory). Posteriormente en 1995 Rational Software Corporation adquiere Objectory AB y entre 1995 y 1997 se desarrolla Rational Objectory Process (ROP) a partir de Objectory 3.8 y del Enfoque Rational (Rational Approach) adoptando UML como lenguaje de modelado.

Desde ese entonces y a la cabeza de Grady Booch, Ivar Jacobson y James Rumbaugh, Rational Software desarrolló e incorporó diversos elementos para expandir ROP, destacándose especialmente el flujo de trabajo conocido como modelado del negocio. En junio del 1998 se lanza Rational Unified Process (Garcia, 2010).

Figura 1

Rational Unified Process (RUP)



Nota. Evolución de la metodología RUP. Tomado de investigación metodológica RUP (2010) https://googlegroups.com/group/analisisdesistemasaumg/attach/aa017922c4817fac/investigaci%C3%B3n%20metodolog%C3%ADa%20RUP.pdf?part=0.1

1.4.3. Características de la Metodología RUP

RUP se adapta a una amplia gama de proyectos y organizaciones al incorporar una variedad de las mejores prácticas del desarrollo de software actual. En otras palabras, explica cómo los grupos de creadores de software pueden desplegar al mercado los alcances del desarrollo de software de manera efectiva. Todas estas cosas se denominan "buenas prácticas", no necesariamente porque se puedan calificar, sino porque las organizaciones exitosas las observan comúnmente en la industria.

Según Cárcenas (2017), Rational Unified Process proporciona a cada miembro del equipo las directrices, plantillas y herramientas necesarias para sacar el máximo provecho de las siguientes buenas prácticas:

1. Desarrollo de software iterativamente

Desarrollo del producto mediante iteraciones con hitos bien definidos, en las cuales se repiten las actividades pero con distinto énfasis, según la fase del proyecto.

2. Gestión de requerimientos.

RUP brinda una guía para encontrar, organizar, documentar y seguir los cambios de los requisitos funcionales y restricciones. Utiliza una notación de Caso de Uso y escenarios para representar los requisitos.

3. Uso de arquitecturas basada en componentes.

La creación de sistemas intensivos en software requiere dividir el sistema en componentes con interfaces bien definidas, que posteriormente serán ensamblados para generar el sistema. Esta característica en un proceso de desarrollo permite que el sistema se vaya creando a medida que se obtienen o se desarrollan sus componentes.

4. Modelos de software visual.(usando UML).

UML es un lenguaje para visualizar, especificar, construir y documentar el software. Utilizar herramientas de modelado visual facilita la gestión de dichos modelos, permitiendo ocultar o exponer detalles cuando sea necesario. El modelado visual también ayuda a mantener la consistencia. En resumen, el modelado visual ayuda a mejorar la capacidad del equipo para gestionar la complejidad del software.

5. Verificación de la calidad del software.

Es importante que la calidad se evalúe en varios puntos durante el proceso de desarrollo, especialmente al final de cada iteración. En esta verificación las pruebas juegan un papel fundamental y se integran a lo largo de todo el proceso.

6. Control de cambios del software (p.18).

El cambio es un factor de riesgo crítico en los proyectos de software. El software cambia no sólo debido a acciones de mantenimiento posteriores a la entrega del producto, sino que durante el proceso de desarrollo, especialmente por su posible impacto son los cambios en los requisitos.

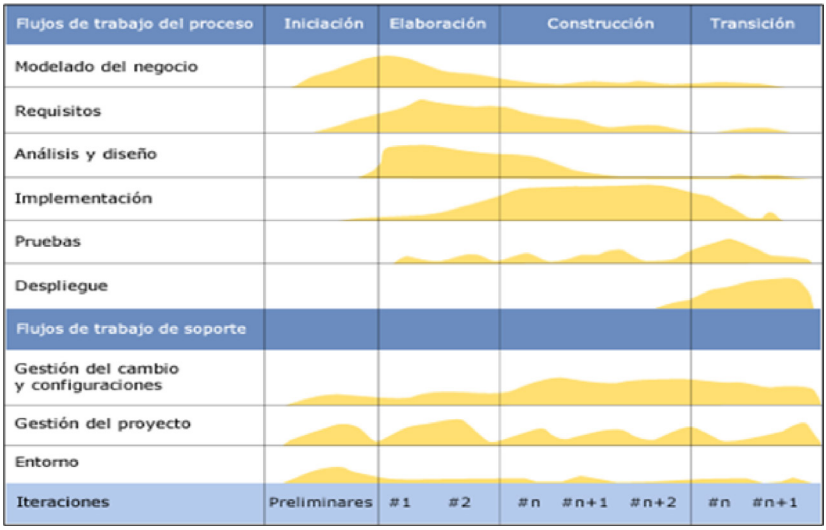
Por otra parte, otro gran desafío que debe abordarse es la construcción de software con la participación de múltiples desarrolladores, trabajando a la vez en una release, y quizás en distintas plataformas. La ausencia de disciplina rápidamente conduciría al caos. La Gestión de Cambios y de Configuración es la disciplina de RUP encargada de este aspecto.

1.4.4. Ciclo de Vida

El ciclo de vida de RUP es la implementación de un desarrollo en espiral. Se crea ensamblando elementos en una secuencia semiordenada. Un ciclo de vida organiza las tareas en fases e iteraciones. RUP divide el proceso en cuatro fases, con iteraciones realizadas en diferentes momentos según el proyecto y con mayor o menor énfasis en diferentes actividades (Serna, 2019).

Figura 2

Fases (iteraciones) metodología RUP



Nota. Fases de la metodología RUP. Tomado de investigación y fundamento de la metodología RUP (2019) https://www.academia.edu/19661140/1PROY1_METODOLOGIA_RUP

La primera iteración (en las fases de iniciación y elaboración) se centra en comprender el problema y la tecnología, definir el alcance del proyecto, abordar los riesgos clave y establecer una primera aproximación o línea de base de la arquitectura.

En la fase inicial, las iteraciones ponen más énfasis en las actividades de modelado del negocio y requerimientos.

En la fase de elaboración, las iteraciones se basan en el desarrollo de puntos de referencia arquitectónicos; también cubren las partes de flujo de trabajo de requisitos, modelo de negocio (refinamiento), análisis, diseño e implementación de los puntos de referencia arquitectónicos.

Durante la fase de construcción, el producto se construye a través de una serie de iteraciones (implementación, pruebas y muestreo del sistema).

Cada iteración selecciona algunos casos de uso, refina su análisis, diseño, los implementa y prueba.

Cada ciclo completa una pequeña cascada. Realice varias iteraciones hasta completar la implementación de la nueva versión del producto deseado.

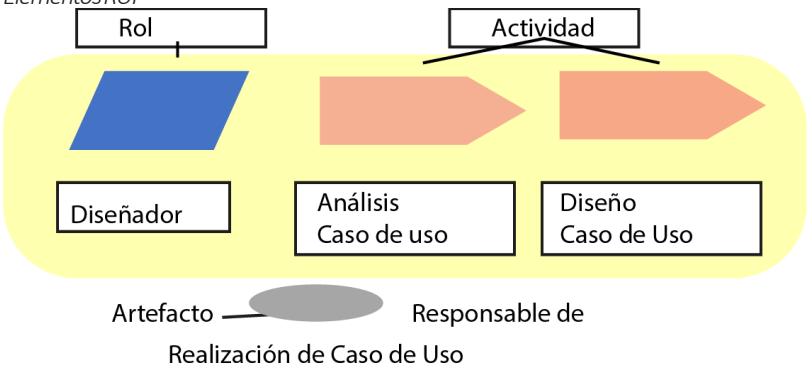
La fase de transición garantiza que el producto esté listo para ser enviado a la comunidad de usuarios para su prueba (Serna, 2019) .

1.4.5. Elementos

Un proceso de desarrollo de software define quién hace qué, cómo y cuándo. RUP define cuatro elementos los roles, que responden a la pregunta ¿Quién?, las actividades que responden a la pregunta ¿Cómo?, los productos, que responden a la pregunta ¿Qué? y los flujos de trabajo de las disciplinas que responde a la pregunta ¿Cuándo?. (Rueda, 2006, pág. 17)

Figura 3

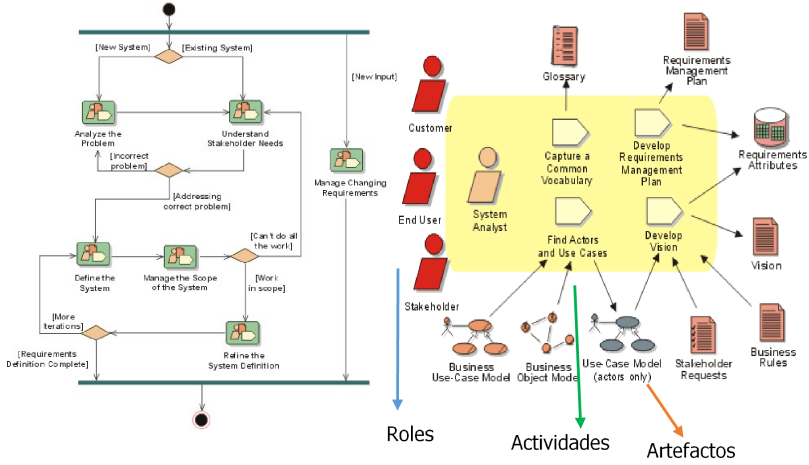
Elementos RUP



Nota. Componentes de metodología RUP. Tomado de Aplicación de metodología RUP (2006) http://biblioteca.usac.edu.gt/tesis/08/08_0308_CS.pdf

Figura 4

Relación entre roles, actividades, artefactos



Nota. Detalle de un workflow mediante roles, actividades y artefactos. Tomado de Aplicación de metodología RUP (2006) http://biblioteca.usac.edu.gt/tesis/08/08_0308_CS.pdf

El método RUP (Proceso Unificado Racional) es un enfoque para el desarrollo de software que utiliza una estructura iterativa e incremental.

Garcia (2010) menciona los elementos clave del método RUP son:

1. Roles: Define las responsabilidades de los participantes en el proceso, como analistas, diseñadores, desarrolladores y probadores.

2. Flujos de Trabajo (Workflows): Representan las secuencias de actividades que deben llevarse a cabo para completar una tarea específica. Algunos ejemplos de flujos de trabajo incluyen el análisis de requisitos, el diseño, la implementación y la prueba.

3. Artefactos: Son los productos de trabajo que se crean y modifican a lo largo del proceso. Incluyen documentos, modelos y código fuente.

4. Fases de desarrollo: La metodología RUP organiza el proceso de desarrollo en fases para facilitar la gestión y el control del proyecto.

1.4.6. Fases

Inicio

En esta etapa se prepara el caso de negocio del sistema y se establece el alcance del proyecto. Para ello, es necesario identificar todas las entidades con las que interactuará el sistema, conocidos como actores, además se define la naturaleza de esta interacción en un alto nivel. Según Jaramillo (2016), manifiesta que al final de la fase de incepción, los entregables son generalmente:

– Documento de visión: Visión general de requerimientos, características clave y restricciones principales.

– Modelo de casos de uso inicial

– Glosario de proyecto inicial

– Caso de negocio Inicial: Incluye contexto del negocio, criterio de éxito (proyección de ingresos, reconocimiento de mercado, entre otros) y pronóstico financiero

– Evaluación de riesgos inicial

– Plan del proyecto que muestre fases e iteraciones

–Modelo de negocio, en caso de ser necesario uno o varios prototipos.

La finalización de la fase inicial representa el primer hito importante del proyecto. Los criterios de evaluación en la etapa preliminar son:

– Concurrencia de las partes interesadas en la definición de alcance y estimación de costos/programas (horarios).

– Evidencia de entendimiento de los requerimientos mediante la comprobación de los casos de uso primarios.

– Credibilidad de los estimados de costos/programas, prioridades, riesgos y proceso de desarrollo.

– Detalle y extensión de cualquier prototipo arquitectónico que se ha desarrollado.

– Gastos actuales Vs. Gastos planeados.

Elaboración

El propósito de la fase de desarrollo es analizar el área problemática y crear una línea de base arquitectura, desarrollar planes de proyecto y eliminar los factores que suponen mayor riesgo para el proyecto. Para lograr estos objetivos, se necesita una comprensión general del sistema. Además, decisiones relacionadas con la arquitectura, se deben hacer con una comprensión de todo el sistema: su alcance, requisitos funcionales y no funcionales según los requerimientos de rendimiento.

Fácilmente se podría argumentar que la etapa de desarrollo es la más importante de las cuatro. Decidir si se continúa con la fase de

construcción y transición. Si bien siempre es necesario realizar cambios en el proceso, las actividades durante la fase de desarrollo garantizan que la arquitectura, los requisitos y los planes sean lo suficientemente estables y que los riesgos se mitiguen lo suficiente como para justificar los costos. Dichos costos del proyecto y los cronogramas de finalización se pueden determinar en una manera predecible.

Durante la fase de desarrollo, se crea un prototipo de arquitectura ejecutable en una o más iteraciones, según el alcance, la escala, los riesgos y la novedad del proyecto.

El resultado de la fase de elaboración es:

– Un modelo de caso de uso (completo por lo menos el 80%), en donde todos los casos de uso y actores han sido identificados.

– Requerimientos suplementarios capturando los requerimientos no funcionales y cualquier requerimiento que no esté asociado con un caso de uso específico.

– Descripción de una Arquitectura de Software.

– Prototipo arquitectónico ejecutable.

– Lista de riesgos y casos de negocio revisados.

– Un plan de desarrollo para el proyecto global, incluyendo el plan del proyecto desglosado, mostrando iteraciones y criterios de evaluación para cada iteración.

– Un caso de desarrollo actualizado especificando el proceso que se usará.

– Un manual de usuario preliminar (opcional).

Al final de la fase de desarrollo, ocurre el segundo hito importante en el ciclo de vida. Esta sección analiza en detalle los objetivos y el alcance del sistema, las opciones arquitectónicas y las soluciones a problemas clave riesgo (Jaramillo, 2016).

Los criterios de evaluación en esta etapa deben responder a las siguientes preguntas:

– ¿La visión del producto es estable?

– ¿La arquitectura es estable?

– ¿La demostración ejecutable muestra que los elementos de mayor riesgo han sido direccionados y realmente resueltos?

– ¿El plan para la construcción de la fase fue lo suficientemente detallado y preciso?

– ¿Está respaldado con una base de estimaciones creíble?

– ¿Todas las partes interesadas están de acuerdo en que la visión actual puede ser alcanzada si el plan actual se ejecuta para desarrollar el sistema completo, en el contexto de la arquitectura actual?

– ¿Los gastos actuales vs. los gastos planeados son aceptables?

Construcción

En la etapa de ensamblaje, se completan todos los componentes faltantes y la funcionalidad de la aplicación. Se desarrolla e integra en el producto y se prueban todas las funciones. En cierto sentido, la fase de construcción es un proceso de producción que enfatiza la gestión y el control de los recursos. Actividades para optimizar costos, cronograma y calidad, en ese sentido, la psicología. La gerencia supervisa la transición del desarrollo de propiedad intelectual durante la fase de construcción y desarrollo de productos listos para su implementación.

Muchos proyectos son tan grandes que la construcción demora aproximadamente se puede crear paralelismo. Estas operaciones paralelas pueden acelerar las cosas significativamente potenciales de implementación; pero también pueden complicar la gestión de recursos

y la sincronización del flujo de trabajo. Una arquitectura confiable es más fácil de construir. Ésta es una de las razones por las que en la fase de diseño se presta especial atención al desarrollo sostenible de la arquitectura y la planificación. El resultado de la fase de construcción es un producto listo para ser entregado al usuario final, incluye al menos:

– Producto de software integrado en la plataforma adecuada.

– Manuales de usuario.

– Descripción de la versión actual.

Al final de la fase de construcción, ocurre el tercer paso importante, aquí es donde se decide si el software se utilizará o no. Los sitios y los usuarios están listos para comenzar sin poner el proyecto en alto riesgo. Este hito a menudo se denomina "beta" (Jaramillo, 2016).

Los criterios de evaluación en la etapa de construcción deben responder a las siguientes preguntas:

– ¿La versión del producto es suficientemente estable y madura para ser desplegada a la comunidad de usuarios?

– ¿Están todas las partes interesadas listas para la transición en la comunidad de usuarios?

– ¿Los costos actuales vs los costos planeados siguen siendo aceptables?

Transición

El propósito de la fase de transición es justamente la transformación del producto de software en la comunidad de usuarios. Una vez que el producto se entrega al usuario final, surgirán problemas que requerirán nuevas versiones, solucionar algunos problemas o agregar funciones retrasadas.

La fase de transición ocurre cuando la versión base está lo suficientemente avanzada como para implementarse en los dominios de los usuarios finales. Esto casi siempre requiere que algún subconjunto útil del sistema se complete con un nivel aceptable de calidad y la disponibilidad de documentación del usuario para que la transición dé frutos positivos para todas las partes. Según Jaramillo (2016) incluye:

– Pruebas "beta" para validar el nuevo sistema en contraste con las expectativas del usuario.

– Operación paralela con un sistema heredado al cual está reemplazando.

– Conversión de bases de datos operacionales.

– Capacitación a usuarios y personal de mantenimiento.

– Emprender el producto al mercado, distribución y grupos de venta.

La fase de transición se centra en las actividades necesarias para entregar el software a los clientes. Normalmente, esta fase incluye varias iteraciones incluyendo versiones beta, versiones generales disponibles, bug-fix 3 y versiones de mejora. En esta etapa del ciclo de vida, primero se deben tener en cuenta los comentarios de los usuarios para ajustar, personalizar, instalar y probar el producto en busca de problemas de usabilidad.

Las principales tareas del período de transición incluyen:

– Lograr que el usuario pueda usar el producto por sí mismo.

– Lograr que la concurrencia desplegada de las partes interesadas esté completa y consistente con el criterio de evaluación de la visión.

– Lograr la línea base del producto final tan rápida y económicamente efectiva como sea posible (Amaro & Valverde, 2007).

Al final del período de transición, se alcanzará el cuarto hito del proyecto. En esta etapa se decide si se han logrado los objetivos y si se debe iniciar un nuevo ciclo de desarrollo.

Los criterios de evaluación para esta fase responden las siguientes preguntas:

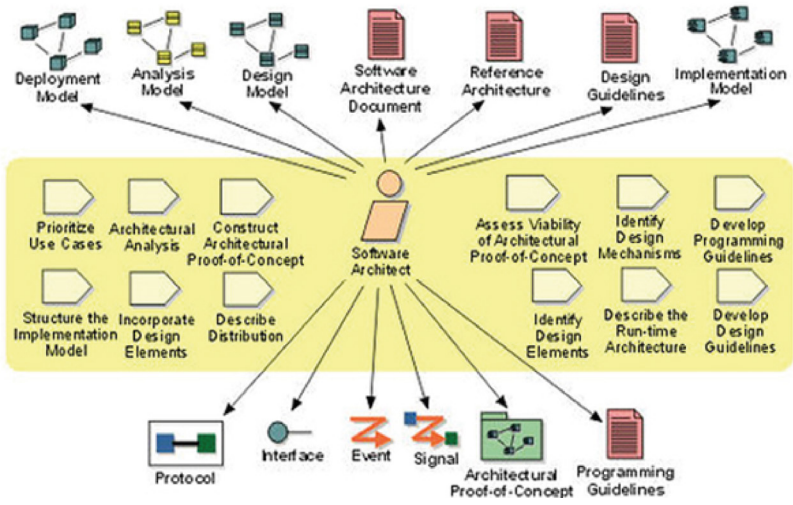
– ¿El usuario está satisfecho?

– ¿Los costos actuales vs. los costos planeados siguen siendo aceptables?

1.4.7. Actores – Roles

Figura 5

Roles RUP



Nota. Los roles principales de RUP son los analistas, los desarrolladores y los gestores. Tomado de Ingenieria de Software (2019) https://unidad2ingesoftware.blogspot.com/2019/10/roles-rup.html

Son las personas responsables de realizar las actividades especificadas en los procedimientos de trabajo de cada departamento de RUP. Para Rueda

(2006), estos actores se dividen en varias categorías: Analistas, Desarrolladores, Testers, Gerentes y Otros. A continuación se muestra la lista:

Actores Principales en RUP:

– Desarrolladores: Son responsables de la implementación del software.

– Ingenieros de Pruebas: Se encargan de realizar pruebas para garantizar la calidad del software.

– Analistas de Negocios: Definen los requisitos del sistema y aseguran que el software cumpla con las necesidades del negocio.

– Gerentes de Proyecto: Supervisan y coordinan el desarrollo del proyecto.

– Usuarios Finales: Aquellos que utilizarán el software una vez completado.

Roles en RUP:

– Roles Técnicos: Incluyen al Arquitecto de Software, Diseñador, Programador, etc.

– Roles de Gestión: Incluyen al Gerente de Proyecto, Coordinador de Configuración, etc.

– Roles de Soporte: Incluyen al Especialista en Soporte Técnico, Documentador Técnico, etc.

– Roles de Negocio: Incluyen al Analista de Negocios, Usuario Final, etc.

1.5. Diferencias entre Metodologías Tradicionales y Ágiles

Enfoque de Desarrollo:

Metodologías Tradicionales: Adoptan un enfoque predictivo y planificado, con una estructura de fases secuenciales.

Metodologías Ágiles: Se basan en un enfoque adaptativo e iterativo, donde el desarrollo es reactivo a cambios y se realiza en ciclos cortos.

Flexibilidad ante Cambios:

Metodologías Tradicionales: Cambios en los requisitos pueden ser costosos y difíciles de manejar después de iniciar una fase.

Metodologías Ágiles: Son inherentemente flexibles, permitiendo adaptaciones continuas a los cambios en los requisitos incluso en etapas avanzadas del proyecto.

Fases del Proyecto:

Metodologías Tradicionales: Progresan de manera lineal a través de fases como análisis, diseño, implementación, pruebas y mantenimiento.

Metodologías Ágiles: Utilizan ciclos de desarrollo iterativos e incrementales, entregando partes funcionales del producto en cortos períodos de tiempo.

Documentación:

Metodologías Tradicionales: Priorizan la documentación exhaustiva en cada fase del proyecto.

Metodologías Ágiles:

Aunque mantienen la documentación, dan más importancia a la comunicación cara a cara y la entrega de software funcional.

Colaboración y Comunicación:

Metodologías Tradicionales: La comunicación está más estructurada y a menudo ocurre a través de documentos formales.

Metodologías Ágiles: Fomentan la colaboración continua y la comunicación directa entre los miembros del equipo y con los interesados.

Entrega de Producto:

Metodologías Tradicionales: La entrega del producto final suele ocurrir al final del proyecto.

Metodologías Ágiles: Buscan entregar valor de manera continua, entregando versiones funcionales del producto en intervalos cortos.

Gestión de Riesgos:

Metodologías Tradicionales: La gestión de riesgos se realiza principalmente al comienzo del proyecto.

Metodologías Ágiles: La gestión de riesgos es continua y se aborda de manera iterativa durante todo el ciclo de vida del proyecto.

Participación del Cliente:

Metodologías Tradicionales: La participación del cliente suele ser más limitada y se espera que se defina completamente al inicio del proyecto.

Metodologías Ágiles: Se enfocan en la colaboración activa con el cliente durante todo el desarrollo, permitiendo cambios y adaptaciones basadas en la retroalimentación continua.

Autoorganización de Equipos:

Metodologías Tradicionales: La organización del trabajo suele ser más jerárquica y dirigida desde arriba.

Metodologías Ágiles: Fomentan la autoorganización de los equipos, dándoles más autonomía en la toma de decisiones y la organización del trabajo.

Resultados Predecibles vs. Adaptabilidad:

Metodologías Tradicionales: Buscan resultados predecibles y controlables desde el principio.

Metodologías Ágiles: Se centran en la adaptabilidad y la capacidad de responder eficientemente a cambios en los requisitos o condiciones del proyecto.

10. Actividad de Aprendizaje

Tema: Tipos de Metodologías

1. Investigar al menos tres tipos diferentes de metodologías de desarrollo de software, puede considerar enfoques tradicionales, ágiles u otros emergentes.

2. Realizar una comparación entre las metodologías seleccionadas.
Destaque sus diferencias y similitudes en términos de enfoque, flexibilidad, comunicación, velocidad de entrega, y adaptabilidad a cambios en los requisitos.

Tema: Metodologías Tradicionales

1. Describir las características clave de tres metodología tradicionales.
Enfocarse en aspectos como la planificación, la documentación, la secuencia de fases y la gestión de cambios.

2. Enumerar las ventajas y desventajas de cada metodología tradicional. Considerar aspectos como la predictibilidad, la gestión de riesgos, la adaptabilidad a cambios y la participación del cliente

3. Reflexionar sobre la idoneidad de las metodologías tradicionales en entornos específicos de desarrollo de software.
¿En qué contextos pueden ser más efectivas y por qué?

Tema: Metodologías tradicionales y Ágiles

1. Realizar una breve comparación entre las metodologías tradicionales y las metodologías ágiles en términos de flexibilidad, velocidad de entrega y adaptabilidad a cambios.

Tema: Rational Unified Process (RUP)

1. Proporcionar una descripción general del Rational Unified Process (RUP). Incluya su origen, propósito y los principios fundamentales en los que se basa.

2. Desglosar las fases principales del ciclo de vida en RUP, como Inicio, Elaboración, Construcción y Transición. Explicar las actividades clave que se llevan a cabo en cada fase.

3. Enumerar y describir los roles clave en RUP, como el Analista de Negocios, el Arquitecto de Software, el Desarrollador, etc. Detalla las responsabilidades asociadas con cada rol.

11. Autoevaluación

1. ¿Qué significa la sigla RUP?
a) Proceso Unificado de Resultados
b) Proceso Unificado de Rendimiento
c) Proceso Unificado Racional
d) Proceso Unificado de Recursos

2. ¿Cuál de las siguientes fases no es parte del ciclo de vida de RUP?
a) Elaboración
b) Construcción
c) Transición
d) Investigación

3. ¿Cuál es el objetivo principal de la fase de Inicio en RUP?
a) Desarrollar el producto final
b) Establecer el alcance del proyecto y la viabilidad
c) Realizar pruebas exhaustivas
d) Implementar el sistema en producción

4. ¿Cuál es la característica clave de RUP en comparación con otros modelos de desarrollo?

a) Enfoque lineal
b) Iterativo e incremental
c) Cascada
d) Agile puro

5. ¿Qué artefacto se utiliza para describir la arquitectura del sistema en RUP?

a) Diagrama de Gantt
b) Documento de visión
c) Modelo de casos de uso
d) Diagrama de despliegue

12. Evaluación final

1. Qué es una metodología de desarrollo de software:

a) Códigos de programación qué sirve para automatizar el sistema.
b) Comprende los procesos a seguir sistemáticamente para idear, implementar y mantener un producto software.
c) En un proyecto de desarrollo de software la metodología ayuda a no definir: quién debe hacer, que cuándo y cómo debe hacerlo.
d) Ninguno

2. Qué tipos de metodologías de desarrollo de software existe:

a) Ágiles, convencionales
b) Tradicionales, ágiles
c) Transaccionales, ágiles
d) Ágiles, convencionales

3. Cuál es el ciclo de vida del software del RUP?

a) Objetivo, capacidad operacional, arquitectura, entregable del producto
b) Diseño, análisis, desarrollo, pruebas e implementación
c) Inicio, desarrollo, construcción, transición
d) Inicio, elaboración, construcción, transición

4. Seleccione una característica de las metodologías tradicionales

a) Permite adaptar el software a las necesidades que van surgiendo en el camino
b) Son poco flexibles, lineal y secuencial ante cambios de un entorno volátil

c) Se basan en la metodología incremental
d) Los equipos trabajan en entregas pequeñas y funcionales llamadas iteraciones, generalmente de corta duración

5. Dentro de la fases del RUP podemos indicar la fase de incepción. Las principales actividades involucradas en la captura de requerimientos son:
a) Identificar actores y casos de uso, Priorizar los casos de uso, Detallar casos de uso, Prototipar la interfaz con el usuario, Estructurar el modelo de casos de uso
b) Las realizaciones de casos de uso a nivel de diseño, Los subsistemas a nivel de diseño, Las especificaciones de Interfaces
c) Especificaciones de Interfaces, La descripción de la arquitectura, Plan de construcción e integración
d) Identificar clases y base de datos, Priorizar los casos de Uso, Detallar casos de uso, Prototipar la interfaz con el usuario, Estructurar el modelo de casos de uso.

6. ¿Cuál metodología emplea un lenguaje de modelamiento unificado?
a) SCRUM
b) RUP
c) CASCADA
d) XP

7. ¿ La finalidad principal de esta fase es alcanzar la capacidad operacional del producto de forma incremental a través de las sucesivas iteraciones?
a) Desarrollo
b) Inicio
c) Construcción
d) Elaboración

8. ¿ Cuáles son las fases principales del ciclo de vida en Rational Unified Process (RUP)?
a) Análisis, Desarrollo, Implementación, Mantenimiento
b) Inicio, Construcción, Despliegue, Evaluación
c) Iniciación, Elaboración, Construcción, Transición
d) Diseño, Pruebas, Implementación, Entrega

9. ¿Cuáles son algunos de los elementos clave en Rational Unified Process (RUP)?
a) Historias de usuario, Sprints, Reuniones diarias
b) Roles, Flujos de trabajo, Artefactos y fases de desarrollo

c) Épicas, Tableros Kanban, Desarrollo iterativo
d) Scrum Master, Product Owner, Backlog de product

10. ¿Cuál es el proceso de ciclo de vida de RUP?

a) El ciclo de vida RUP es una implementación del Desarrollo en casada
b) El ciclo de vida RUP es una implementación del Desarrollo en espiral
c) El ciclo de vida RUP es una implementación del Desarrollo en Scrum
d) El ciclo de vida RUP es una implementación del Desarrollo en Xp

13. Solucionario de autoevaluación

Evaluación final

1. Qué es una metodología de desarrollo de software:

a) Códigos de programación qué sirve para automatizar el sistema.
b) Comprende los procesos a seguir sistemáticamente para idear, implementar y mantener un producto software.
c) En un proyecto de desarrollo de software la metodología ayuda a no definir: quién debe hacer, que cuándo y cómo debe hacerlo.
d) Ninguno

2. Qué tipos de metodologías de desarrollo de software existe:

a) Ágiles, convencionales
b) Tradicionales, ágiles
c) Transaccionales, ágiles
d) Development, unidimensionales

3. Cuál es el ciclo de vida del software del RUP?

a) Objetivo, capacidad operacional, arquitectura, entregable del producto
b) Diseño, análisis, desarrollo, pruebas e implementación
c) Inicio, desarrollo, construcción, transición
d) Inicio, elaboración, construcción, transición

4. Seleccione una característica de las metodologías tradicionales

a) Permite adaptar el software a las necesidades que van surgiendo en el camino
b) Son poco flexibles, lineal y secuencial ante cambios de un entorno volátil
c) Se basan en la metodología incremental
d) Los equipos trabajan en entregas pequeñas y funcionales llamadas iteraciones, generalmente de corta duración

5. Dentro de la fases del RUP podemos indicar la fase de incepción. Las principales actividades involucradas en la captura de requerimientos son

a) Identificar actores y casos de Uso, Priorizar los casos de uso, Detallar casos de uso, Prototipar la interfaz con el usuario, Estructurar el modelo de casos de uso

b) Las realizaciones de casos de uso a nivel de diseño, Los subsistemas a nivel de diseño, Las especificaciones de interfaces

c) Especificaciones de interfaces, La descripción de la arquitectura ,Plan de construcción e integración

d) Identificar clases y base de datos, Priorizar los casos de uso, Detallar casos de uso, Prototipar la interfaz con el usuario, Estructurar el modelo de casos de uso.

6. ¿Cuál metodología emplea lenguaje de modelamiento unificado?

a) SCRUM

b) RUP

c) CASCADA

d) XP

7. ¿ La finalidad principal de esta fase es alcanzar la capacidad operacional del producto de forma incremental a través de las sucesivas iteraciones ?

a) Desarrollo

b) Inicio

c) Construcción

d) Elaboración

8. ¿ Cuáles son las fases principales del ciclo de vida en Rational Unified Process (RUP)?

a) Análisis, Desarrollo, Implementación, Mantenimiento

b) Inicio, Construcción, Despliegue, Evaluación

c) Iniciación, Elaboración, Construcción, Transición

d) Diseño, Pruebas, Implementación, Entrega

9. ¿Cuáles son algunos de los elementos clave en Rational Unified Process (RUP)?

a) Historias de usuario, Sprints, Reuniones diarias

b) Roles, Flujos de trabajo, artefactos y fases de desarrollo

c) Épicas, Tableros Kanban, Desarrollo iterativo

d) Scrum Master, Product Owner, Backlog de product

10. ¿Cuál es el proceso de ciclo de vida de RUP?

a) El ciclo de vida RUP es una implementación del Desarrollo en casada

b) El ciclo de vida RUP es una implementación del Desarrollo en espiral

c) El ciclo de vida RUP es una implementación del Desarrollo en Scrum
d) El ciclo de vida RUP es una implementación del Desarrollo en Xp

14. Glosario

A

Actor: Individuo, equipo o sistema que interactúa con el sistema siendo desarrollado.

Artefacto: Cualquier cosa que sea producida, modificada o utilizada por un proceso.

C

Ciclo de Vida: Las fases a través de las cuales un proyecto evoluciona desde la concepción hasta la entrega y posterior mantenimiento.

Casos de Uso: Descripciones de cómo actores externos interactúan con el sistema.

Ciclo de Desarrollo: Las fases específicas y las actividades que se realizan durante el desarrollo del sistema.

D

Diagrama de Actividades: Representación gráfica del flujo de trabajo en un proceso.

Diagrama de clases: Representación visual de las clases en el sistema y sus relaciones.

E

Especificación de requisitos: Documento que describe las necesidades y expectativas del sistema.

I

Iteración: Un enfoque repetitivo e incremental para el desarrollo que refina y amplía gradualmente el sistema.

M

Modelo de Dominio: Representación visual de las entidades clave dentro del sistema y sus relaciones.

Modelo de Despliegue: Representación visual de cómo los componentes del sistema se distribuyen en el hardware.

P

Prototipo: Una versión temprana y simplificada del sistema que se utiliza para obtener retroalimentación rápida.

R

Riesgo: Un evento o condición incierta que, si ocurre, tiene un impacto positivo o negativo en el proyecto.

T

Transición: Son actividades necesarias para entregar el software a los clientes

15. Referencias Bibliográficas

– Espinoza, A. (15 de febrero de 2017). https://pirhua.udep.edu.pe/. Obtenido de Manual para elegir una metodología de desarrollo de software dentro dee un proyecto informático: https://pirhua.udep.edu.pe/items/abd6e6c1-56be-412b-bb6b-67d08f962b2e

– Alaimo, D. (2013). Proyectos Ágiles con Scrum. Buenos Aires: Kleer.

– Amaro, S., & Valverde, J. (21 de Marzo de 2007). Metodologías Ágiles. Obtenido de Escuela Informática: https://d1wqtxts1xzle7.cloudfront.net/53222887/Metodologias_Agiles-libre.pdf?1495404476=&response-content-disposition=inline%3B+filename%3DUniversidad_Nacional_de_Trujillo.pdf&Expires=1704228181&Signature=QGskhRVbvtArI88T1ZJSImJPUICd844JUf9JN-8gis-nmc-4Shg

– Garcia, S. (28 de Agosto de 2010). Universidad Guatemala. Obtenido de Metodología RUP: https://googlegroups.com/group/analisisdesistemasaumg/attach/aa017922c4817fac/investigaci%C3%B3n%20metodolog%C3%ADa%20RUP.pdf?part=0.1

– Cardenas, J. (17 de abril de 2017). Unab. Obtenido de Evalución de Metodologías de Desarrollo de Software utilizadas para valorar el impacto de proyectos de software: https://repository.unab.edu.co/bitstream/handle/20.500.12749/3412/2017_Tesis_Cardenas_Castellanos_Marlon%20_Jose.pdf?sequence=1

– Morales, I. (21 de septiembre de 2015). upo. Obtenido de Metodologías de Desarrollo Software. ¿Tradicional o Ágil?: https://www.upo.es/cms1/export/sites/upo/moleqla/documentos/Numero19/Nxmero_19.pdf

– Navarro, A., Fernandez, J., & Morales, J. (10 de julio de 2013). Redalyc.org. Obtenido de Revisión de metodologías ágiles para el desarrollo de software: https://www.redalyc.org/

Pérez, A. (2011). Cuatro enfoques metodológicos para el desarrollo de Software RUP – MSF – XP - SCRUM. Colombia: INVENTUM.

– Pérez-Ilzarbe, P. (21 de abril de 2004). Grupo de Estudios Perceanos. Obtenido de Grupo de Estudios Perceanos: https://www.unav.es/gep/PerezIlzarbeSeminarioVaz.html
– Serna, J. (16 de marzo de 2019). UAMI. Obtenido de Investigaci'on y Fundamento de la Metodologia RUP: https://www.academia.edu/19661140/1PROY1_METODOLOGIA_RUP
– Rueda, J. (23 de marzo de 2006). Usac. Obtenido de Aplicación de la metodología rup para el desarrollo rápido de aplicaciones basado en el estándar J2EE: http://biblioteca.usac.edu.gt/tesis/08/08_0308_CS.pdf
– Jaramillo, W. (25 de abril de 2016). Puce. Obtenido de Aplicación de la metodología RUP y el patrón de diseño MVC en la construcción de un sistema de gestión académica para la Unidad Educativa Ángel de la Guarda: http://repositorio.puce.edu.ec/bitstream/handle/22000/11264/Documento%20Disertaci%C3%B3n%20Wendy%20Jaramillo.pdf?sequence=1&isAllowed=y

16. Anexos o recursos

–https://recimundo.com/index.php/es
– https://www.google.com.ec/books/edition/Desarrollo_Global_de_Software/po2fDwAAQBAJ?hl=es-419&gbpv=1
– https://www.google.com.ec/books/edition/El_proceso_unificado_de_desarrollo_de_so/zHKbQgAACAAJ?hl=es-419
– https://www.google.com.ec/books/edition/Metodolog%C3%ADa_del_an%C3%A1lisis_estructurado/PUqxsNVaQC8C?hl=es-419&gbpv=1&dq=-METODOLOGIA+desarrollo+software&printsec=frontcover
–https://www.youtube.com/watch?v=Li4K_QgqcMg
–https://www.youtube.com/watch?v=_6G2o7O54no
–https://www.youtube.com/watch?v=_6G2o7O54no
–https://virtual.cun.edu.co/contenidos/migracion2020/sistemas/s8/refinamientoenproducciondesoftware/u2/recurso7.pdf

INSTITUTO SUPERIOR
TECNOLÓGICO
VICENTE LEÓN

Guía general de estudio de la asignatura

Agosto 2024

ISBN: 978-9942-676-38-2