UNIVERSIDAD DON BOSCO

FACULTAD DE INGENIERÍA

TRABAJO DE GRADUACIÓN PARA OPTAR AL GRADO DE

INGENIERO EN CIENCIAS DE LA COMPUTACIÓN

MONOGRAFIA SOBRE LA METODOLOGIA DE DESARROLLO DE SOFTWARE, RATIONAL UNIFIED PROCESS (RUP)

PRESENTADO POR

BELLOSO CICILIA, CLAUDIA IVONNE

ASESOR

ING. MELVIN CARIAS

SEPTIEMBRE 2009

EL SALVADOR, CENTROAMERICA.

UNIVERSIDAD DON BOSCO

FACULTAD DE INGENIERÍA

RECTOR

ING. FEDERICO MIGUEL HUGUET

SECRETARIO GENERAL

ING. YESENIA XIOMARA MARTINEZ OVIEDO

DECANO FACULTAD DE INGENIERIA

ING. ERNESTO GODOFREDO GIRON

SEPTIEMBRE 2009

EL SALVADOR, CENTROAMERICA.

UNIVERSIDAD DON BOSCO

FACULTAD DE INGENIERÍA

TRABAJO DE GRADUACIÓN PARA OPTAR AL GRADO DE

INGENIERO EN CIENCIAS DE LA COMPUTACIÓN

MONOGRAFIA SOBRE LA METODOLOGIA DE DESARROLLO DE SOFTWARE, RATIONAL UNIFIED PROCESS (RUP)

_____	_____
LECTOR	**ASESOR**

SEPTIEMBRE 2009

EL SALVADOR, CENTROAMERICA.

AGRADECIMIENTOS:

- En primer lugar a Dios todo poderoso que me ha permitido llegar hasta aquí, que me dio la bendición de culminar mis estudios superiores, a la virgen María quien también estuvo en los momentos difíciles

- A mi madre, a quien le debo este logro, quien siempre me apoyo desde el inicio de mi carrera, tanto en lo económico, como en lo moral, a mi padre que está en el cielo que se que desde ahí, ah visto y apoyado cada uno de mis pasos

- A mis compañeros de Universidad y a mis compañeros de trabajo, asesor, amigos y todas las personas que han creído en mi, y que de una u otra forma han ayudado a la finalización de este trabajo

Claudia Ivonne Belloso Cicilia

INDICE:

INDICE DE IMÁGENES:

CAPITULO I
MARCO DE REFERENCIA

1. Introducción:

La presente exposición se refiere al tema “Rational Unified Process (RUP)” el cual puede ser definido como un proceso de ingeniería de software, para producir software de calidad, que cumpla con las normas a nivel mundial y que ofrezca flexibilidad en plazos y presupuestos.

Gracias a la incorporación de las mejores prácticas de la ingeniería de software, como: gestión de requisitos, uso de arquitectura de componentes, modelado visual, verificación continua de la calidad y control de cambios entre otros. Convierte a RUP en una de las metodologías estándares más utilizadas para el análisis, implementación y documentación de sistemas orientados a objetos.

La importancia de la investigación de esta metodología radica, en los altos niveles de calidad de software exigidos por las grandes empresas. RUP puede ser utilizada sin importar el tamaño y rubro de la organización, sin embargo es más utilizada en las grandes empresas, debido a la complejidad y tamaño de los sistemas, en la actualidad el 50% de E-business y un aproximado de 1000 empresas multinacionales como: VISA, XEROX , VOLVO .etc. Utilizan RUP en sus procesos de desarrollo de software.

1.1 Antecedentes

1.1.1 Breve Historia de RUP

El antecedente más importante se ubica en el año 1967 con la metodología Ericsson (Erickson Approach) elaborada por Iván Jacobson, una aproximación de desarrollo basada en componentes, que introdujo el concepto de casos de uso

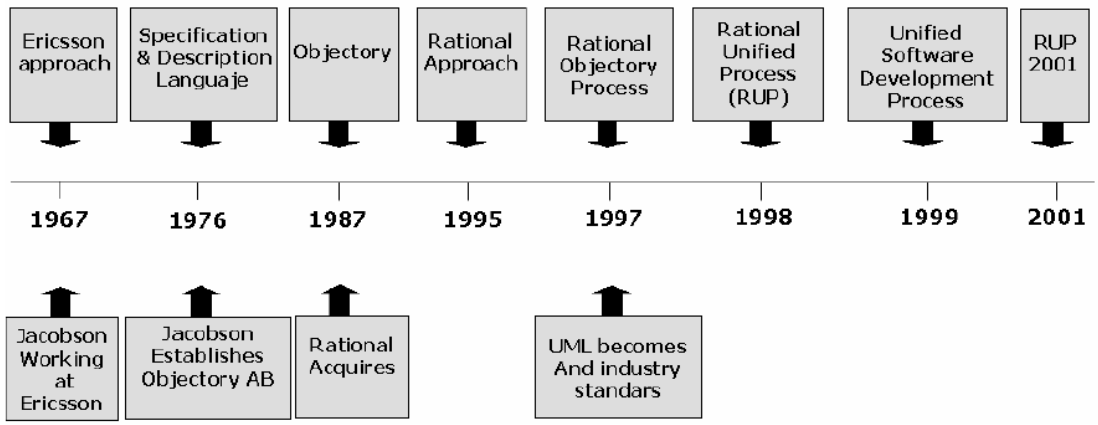


Figura 1 Historia de RUP

Los orígenes de RUP se remontan al modelo espiral original de **Barry Boehm**. **Ken Hartman**, uno de los contribuidores claves de **RUP** colaboró con Boehm en la investigación.

Entre los años 1987 a 1995 Jaconson fundó la compañía Objetoty AB y lanza en proceso de desarrollo Objectory (Abreviatura de Object Factory), En 1995 Rational Software es comprada por Objectory AB, propiedad de Jacobson
RUP fue el resultado de una convergencia de Rational Approach y Objectory, proceso desarrollado por el fundador de Objectory Ivar Jacobson. El primer resultado de esta fusión fue el Rational Objectory Processs, la primera versión de RUP fue puesta al mercado en 1998, siendo arquitecto en jefe Philippe kruchten.

También se conoce por este nombre al software desarrollado por Rational de IBM, el cual incluye información entrelazada de diversos artefactos y descripciones de las diversas actividades, está incluido en el rational Method Composer (RMC) que permite la personalización de acuerdo a necesidades de los proyectos.

1.1.2 Investigaciones Realizadas en El Salvador

De un total de tres empresas encuestadas las tres coinciden en el empleo de RUP en sus procesos de desarrollo de Software, un dato importante es que estas compañías se dedica exclusivamente al desarrollo de Software como por ejemplo SVSOFT S.A de C.V; [1] y se apegan a las tecnologías a nivel internacional.

En cuanto a conocimientos, esta metodología no es conocida por la mayor parte de estudiantes de la facultad de ingeniería de la Universidad Don Bosco (de un total de 74 encuestados) únicamente un 41% conocen acerca de la metodología, y sus cocimientos han sido adquiridos a través de investigaciones en Internet, o les han comentado sobre esta metodología.

En las nuevas revisiones de planes curriculares de escuelas de computación ya forma parte de algunos temarios de Escuelas de Ingeniería en Sistemas como por ejemplo en la Universidad Don Bosco, en la materia "Negocio en Internet", técnica electiva del área de sistemas de información, en donde se hace una mención de RUP.

[1] Website SVSFOT http://www.svsoft-sa.com/sp

1.2 Planteamiento del Problema:

Actualmente en El Salvador no se cuenta con suficiente información acerca de la metodología de desarrollo de software RUP, muchos profesionales y estudiantes del área de informática, desconocen el término RUP.

Debido al auge que este tiene en el desarrollo de aplicaciones empresariales a nivel internacional, y tomando en cuenta que algunas empresas orientadas exclusivamente a tecnologías de información en nuestro país están comenzando a adoptar esta metodología, se vuelve muy importante que todo Informático esté al tanto de las últimas tecnologías tanto de desarrollo de software como de programas de aplicación.

1.2.1 Definición del Tema:

Desarrollar un estudio monográfico que explore de una forma general los conceptos, lógicas de funcionamiento y ejemplos prácticos de la Metodología de desarrollo de Software RUP, que han sido implementados en Volvo IT; conocer sus ventajas y desventajas.

1.3 Justificación:

La búsqueda y adopción de instrumentos que fortalezcan y amplíen los conocimientos de los estudiantes es uno de los aspectos que ha sido muy tomado en cuenta por la Universidad Don Bosco.

El presente proyecto de investigación del Proceso Unificado de Rational (RUP) es un esfuerzo para propiciar la investigación por parte de los estudiantes y/o profesionales en el área de informática sobre las últimas metodologías de desarrollo de software, todo ello con el fin de crear profesionales integrales listos para afrontar las nuevas tecnologías y paradigmas de trabajo.

Se espera que este trabajo sirva como base para futuras investigaciones. Tomando en cuenta que este contenido ya forma parte del temario de la materia "**Negocios en Internet"**, impartida por la Universidad Don Bosco y que en la actualidad esta es una de las herramientas más utilizadas gracias a un gran número de casos de éxito. Alrededor 1000 empresas grandes a nivel internacional como Ericsson, Alcatel, Visa, Xerox, Volvo, Intel, MCI, etc. la utilizan al igual que el 50% de E-Business; Con el fin de determinar de una forma general la importancia que tiene la aplicación de metodologías de desarrollo en el campo laboral real de nuestro país.

1.4 Objetivos:

1.4.1 Objetivo General:

Elaborar una monografía de RUP (Proceso Unificado Rational) que presente los conceptos, elementos, ciclo de vida, casos de éxito de empresas internacionales como VOLVO IT que ha adoptado esta metodología, así como conocer la percepción de los profesionales del área de informática de nuestro país sobre este tema, con el propósito que esta investigación sirva de base a los estudiantes de la Universidad Don Bosco, para futuras investigaciones sobre RUP, metodologías de desarrollo de Software y técnicas de Ingeniería de Software.

1.4.2 Objetivos Específicos

1. Crear un documento que describa los conceptos generales de Ingeniería de Software y las metodologías de desarrollo de software.

2 Conocer de forma general el grado de conocimiento y utilización de RUP en nuestro país.

3 Mostrar la importancia que tiene el uso de estas metodologías en un ambiente de real de trabajo.

4 Describir los elementos que forman parte de la metodología RUP, componentes básicos de RUP, ciclo de vida de la metodología.

5 Presentar el caso práctico de VOLVO IT que ha adoptado esta metodología en su proceso de desarrollo de software.

1.5 Alcances

Dar a conocer conceptos básicos de las metodologías de desarrollo de software, componentes básicos de RUP, ciclo de vida de la metodología, y ejemplos reales de aplicación, de empresas a nivel Latinoamericano, para este proyecto se presentara el caso de VOLVO IT.

Redactar un documento que contenga los conceptos básicos de RUP, así como sus ventajas y desventajas tomando en cuenta la percepción de esta metodología por parte de los profesionales/empresas en nuestro país.

1.6 Limitantes

Entre algunos de los factores que podrían afectar el desarrollo de este proyecto se encuentran los siguientes:

Debido a que el tema no es muy conocido en nuestro medio son pocas las empresas que utilizan esta metodología, y son muy pocos los profesionales familiarizados por lo que resulta un poco difícil obtener datos y ejemplos de empresas nacionales.

1.7 Delimitantes:

Este estudio monográfico únicamente contendrá aspectos teóricos de la metodología RUP, y ejemplos de empresas de las cuales ya forma parte de su desarrollo de software, como es el caso de Volvo IT.

Este proyecto no ahondara en temas de diseño y programación del sistema RUP desarrollado por IBM.

1.8 Marco Teórico:

La ingeniería de Software es una de las ramas de las ciencias de la computación, la cual nos brinda conocimientos, técnicas y metodologías para desarrollar software de calidad brindándonos un enfoque sistemático, disciplinado y cuantificable.

El proceso de desarrollo de software por lo tanto es el conjunto de actividades necesarias para transformar los requisitos del cliente en un sistema de software

Figura 2. Esquema lógico de proceso de desarrollo de Software

Los procesos de desarrollo de software juegan un papel muy importante desde la concepción de un sistema de software, hasta su culminación interviniendo en todas las fases del sistema desde los requerimientos de los usuarios hasta el desarrollo y puesta en funcionamiento de cada una de las etapas e iteraciones.

La comunicación que debe tener en el desarrollo de un sistema es uno de los principales causantes de fallas en los proyectos de software ya que ni los desarrolladores, jefes de Software, usuarios, etc. están en contacto continuo para evaluar las fases de desarrollo.

El principio básico de desarrollar software de calidad nos lleva a la búsqueda de las mejores metodologías de desarrollo, que mitiguen los principales riesgos de

fracasos en los sistemas, detectados cuando no se utiliza o se hace mal uso de una metodología de desarrollo de software.

En general las metodologías llevan a cabo una serie de procesos comunes que son buenas prácticas para lograr objetivos del proyecto, independientemente de cómo estos hayan sido diseñados. Las fases que agrupan estos procesos son: Análisis de especificaciones, diseño, programación, prueba, documentación, mantenimiento y reingeniería.

Así mismo las diferentes metodologías tienen diversos ciclos de vida del desarrollo de software, los modelos más utilizados son los siguientes:

- Modelo en cascada
- Modelo en espiral
- Modelo de prototipos
- Método en V
- Desarrollo por etapas

También podemos encontrar diferentes enfoques de las metodologías de desarrollo de software.

1.8.1 Metodología de Desarrollo de software orientada a objetos

Estas metodologías describen e implementan los sistemas de información desde un punto de vista ontológico.

Incluye los siguientes aspectos:

- Conceptos y diagramas
- Etapas y definiciones de entrega en cada una de ellas
- Actividades y recomendaciones

1.8.2 RUP (Rational Unified Process)

Es una metodología de desarrollo de software formal, orientadas a objetos, con un ciclo de vida espiral.

Este proceso de desarrollo de software utiliza el lenguaje unificado de modelado UML, y constituye una de las mejores y más utilizadas; para el análisis, implementación y documentación de sistemas orientados a objetos.

El RUP es un conjunto de metodologías adaptables al contexto y necesidades de cada organización.

Las principales características de RUP son:

- Dirigido por Casos de uso
- Centrado en arquitectura
- Iterativo e incremental

1.8.3 Dirigido por casos de uso:

Los casos de uso son una técnica que utilizamos para la captura de requisitos por parte de los clientes/usuarios, se define un caso de uso como un fragmento de funcionalidad del sistema que proporciona al usuario un valor añadido.

Los casos de uso representan los requisitos funcionales del sistema.[2]

En RUP los casos de uso guían el diseño, implementación y prueba de un producto de software y sirven como guía de trabajo en todas las fases, como podemos ver en la figura 4.

Figura 3 Integración del trabajo con los diagramas de caso de uso

1.8.4 Centrado en la Arquitectura :

La arquitectura es la organización o estructura de todas las partes más relevantes del sistema, la arquitectura juega un papel muy importante en el desarrollo de software ya que nos permite tener una visión común entre todos los involucrados en el proceso.

[2] [Ku00]

La arquitectura involucra los aspectos estáticos y dinámicos más significativos del sistema, está muy relacionada con la toma de decisiones de cómo debe ser desarrollado el sistema y que orden debe seguirse para tal fin.

La Arquitectura en RUP ocupa un papel muy importante, el establecimiento temprano de una buena arquitectura que haga frente a cualquier cambio posterior durante la construcción y el mantenimiento.

1.8.5 Interactivo e Incremental

Según [JRB00] el equilibrio correcto entre los casos de uso y la arquitectura es muy parecido al equilibrio de la forma y la función en el desarrollo del producto lo cual se consigue con el tiempo.

Tomando en cuenta estos conceptos, RUP apuesta por procesos interactivos e incrementales en donde el trabajo se divide en partes más pequeñas o mini proyectos permitiendo el equilibrio entre casos de uso y arquitectura.

1.9 Metodología de la Investigación:

Para la investigación del proyecto se utilizarán principalmente dos herramientas:

- La investigación documental, en el caso de conceptos de Ingeniería de Software, se hará uso de libros de texto, y otros materiales impresos en el caso de la investigación de RUP se hará principalmente con documentación en digital disponible en Internet, manuales de uso, ejemplos prácticos y foros de discusión.

- Se realizara una pequeña encuesta a setenta y cuatro [3]estudiantes de la facultad de Ingeniería en sistemas de la Universidad Don Bosco, y tres

[3] Datos obtenidos según fórmula para muestras en población finita, capitulo 4.6. página 65

profesionales en el área de informática, de empresas nacionales, dedicadas a la fabricación de software como por ejemplo SVSOFT esto con el fin de medir el grado de conocimiento que estos tienen referente a la metodología RUP.

CAPITULO II
INTRODUCCION A LA INGENIERIA DE SOFTWARE

2 Introducción:

No podemos comenzar hablando de una metodología de análisis de software, sin antes conocer de forma general cuales son las etapas generales, de desarrollo de software.

Un sistema informático como todos sabemos está compuesto por hardware y software. En cuanto al hardware, su producción se realiza sistemáticamente y la base de conocimiento para el desarrollo de dicha actividad está claramente definida. La fiabilidad del hardware es, en principio, equiparable a la de cualquier otra máquina construida por el hombre. A diferencia del software, el cual su construcción y resultados han sido históricamente cuestionados debido a los problemas asociados, entre ellos podemos destacar los siguientes: los sistemas no responden a las expectativas de los usuarios, los programas “fallan” con cierta frecuencia, los costes del software son difíciles de prever y normalmente superan las estimaciones, la modificación del software es una tarea difícil y costosa. El

software se suele presentar fuera del plazo establecido y con menos prestaciones de las consideradas inicialmente.

Normalmente, es difícil cambiar de entorno hardware usando el mismo software. El aprovechamiento óptimo de los recursos (personas, tiempo, dinero, herramientas, etc.) no suele cumplirse

Sin duda trabajar con software representa una complejidad muy grande, esto en parte por el alto grado de abstracción, y trabajo intelectual, algunos de los problemas surgidos en las fases de desarrollo como los mencionados anteriormente, se han venido dando desde hace mucho tiempo, y darle solución ha sido uno de los propósitos de la Ingeniería de software, la cual está en la búsqueda constante de las mejores herramientas, técnicas, metodologías que hagan frente a estas dificultades, desde las primeras fases del proyecto, evitando así demoras y desaprovechamiento de los recursos

La elección de la herramienta y/o metodología adecuada, dependerá en gran medida de las características propias del proyecto, considerando tanto la naturaleza del mismo, como la disponibilidad de recursos (Tecnológicos, Humanos y económicos).

Los grandes proyectos de software necesitan herramientas estables y completas, las cuales brinden medios a lo largo de todas las etapas del desarrollo del proyecto, Rational Unified Process (RUP) es una herramienta que brinda una gran cantidad de ventajas, una de las más importantes es el involucramiento de todos los actores en todas las etapas de desarrollo del proyecto, mejorando así la comunicación entre el usuario y el programador, reduciendo de esta forma los problemas de mala interpretación, y previniendo las posibles deficiencias del sistema, detectando estas de una manera rápida; debido a estas ventajas y muchas otras, RUP ha tenido una buena aceptación por parte de las empresas

dedicadas a la producción de software, en nuestro país, empresas como **svsoft**, ya han adoptado esta herramienta en su plan de trabajo.

Tomando en cuenta lo anterior y considerando el auge en el uso de esta herramienta, contrastado con el poco conocimiento que tienen los estudiantes universitarios del área de informática, la escuela de computación de la Universidad Don Bosco no es la excepción, en la materia **"Negocios por Internet"**, se hace mención a este tema.

2.1 Conceptos Básicos:

2.1.1 ¿Qué es Ingeniería de Software?

Haciendo una recopilación de todos los conceptos que se han dado sobre la Ingeniería de software, la podemos definir como la disciplina o área de la informática, que hace uso razonable de los principios de ingeniería con el objetivo de obtener soluciones informáticas económicamente factible y que se adapte a las necesidades de las empresas reales, tomando en cuenta los procesos de producción y mantenimiento de software que son desarrollados y modificados en el tiempo y con los costos estimados.

Esta ingeniería trata con áreas muy diversas de la informática y de las Ciencias de la Computación, tales como construcción de compiladores, Sistemas Operativos, o desarrollos Intranet/Internet, abordando todas las fases del ciclo de vida del desarrollo de cualquier tipo de Sistema de Información y aplicables a infinidad de áreas (negocios, investigación científica, medicina, producción, logística, banca, control de tráfico, meteorología, derecho, Internet e Intranet, etc.).

Algunas definiciones, dadas a través del tiempo son:

- “Ingeniería de Software es el estudio de los principios y metodologías para el desarrollo y mantenimiento de sistemas software” (Zelkovitz, 1978)

- “Ingeniería de software es la aplicación práctica del conocimiento científico al diseño y construcción de programas de computadora y a la documentación asociada requerida para desarrollar, operar y mantenerlos. Se conoce también como Desarrollo de Software o Producción de Software” (Bohem, 1976)

- Ingeniería de Software trata del establecimiento de los principios y métodos de la ingeniería a fin de obtener software de modo rentable, que sea fiable y trabaje en máquinas reales (Bauer, 1972).

- Es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del software; es decir, la aplicación de la ingeniería al software (IEEE, 1993).

En conclusión podemos decir que los cuatro autores anteriores, de manera diferente describen en si el principal objetivo de la ingeniería de software, la cual es el establecimiento y puesta en práctica de los principios y metodologías que nos lleven a un desarrollo eficiente de software en todas las etapas desde sus inicios hasta su implementación y mantenimiento.

2.1.2 ¿Qué es software de calidad?

Los requisitos del software constituyen la calidad del mismo. Debemos estar claros que el software no se certifica con calidad, no se puede medir la calidad del software de forma correcta debido a su naturaleza lo que se certifica son los procesos de desarrollo, la correcta consecución de los mismos.

El usuario final mide la calidad del software según los resultados que este le brinde, en la medida que cumple con los objetivos por los que fue creado y la optimización de los recursos utilizados, así como también el comportamiento que este tenga en el transcurso del tiempo. Hablamos de software de fácil adaptación de cambios y mejoras, es en ese sentido de que la calidad del software depende de quien la juzgue. Existen también formas más objetivas de evaluar el software, estas son: métricas, estadísticas, etc.

¿Cuáles son algunas de las normalizaciones bajo las cuales se rigen la certificación de los procesos de software?

ISO 9000, CMMI

2.1.2.1 **Medición del software**

En el software lo que se mide son atributos propios del mismo, dichos atributos se encuentran uno general y este a su vez en otros más simples de medir, la medición de estos no es un dato muy preciso ya que la descomposición del atributo genérico de calidad en otros sub-atributos se torna irreal, se mide con datos estadísticos no avalados, es imposible decir que la medición se hace en forma correcta.

El concepto de medida va de más a menos, va de lo general a lo concreto y lo concreto es asociado a la métrica, cuya combinación daría el nivel de calidad o seguridad del producto.

2.1.2.2 Tipos de medidas:

- Número de errores durante un periodo determinado.
- Fallo en la codificación o diseño de un sistema que causa que el programa no funcione correctamente.
- Tamaño de un producto informático (líneas de código)
- Métrica de punto función (IBM): relaciona funcionalidades que ofrece:
  - Estimación de costes y esfuerzos.
  - COCOMO [4]

[4] COCOMO, Modelo Constructivo de costes, por sus siglas en ingles (Constructive Cost Model)

¿Cuál es la utilidad de la medida del software?

Como hemos podido ver a lo largo del tiempo, las grandes empresas traducen la calidad de sus productos, en ahorros de costos y una mejora en general las empresas dedicadas al desarrollo del software, no son ajenas a esta concepción de calidad, debido a esto en los últimos años se han realizado extensos trabajos de investigación que ayuden a estandarizar y crear parámetros de medición para determinar la calidad del software, debido a la complejidad que esto representa, ya que el software es medido de forma cualitativa, por su naturaleza, a diferencia de otros productos que son medidos de forma cuantitativa

Los trabajos anteriormente mencionados, han dado lugar a un gran número de documentos que buscan definir y estandarizar los parámetros de medición de calidad del software, algunos de ellos son:

Familia de normas ISO 9000, en especial ISO 9126, El modelo de niveles de madurez CMM (Capability Maturity Model), el estándar para aseguramiento de planes de calidad del IEEE 730:1984,.en este trabajo se expondrá el esquema general de la norma ISO 9126, con el propósito de mostrar los elementos que deben ser considerados para la evaluación de la calidad de los sistemas informáticos.

Modelo ISO 9126

El Modelo de calidad establecido por el estándar ISO 9126, ha establecido un estándar internacional para la evaluación de la calidad del software, dicho trabajo publicado en 1992, bajo el nombre de **"Information Technology-software product evaluation: Quality characteristic and guidelines for their use"**, en el cual se establecen las características de calidad para productos de software.

El estándar ISO 9126, establece que cualquier componente de la calidad del software puede ser descrito en términos de una o más de seis características básicas las cuales son:

- Funcionalidad
- Confiabilidad
- Usabilidad
- Eficiencia
- Mantenibilidad
- Portabilidad

Cada una de las cuales se detalla a través de un conjunto de sub- características que permiten profundizar en la evaluación de la calidad del producto de software:

- Atributo de funcionalidad.
- Atributo de capacidad de respuesta frente a errores externos.
- Atributo de nivel de seguridad. La calidad no puede existir sin seguridad, un producto sin seguridad seria un producto sin calidad.

Las características de calidad de software, deben ser tomadas en cuenta a lo largo de todo el ciclo de vida del software y esta debe ser gestionada a diferentes niveles:

A. A nivel de producto:

Cuando nos centramos en el proceso de desarrollo de software y hacemos una serie de pruebas en paralelo con cada etapa, para detectar y corregir los posibles defectos que puedan surgir.

B. A nivel de proyecto:

Cuando nos centramos en controlar todas las fases y áreas de gestión de proyecto, implantando metodologías y mejores prácticas que aseguren la correcta gestión de las mismas.

C. A nivel de proceso:

Cuando nos centramos en gestionar todas las áreas de proceso de una organización, mediante la implantación de una metodología. Así se consigue tener Mayor información de los procesos de modo que pueda controlarse y mejorarse, y produzcan así un aumento de la calidad de los productos y servicios relacionados con ellos.

CAPITULO III

PROCESO DE DESARROLLO DE SOFTWARE

3 Introducción:

Un proceso de desarrollo de software tiene como propósito la producción eficaz y eficiente de un producto que reúna los requisitos del cliente.

Este proceso se puede apreciar en la figura 4, y es puramente intelectual, afectado por la creatividad y juicio de las personas involucradas. Aunque un proyecto de desarrollo de software es equiparable en muchos aspectos a cualquier otro proyecto de ingeniería, en el desarrollo de software hay una serie de desafíos adicionales, relativos esencialmente a la naturaleza del producto obtenido.

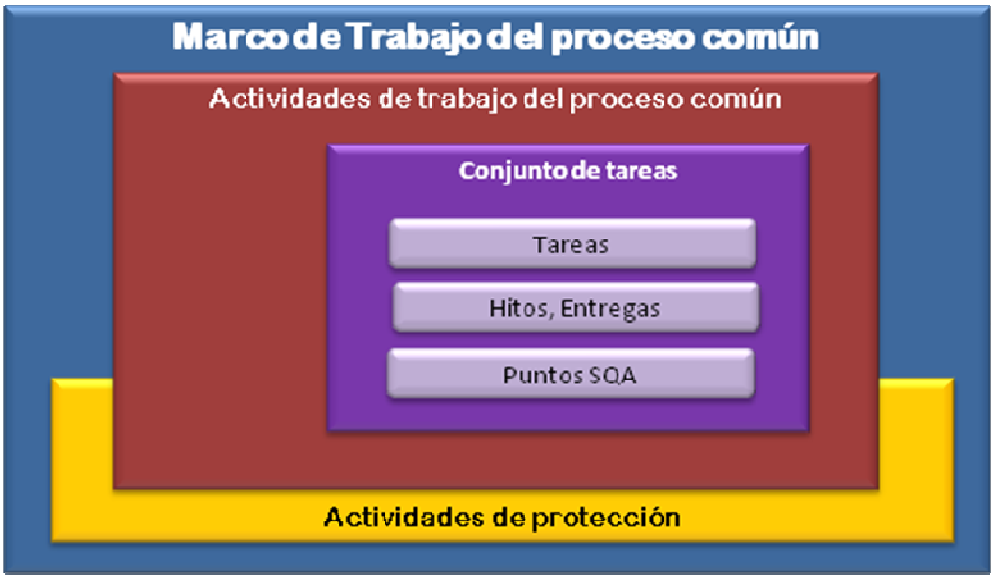


Figura 4: Proceso de desarrollo de software

3.1 Etapas del proceso de desarrollo de software:

Una etapa describe las actividades que hay que realizar para obtener un conjunto concreto de productos de desarrollo del software. Cada etapa se describe a partir de los siguientes conceptos:

Introducción: esta describe los objetivos de la etapa y la relación que esta tiene con otras etapas a lo largo del proceso

Flujos de trabajo: Una agrupación de actividades que se realizan juntas. Además, se identifican los productos de desarrollo de software que se obtienen en cada actividad

Actividades: La definición de todas las actividades se llevan a cabo en la etapa

Productos de desarrollo de software: se describen todos los productos de desarrollo del software que se obtienen en cada una de las etapas

3.1.1 Análisis de requisitos:

El propósito principal de esta etapa es conseguir una comprensión más precisa de los requisitos y una descripción de los mismos que sea fácil de mantener y que nos ayude a estructurar el sistema completo, incluyendo la arquitectura.

El análisis de requerimientos facilita al ingeniero de sistemas especificar la función y comportamiento de los programas, indicar la interfaz con otros elementos del sistema y establecer las ligaduras de diseño que debe cumplir el programa.

También permite al ingeniero refinar la asignación de software y representar el dominio de la información que será tratada por el programa. Así como también brinda al diseñador la representación de la información y las funciones que pueden ser traducidas en datos, arquitectura y diseño procedimental.

Finalmente, la especificación de requerimientos suministra al técnico y al cliente, los medios para valorar la calidad de los programas, una vez que se haya construido.

En la medida que logremos una clara comprensión de lo anterior obtendremos una arquitectura estable y sólida que nos facilite una comprensión en profundidad de los requisitos. Además, podemos destacar los siguientes objetivos:

- Describir un modelo del sistema utilizando el lenguaje de los desarrolladores.
- Utilizar un lenguaje más formal para refinar detalles relativos a los requisitos del sistema.
- Razonar más sobre los aspectos internos del sistema.
- Estructurar los requisitos de un modo que facilite su comprensión, desarrollo, modificación, y en general, su mantenimiento.

¿Cómo logramos estos objetivos?

Para conseguir estos objetivos el flujo de trabajo de la etapa de Análisis de requisitos consta de las siguientes etapas:

- Definir la arquitectura candidata.
- Análisis de los Casos de uso.
- Refinar la arquitectura.
- Análisis de la Realización de los Casos de uso.
- Evaluación.

¿Quiénes son los participantes responsables de realizar las actividades y los productos de desarrollo de software?

- Arquitecto software

- Analista del sistema

¿Cuáles son los pasos a seguir para lograr una captura de requisitos eficiente?

Esta es la etapa inicial, es aquí donde se concibe una idea de que es lo que realmente se quiere lograr con el software, es decir cuáles son las condiciones o capacidades que el sistema debe cumplir , es por ello la importancia de este paso y la mutua comprensión entre las partes involucradas tanto usuario como programadores.-

Primer paso: captura de requisitos

Comenzaremos con una fase de especificación de requisitos. En ella tratamos de conseguir un juego de requisitos de usuario razonablemente completo y consistente, así como una especificación sintetizada de éstos y con un nivel superior de formalización. Vamos a tratar de organizar los requerimientos de usuario dentro de tres conceptos: Descripciones, Lista de requisitos y Escenarios

a) Descripciones

Con mucha frecuencia, los usuarios proporcionan descripciones textuales del problema a resolver. En esas descripciones suelen estar mezclados diferentes aspectos, como descripciones de los elementos del problema, pero al mismo tiempo puede haber muchos requisitos entremezclados con ellas.

b) Lista de requisitos:

Prepararemos una lista de requisitos. Estos pueden haber sido expresados por los usuarios directamente, o quizás haya que extraerlos de los textos y descripciones. En cualquier caso, debemos procurar que todos los requisitos estén recogidos en esa lista.

c) Escenarios:

Finalmente, resulta muy útil conseguir descripciones que nos permitan ejemplificar el funcionamiento del sistema: es lo que denominamos escenarios. Un escenario es una secuencia de interacciones concretas entre los elementos externos al sistema (los "actores" de Jacobson) y el propio sistema que estamos analizando. Estos escenarios nos facilitarán la comprensión del sistema al ser a modo de ejemplos concretos. Por supuesto, no podemos -ni pretendemos- describir todos los escenarios posibles, pero sí tener una pequeña colección que nos permita caracterizar el sistema. Estos escenarios, frente a los requisitos aislados y puntuales, permiten comprender mejor lo que se espera del sistema. Por decirlo de alguna forma, le aportan un "contexto".

Paso 2 Especificaciones

La obtención de especificaciones a partir del cliente (u otros actores intervinientes) es un proceso humano muy interactivo e iterativo; normalmente a medida que se captura la información, se la analiza y realimenta con el cliente, refinándola, puliéndola y corrigiendo si es necesario; EL analista siempre debe llegar a conocer la temática y el problema a resolver, dominarlo, hasta cierto punto, hasta el ámbito que el futuro sistema a desarrollar lo abarque. Por ello el analista debe tener alta capacidad para comprender problemas de muy diversas áreas o disciplinas de trabajo (que no son específicamente suyas); así por ejemplo, si el sistema a desarrollar será para gestionar información de una aseguradora y sus sucursales remotas, el analista se debe compenetrar en cómo ella trabaja y maneja su información, desde niveles muy bajos e incluso llegando hasta los gerenciales.

Dada la gran diversidad de campos a cubrir, los analistas suelen ser asistidos por especialistas, es decir gente que conoce profundamente el área para la cual se desarrollará el software; evidentemente una única persona (el analista) no puede abarcar tan vasta cantidad de áreas del conocimiento. En empresas grandes de

desarrollo de productos software, es común tener analistas especializados en ciertas áreas de trabajo.

Al contrario de los analistas, los clientes no tiene por qué saber nada de software, ni de diseños, ni otras cosas relacionadas; sólo se debe limitar a aportar objetivos, datos e información (de mano propia o de sus registros, equipos, empleados, etc.) al analista, y guiado por él, para que, en primera instancia, defina el "**Universo de Discurso**", y con posterior trabajo logre confeccionar el Adecuado documento

ERS (Especificación de requerimientos de Software).

Las técnicas de análisis pueden conducir a una especificación en papel que contenga las descripciones graficas y el lenguaje natural de los requerimientos del software. La construcción de prototipos conduce a una especificación ejecutable, esto es, el prototipo sirve como una representación de los requerimientos. Los lenguajes de especificación formal conducen a representaciones formales de los requerimientos que pueden ser verificados o analizados.

3.1.2 Principios de Especificación

La especificación, independientemente del modo en que se realice, puede ser vista como un proceso de representación. Los requerimientos se representan de forma que conduzcan finalmente a una correcta implementación del software. Baltzer y Goldman proponen ocho principios para una buena especificación:

PRINCIPIO #1. Separar funcionalidad de implementación.
Primero, por definición, una especificación es una descripción de lo que se desea, hacer en vez de cómo se realiza (implementa). Las especificaciones pueden adoptar dos formas muy diferentes. La primera forma es la de funciones matemáticas: dado algún conjunto de entrada, producir un conjunto particular de

salida. La forma general de tales especificaciones es encontrar [un/el/todos] resultado tal que P (entrada), donde P representa un predicado arbitrario. En tales especificaciones, el resultado a ser obtenido ha sido expresado enteramente en una forma sobre el que (en vez de cómo). En parte, esto es debido a que el resultado es una función matemática de la entrada (la operación tiene puntos de comienzo y parada bien definidos) y no está afectado por el entorno que le rodea.

PRINCIPIO #2. Se necesita un lenguaje de especificación de sistemas orientado al proceso.

Considerar una situación en la que el entorno sea dinámico y sus cambios afecten al comportamiento de alguna entidad que interactúe con dicho entorno. Su comportamiento no puede ser expresado como una función matemática de su entrada. En vez de ello, debe emplearse una descripción orientada al proceso, en la cual la especificación del que se consigue mediante la especificación de un modelo del comportamiento deseado en términos de respuestas funcionales, a distintos estímulos del entorno.

PRINCIPIO #3. Una especificación debe abarcar el sistema del cual el software es una componente.

Un sistema está compuesto de componentes que interactúan. Solo dentro del contexto del sistema completo y de la interacción entre sus partes puede ser definido el comportamiento de una componente específica. En general, un sistema puede ser modelado como una colección de objetos pasivos y activos. Estos objetos están interrelacionados y dichas relaciones entre los objetos cambian con el tiempo. Estas relaciones dinámicas suministran los estímulos a los cuales los objetos activos, llamados agentes, responden. Las respuestas pueden causar posteriormente cambios y, por tanto, estímulos adicionales a los cuales los agentes deben responder.

PRINCIPIO #4. Una especificación debe abarcar el entorno en el que el sistema opera. Similarmente, el entorno en el que opera el sistema y con el que interactúa debe ser especificado.

Afortunadamente, esto tan solo necesita reconocer que el propio entorno es un sistema compuesto de objetos que interactúan, pasivos y activos, de los cuales el sistema especificado es una agente, Los otros agentes, los cuales son por definición inalterables debido a que son parte del entorno, limitan el ámbito del diseño subsecuente y de la implementación. De hecho, la única diferencia entre el sistema y su entorno es que el esfuerzo de diseño e implementación subsecuente opera exclusivamente sobre la especificación del sistema. La especificación del entorno facilita que se especifique la interfaz del sistema de la misma forma que el propio sistema, en vez de introducir otro formalismo.

PRINCIPIO #5. Una especificación de sistema debe ser un modelo cognitivo.

La especificación de un sistema debe ser un modelo cognitivo, en vez de un modelo de diseño o implementación. Debe describir un sistema tal como es percibido por su comunidad de usuario. Los objetivos que manipula deben corresponderse con objetos reales de dicho dominio; los agentes deben modelar los individuos, organizaciones y equipo de ese dominio; y las acciones que ejecutan deben modelar lo que realmente ocurre en el dominio. Además, debe ser posible incorporar en la especificación las reglas o leyes que gobiernan los objetos del dominio. Algunas de estas leyes proscriben ciertos estados del sistema (tal como "dos objetos no pueden estar en el mismo lugar al mismo tiempo"), y por tanto limitan el comportamiento de los agentes o indican la necesidad de una posterior elaboración para prevenir que surjan estos estados.

PRINCIPIO #6. Una especificación debe ser operacional.

La especificación debe ser completa y lo bastante formal para que pueda usarse para determinar si una implementación propuesta satisface la especificación de pruebas elegidas arbitrariamente. Esto es, dado el resultado de una

implementación sobre algún conjunto arbitrario de datos elegibles, debe ser posible usar la especificación para validar estos resultados.

Esto implica que la especificación, aunque no sea una especificación completa del como, pueda actuar como un generador de posibles comportamientos, entre los que debe estar la implementación propuesta. Por tanto, en un sentido extenso, la especificación debe ser operacional.

PRINCIPIO #7. La especificación del sistema debe ser tolerante con la in completitud y aumentable.
Ninguna especificación puede ser totalmente completa. El entorno en el que existe es demasiado complejo para ello. Una especificación es siempre un modelo, una abstracción, de alguna situación real (o imaginada). Por tanto, será incompleta. Además, al ser formulada existirán muchos niveles de detalle. Lo requerido anteriormente no necesita ser completo. Las herramientas de análisis empleadas para ayudar a los especificadores y para probar las especificaciones, deben ser capaces de tratar con estas limitaciones. Naturalmente esto debilita el análisis, el cual puede ser ejecutado ampliando el rango de comportamiento aceptables, los cuales satisfacen la especificación, pero tal degradación debe reflejar los restantes niveles de incertidumbre.

PRINCIPIO #8. Una especificación debe ser localizada y débilmente acoplada.
Los principios anteriores tratan con la especificación como una entidad estática. Esta surge de la dinámica de la especificación. Debe ser reconocido que aunque el principal propósito de una especificación sea servir como base para el diseño e implementación de algún sistema, no es un objeto estático pre compuesto, sino un objeto dinámico que sufre considerables modificaciones. Tales modificaciones se presentan en tres actividades principales: formulación, cuando se está creando una especificación inicial; desarrollo, cuando la especificación se está elaborando durante el proceso iterativo de diseño e implementación; y mantenimiento, cuando

la especificación se cambia para reflejar un entorno modificado y/o requerimientos funcionales adicionales.

Durante la fase de especificación de requerimientos del software, se debe definir de manera clara y precisa todas y cada una de las funcionalidades que el sistema en cuestión debe cubrir.

Considerar que el software a corregir dentro de las especificaciones iníciales no debe de perder la misión encomendada por el usuario, a menos que esté justifique un cambio radical y sea valido alterar las especificaciones originales.

3.1.3 Diseño y arquitectura

Este proceso conlleva la realización de un conjunto complejo de actividades, en las que deben intervenir la mayoría de las áreas funcionales del diseño.

Generalmente este proceso de desarrollo se suele dividir en cinco fases o etapas:

1.- Identificación de oportunidades.

2.- Evaluación y selección.

3.- Desarrollo e ingeniería del producto y del proceso.

4.- Pruebas y evaluación.

5.- Comienzo de la producción.

En esta etapa se modela el sistema y definimos su estructura (incluida la arquitectura) para que soporte todos los requisitos, incluyendo los requisitos no funcionales y otras restricciones

En concreto; los propósitos de la etapa de diseño son:

- Adquirir una comprensión en profundidad de los aspectos relacionados con los requisitos no funcionales y restricciones relacionadas con los lenguajes de programación, componentes reutilizables, sistemas operativos, tecnologías de distribución y concurrencia
- Tecnologías de interfaz de usuario, tecnologías de gestión de transacciones, etc.
- Crear una entrada apropiada y un punto de partida para actividades de implementación subsiguientes capturando los requisitos o subsistemas individuales, interfaces y clases.
- Ser capaces de descomponer los trabajos de implementación en partes más manejables que puedan ser llevadas a cabo por diferentes equipos de desarrollo, teniendo en cuenta la posible concurrencia.
- Capturar las interfaces entre los subsistemas antes en el ciclo de vida del software. Esto ayuda cuando reflexionamos sobre la arquitectura y cuando utilizamos interfaces como elementos de sincronización entre diferentes equipos de desarrollo.
- Ser capaces de visualizar y reflexionar sobre el diseño utilizando una notación común.
- Crear una abstracción sin costuras de la implementación del sistema, en el sentido de que la implementación es un refinamiento directo del diseño que rellena lo existente sin cambiar la estructura. Esto permite la utilización de tecnologías como la generación de código y la ingeniería de ida y vuelta entre el diseño y la implementación.
- Definir el modelo de interfaces de usuario.

Para conseguir estos objetivos el flujo de trabajo de la etapa de Diseño consta de las siguientes etapas:

- Diseño arquitectónico.
- Diseño de la Realización de los Casos de Uso.
- Diseño de Casos de Uso.
- Diseño de Subsistemas y Servicios.
- Diseño de Elementos de Prueba.
- Refinar la Arquitectura del Software.
- Evaluación.

Los productos de desarrollo del software fundamentales que se desarrollan en la etapa de Diseño son:

- Modelo del Diseño, incluyendo el Diagrama de Clases, informes y la Realización de los Casos de Uso.
- Modelo de Interfaz de Usuario, incluyendo los Bocetos del Interfaz de Usuario y el Mapa de Navegación Storyboard.
- Informe del Modelo del Interfaz de Usuario.
- Modelo de Datos, Informe del Modelo de Datos.
- Arquitectura del Software.
- Informe de Evaluación.
- Casos de Prueba.

Los participantes responsables de las realizar las actividades y los productos de desarrollo del software son:

- Arquitecto software.

Dentro del ciclo de vida del software la etapa de Diseño es el centro de atención al final de la fase de Elaboración y el comienzo de las iteraciones de construcción. Esto contribuye a una arquitectura estable y sólida y a crear un plano del modelo de implementación. Más tarde, durante la fase de Construcción cuando la arquitectura es estable y los requisitos están bien entendidos, el centro de atención se desplaza a la etapa de Implementación.

3.1.4 Programación:

Reducir un diseño a código puede ser la parte más obvia del trabajo de ingeniería de software, pero no necesariamente es la que demanda mayor trabajo y ni la más complicada. La complejidad y la duración de esta etapa está íntimamente relacionada al o a los lenguajes de programación utilizados, así como al diseño previamente realizado

3.1.5 Prueba:

La prueba del software es un elemento crítico para la garantía de la calidad del software. El objetivo de la etapa de pruebas es garantizar la calidad del producto desarrollado. Además, esta etapa implica:

- Verificar la interacción de componentes.
- Verificar la integración adecuada de los componentes.
- Verificar que todos los requisitos se han implementado correctamente.
- Identificar y asegurar que los defectos encontrados se han corregido antes de entregar el software al cliente.
- Diseñar pruebas que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo.

La prueba no es una actividad sencilla, no es una etapa del proyecto en la cual se asegura la calidad, sino que la prueba debe ocurrir durante todo el ciclo de vida: podemos probar la funcionalidad de los primeros prototipos; probar la estabilidad, cobertura y rendimiento de la arquitectura; probar el producto final, etc. Lo que conduce al principal beneficio de la prueba: proporcionar feedback mientras hay todavía tiempo y recursos para hacer algo.

La prueba es un proceso que se enfoca sobre la lógica interna del software y las funciones externas. La prueba es un proceso de ejecución de un programa con la intención de descubrir un error; que probablemente no fue previsto en las fases iníciales del desarrollo del software.

Cualquier proceso de ingeniería puede ser probado de una de estas dos formas:

1) Se pueden llevar a cabo pruebas que demuestren que cada función es completamente operativa.
2) Se pueden desarrollar pruebas que aseguren que la operación interna se ajusta a las especificaciones y que todos los componentes internos se han comprobado de forma adecuada.

La primera aproximación se denomina prueba de la caja negra y la segunda prueba de la caja blanca.

Prueba de caja blanca:

Permiten examinar la estructura interna del programa. Se diseñan casos de prueba para examinar la lógica del programa. Es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivar casos de prueba que garanticen que:

- ✓ Se ejercitan todos los caminos independientes de cada módulo.
- ✓ Se ejercitan todas las decisiones lógicas.
- ✓ Se ejecutan todos los bucles.

- ✓ Se ejecutan las estructuras de datos internas.

Prueba de caja negra:

Las pruebas se llevan a cabo sobre la interfaz del software, y es completamente indiferente el comportamiento interno y la estructura del programa.

Los casos de prueba de la caja negra pretende demostrar que:

- ✓ Las funciones del software son operativas.
- ✓ La entrada se acepta de forma adecuada.
- ✓ Se produce una salida correcta, y
- ✓ La integridad de la información externa se mantiene.

Se derivan conjuntos de condiciones de entrada que ejerciten completamente todos los requerimientos funcionales del programa.

La prueba de la caja negra intenta encontrar errores de las siguientes categorías:

- ✓ Funciones incorrectas o ausentes.
- ✓ Errores de interfaz.
- ✓ Errores en estructuras de datos o en accesos a bases de datos externas.
- ✓ Errores de rendimiento.
- ✓ Errores de inicialización y de terminación.

Los casos de prueba deben satisfacer los siguientes criterios:

- Reducir, en un coeficiente que es mayor que uno, el número de casos de prueba adicionales.
- Que digan algo sobre la presencia o ausencia de clases de errores.

TIPOS DE PRUEBAS

Pruebas de unidad:

La prueba de unidad se centra en el módulo. Usando la descripción del diseño detallado como guía, se prueban los caminos de control importantes con el fin de descubrir errores dentro del ámbito del módulo. La prueba de unidad hace uso intensivo de las técnicas de prueba de caja blanca.

Prueba de integración:

El objetivo es coger los módulos probados en la prueba de unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño.

Hay dos formas de integración:

- Integración no incremental: Se combinan todos los módulos por anticipado y se prueba todo el programa en conjunto.
- Integración incremental: El programa se construye y se prueba en pequeños segmentos.

En la prueba de integración el foco de atención es el diseño y la construcción de la arquitectura del software.

Las técnicas que más prevalecen son las de diseño de casos de prueba de caja negra, aunque se pueden llevar a cabo unas pocas pruebas de caja blanca.

Prueba del sistema:

Verifica que cada elemento encaja de forma adecuada y que se alcanza la funcionalidad y el rendimiento del sistema total. La prueba del sistema está constituida por una serie de pruebas diferentes cuyo propósito primordial es ejercitar profundamente el sistema basado en computadora. Algunas de estas pruebas son:

- Prueba de validación: Proporciona una seguridad final de que el software satisface todos los requerimientos funcionales y de rendimiento. Además, valida los requerimientos establecidos comparándolos con el sistema que ha

sido construido. Durante la validación se usan exclusivamente técnicas de prueba de caja negra.

- Prueba de recuperación: Fuerza un fallo del software y verifica que la recuperación se lleva a cabo apropiadamente.
- Prueba de seguridad: Verificar los mecanismos de protección.
- Prueba de resistencia: Enfrenta a los programas a situaciones anormales.
- Prueba de rendimiento: Prueba el rendimiento del software en tiempo de ejecución.
- Prueba de instalación: Se centra en asegurar que el sistema software desarrollado se puede instalar en diferentes configuraciones hardware y software y bajo condiciones excepciones, por ejemplo con espacio de disco insuficiente o continuas interrupciones.

Pruebas de regresión:

Las pruebas de regresión son una estrategia de prueba en la cual las pruebas que se han ejecutado anteriormente se vuelven a realizar en la nueva versión modificada, para asegurar la calidad después de añadir la nueva funcionalidad. El propósito de estas pruebas es asegurar que:

- Los defectos identificados en la ejecución anterior de la prueba se ha corregido.
- Los cambios realizados no han introducido nuevos defectos o reintroducido defectos anteriores.

La prueba de regresión puede implicar la re-ejecución de cualquier tipo de prueba. Normalmente, las pruebas de regresión se llevan a cabo durante cada iteración, ejecutando otra vez las pruebas de la iteración anterior.

Estrategias de pruebas del software:

Una estrategia de prueba del software integra las técnicas de diseño de casos de prueba en una serie de pasos bien planificados que llevan a la construcción correcta del software.

Las características generales son:

- La prueba comienza en el nivel de módulo y trabaja “hacia afuera”.
- En diferentes puntos son adecuadas a la vez distintas técnicas de prueba.
- La prueba la realiza la persona que desarrolla el software y (para grandes proyectos) un grupo de pruebas independiente.
- La prueba y la depuración son actividades diferentes.

Una estrategia de prueba para el software debe constar de pruebas de bajo nivel, así como de pruebas de alto nivel.

Más concretamente, los objetivos de la estrategia de prueba son:

- Planificar las pruebas necesarias en cada iteración, incluyendo las pruebas de unidad, integración y las pruebas de sistema. Las pruebas de unidad y de integración son necesarias dentro de la iteración, mientras que las pruebas de sistema son necesarias sólo al final de la iteración.
- Diseñar e implementar las pruebas creando los casos de prueba que especifican qué probar, cómo realizar las pruebas y creando, si es posible, componentes de prueba ejecutables para automatizar las pruebas.
- Realizar diferentes pruebas y manejar los resultados de cada prueba sistemáticamente. Los productos de desarrollo de software en los que se detectan defectos son probadas de nuevo y posiblemente devueltas a otra etapa, como diseño o implementación, de forma que los defectos puedan ser arreglados.

Para conseguir estos objetivos el flujo de trabajo de la etapa de Pruebas consta de las siguientes etapas:

- Planificación de las pruebas.
- Diseño de las pruebas.
- Implementación de las pruebas.
- Ejecución de las pruebas.
- Evaluación de las pruebas.

Los productos de desarrollo del software fundamentales que se desarrollan en la etapa de Pruebas son:

- Plan de Pruebas.
- Casos de Prueba.
- Informe de evaluación de Pruebas.
- Modelo de Pruebas, que incluye Clases de Prueba, Entorno de Configuración de Pruebas, Componentes de Prueba y los Datos de prueba.

Los participantes responsables de las realizar las actividades y los productos de desarrollo del software son:

Diseñador de pruebas: Es responsable de la planificación, diseño, implementación y evaluación de las pruebas. Esto conlleva generar el plan de pruebas y modelo de pruebas, implementar los casos de prueba y evaluar los resultados de las pruebas. Los diseñadores de prueba realmente no llevan a cabo las pruebas, sino que se dedican a la preparación y evaluación de las mismas.

Probador (Tester): Es responsable de desarrollar las pruebas de unidad, integración y sistema, lo que incluye ejecutar las pruebas, evaluar su ejecución, recuperar los errores y garantizar los resultados de las pruebas.

Durante la fase de Inicio puede hacerse parte de la planificación inicial de las pruebas cuando se define el ámbito del sistema. Sin embargo, las pruebas se llevan a cabo sobre todo cuando un producto de desarrollo de software es sometido a pruebas de integración y de sistema. Esto quiere decir que la realización de pruebas se centra en las fases de Elaboración, cuando se prueba la

línea base ejecutable de la arquitectura, y de construcción, cuando el grueso del sistema está implementado. Durante la fase de Transición el centro se desplaza hacia la corrección de defectos durante los primeros usos y a las pruebas de regresión Debido a la naturaleza iterativa del esfuerzo de desarrollo, algunos de los casos de prueba que especifican cómo probar los primeros productos de desarrollo software pueden ser utilizadas también como casos de prueba de regresión que especifican cómo llevar a cabo las pruebas de regresión sobre los productos de desarrollo software siguientes. El número de pruebas de regresión necesarias crece por tanto de forma estable a lo largo de las iteraciones, lo que significa que las últimas iteraciones requerirán un gran esfuerzo en pruebas de regresión. Es natural, por tanto, mantener el modelo de pruebas a lo largo del ciclo de vida del software completo, aunque el modelo de pruebas cambia constantemente debido a:

- La eliminación de casos de prueba obsoletos.
- El refinamiento de algunos casos de prueba en casos de prueba de regresión.
- La creación de nuevos casos de prueba para cada nuevo producto de desarrollo de software

3.1.6 **Documentación**:

Todo lo concerniente a la documentación del propio desarrollo del software y de la gestión del proyecto, pasando por modelaciones (UML),modelado de negocio, RUP, diagramas, pruebas, manuales de usuario, manuales técnicos, etc.; todo con el propósito de eventuales correcciones, utilización, mantenimiento futuro y ampliaciones al sistema.

3.1.7 Mantenimiento:

Mantener y mejorar el software para enfrentar errores descubiertos y nuevos requisitos. Esto puede llevar más tiempo incluso que el desarrollo inicial del software. Alrededor de 2/3 de toda la ingeniería de software tiene que ver con dar mantenimiento. Una pequeña parte de este trabajo consiste en arreglar errores *bugs* [5]La mayor parte consiste en extender el sistema para hacer nuevas cosas. De manera similar, alrededor de 2/3 de toda la ingeniería civil, arquitectura y trabajo de construcción es dar mantenimiento.

La fase de mantenimiento de software es una parte explícita del modelo en cascada del proceso de desarrollo de software el cual fue desarrollado durante el movimiento de programación estructurada en computadores. El otro gran modelo, el Desarrollo en espiral desarrollado durante el movimiento de ingeniería de software orientada a objeto no hace una mención explícita de la fase de mantenimiento.

En un ambiente formal de desarrollo de software, la organización o equipo de desarrollo tendrán algún mecanismo para documentar y rastrear defectos y deficiencias. El Software tan igual como la mayoría de otros productos, es típicamente lanzado con un conjunto conocido de defectos y deficiencias. El software es lanzado con esos defectos conocidos porque la organización de desarrollo en las utilidades y el valor del software en un determinado nivel de calidad compensan el impacto de los defectos y deficiencias conocidas.

Las deficiencias conocidas son normalmente documentadas en una carta de consideraciones operacionales o notas de lanzamiento (release notes) es así que los usuarios del software serán capaces de trabajar evitando las deficiencias conocidas y conocerán cuando el uso del software sería inadecuado para tareas específicas.

[5] Bugs: Defecto de software, es el resultado de un fallo o deficiencia durante el proceso de creación de programas de software

Tipos de mantenimiento

A continuación se señalan los tipos de mantenimientos existentes, definidos tal y como se especifican para la metodología de métrica:

- **Perfectivo:** son las acciones llevadas a cabo para mejorar la calidad interna de los sistemas en cualquiera de sus aspectos: reestructuración del código, definición más clara del sistema y optimización del rendimiento y eficiencia.
- **Evolutivo:** son las incorporaciones, modificaciones y eliminaciones necesarias en un producto software para cubrir la expansión o cambio en las necesidades del usuario.
- **Adaptativo:** son las modificaciones que afectan a los entornos en los que el sistema opera, por ejemplo, cambios de configuración del hardware, software de base, gestores de base de datos, comunicaciones, etc.
- **Correctivo:** son aquellos cambios precisos para corregir errores del producto software.

3.2 Modelos del ciclo de vida:

Todo proyecto de ingeniería tiene unos fines ligados a la obtención de un producto, proceso o servicio que es necesario generar a través de diversas actividades. Algunas de estas actividades pueden agruparse en fases porque globalmente contribuyen a obtener un producto intermedio, necesario para continuar hacia el producto final y facilitar la gestión del proyecto. Al conjunto de las fases empleadas se le denomina “ciclo de vida”.

Para facilitar una metodología común entre el cliente y la compañía de software, los modelos de ciclo de vida se han actualizado para reflejar las etapas de desarrollo involucradas y la documentación requerida, de manera que cada etapa se valide antes de continuar con la siguiente etapa. Al final de cada etapa se arreglan las revisiones de manera que los involucrados evalúen si se cumple con los objetivos de esa etapa.

3.2.1 Modelo en Cascada

En cascada (Waterflow): Se denomina modelo en cascada porque su característica principal es que no se comienza con un paso hasta que no se ha terminado el anterior. Comenzó a diseñarse en 1966 y se terminó alrededor de 1970. El principal problema de esta aproximación es el que no podemos esperar el que las especificaciones iníciales sean correctas y completas y que el usuario puede cambiar de opinión sobre una u otra característica. Además los resultados no se pueden ver hasta muy avanzado el proyecto por lo que cualquier cambio debido a un error puede suponer un gran retraso además de un alto coste de desarrollo.

Como es evidente esto es solo un modelo teórico, si el usuario cambia de opinión en algún aspecto tendremos que volver hacia atrás en el ciclo de vida.

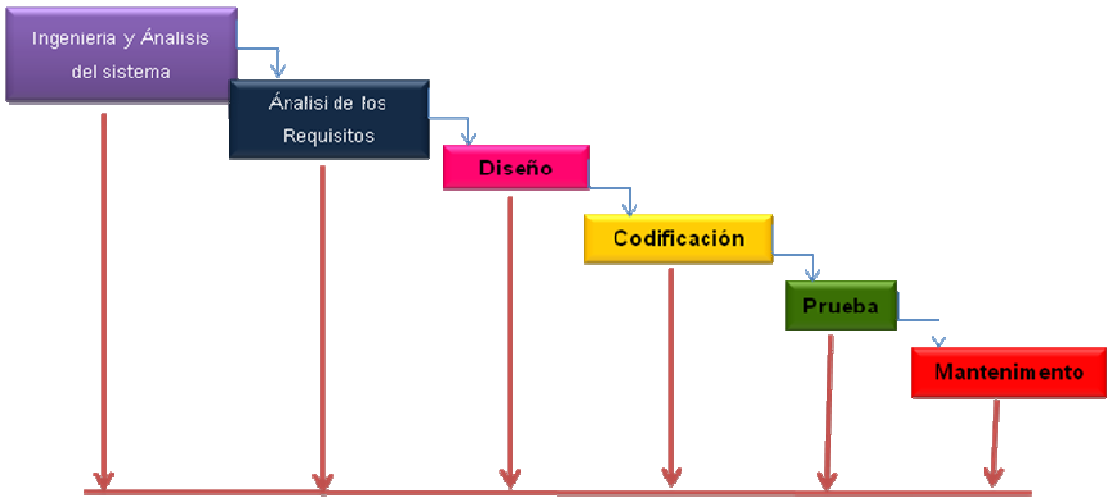


Figura 5 Modelo de ciclo de vida en cascada

Ingeniería y Análisis del Sistema: Debido a que el software es siempre parte de un sistema mayor el trabajo comienza estableciendo los requisitos de todos los

elementos del sistema y luego asignando algún subconjunto de estos requisitos al software.

Análisis de los requisitos del software: el proceso de recopilación de los requisitos se centra e intensifica especialmente en el software. El ingeniero de software (Analistas) debe comprender el ámbito de la información del software, así como la función, el rendimiento y las interfaces requeridas.

Diseño: el diseño del software se enfoca en cuatro atributos distintos del programa: la estructura de los datos, la arquitectura del software, el detalle procedimental y la caracterización de la interfaz. El proceso de diseño traduce los requisitos en una representación del software con la calidad requerida antes de que comience la codificación.

Codificación: el diseño debe traducirse en una forma legible para la maquina. El paso de codificación realiza esta tarea. Si el diseño se realiza de una manera detallada la codificación puede realizarse mecánicamente.

Prueba: una vez que se ha generado el código comienza la prueba del programa. La prueba se centra en la lógica interna del software, y en las funciones externas, realizando pruebas que aseguren que la entrada definida produce los resultados que realmente se requieren.

Mantenimiento: el software sufrirá cambios después de que se entrega al cliente. Los cambios ocurrirán debido a se han encontrado errores, a que el software deba adaptarse a cambios del entorno externo (sistema operativo o dispositivos periféricos), o debido a que el cliente requiera ampliaciones funcionales o del rendimiento.

Desventajas:

- Los proyectos reales raramente siguen el flujo secuencial que propone el modelo, siempre hay iteraciones y se crean problemas en la aplicación del paradigma.

- Normalmente, es difícil para el cliente establecer explícitamente al principio todos los requisitos. El ciclo de vida clásico lo requiere y tiene dificultades en acomodar posibles incertidumbres que pueden existir al comienzo de muchos productos.

- El cliente debe tener paciencia. Hasta llegar a las etapas finales del proyecto, no estará disponible una versión operativa del programa. Un error importante no detectado hasta que el programa este funcionando puede ser desastroso.

3.2.2 Modelo de desarrollo Evolutivo

Como el modelo de desarrollo incremental, el modelo de desarrollo evolutivo (algunas veces denominado como prototipado evolutivo) construye una serie de grandes versiones sucesivas de un producto. Sin embargo, mientras que la aproximación incremental presupone que el conjunto completo de requerimientos es conocido al comenzar, el modelo evolutivo asume que los requerimientos no son completamente conocidos al inicio del proyecto.

En el modelo evolutivo, los requerimientos son cuidadosamente examinados, y sólo esos que son bien comprendidos son seleccionados para el primer incremento. Los desarrolladores construyen una implementación parcial del sistema que recibe sólo estos requerimientos.

El sistema es entonces desarrollado, los usuarios lo usan, y proveen retroalimentación a los desarrolladores. Basada en esta retroalimentación, la especificación de requerimientos es actualizada, y una segunda versión del producto es desarrollada y desplegada. El proceso se repite indefinidamente.

Note que el desarrollo evolutivo es 100% compatible con el modelo cascada. El desarrollo evolutivo no demanda una forma específica de observar el desarrollo de algún incremento. Así, el modelo cascada puede ser usado para administrar cada esfuerzo de desarrollo. Obviamente, el desarrollo incremental y evolutivo puede ser combinado también.

Todo lo que uno tiene que hacer es construir un subconjunto de requerimientos conocidos (incremental), y comprender al principio que muchos nuevos requerimientos es probable que aparezcan cuando el sistema sea desplegado o desarrollado.

El desarrollo de software en forma evolutiva requiere un especial cuidado en la manipulación de documentos, programas, datos de test, etc. desarrollados para distintas versiones del software. Cada paso debe ser registrado, la documentación debe ser recuperada con facilidad, los cambios deben ser efectuados de una manera controlada.

3.2.3 Modelo de Prototipado de Requerimientos.-

El prototipado de requerimientos es la creación de una implementación parcial de un sistema, para el propósito explícito de aprender sobre los requerimientos del sistema. Un prototipo es construido de una manera rápida tal como sea posible. Esto es dado a los usuarios, clientes o representantes de ellos, posibilitando que ellos experimenten con el prototipo. Estos individuos luego proveen la retroalimentación sobre lo que a ellos les gustó y no les gustó acerca del prototipo proporcionado, quienes capturan en la documentación actual de la especificación de requerimientos la información entregada por los usuarios para el desarrollo del sistema real. El prototipado puede ser usado como parte de la fase de requerimientos (determinar requerimientos) o justo antes de la fase de requerimientos (como predecesor de requerimientos). En otro caso, el prototipado

Puede servir su papel inmediatamente antes de algún o todo el desarrollo incremental en modelos incremental o evolutivo.

El Prototipado ha sido usado frecuentemente en los 90, porque la especificación de requerimientos para sistemas complejos tiende a ser relativamente dificultoso de cursar. Muchos usuarios y clientes encuentran que es mucho más fácil proveer retroalimentación convenientemente basada en la manipulación, leer una especificación de requerimientos potencialmente ambigua y extensa.

Diferente del modelo evolutivo donde los requerimientos mejor entendidos están incorporados, un prototipo generalmente se construye con los requerimientos entendidos más pobremente.

En caso que ustedes construyan requerimientos bien entendidos, el cliente podría responder con "sí, así es", y nada podría ser aprendido de la experiencia.

3.2.4 Modelo de desarrollo basado en reutilización**:**

Un componente es una pieza de código pre-elaborado que encapsula alguna funcionalidad expuesta a través de interfaces estándar. El paradigma de ensamblar componentes y escribir código para hacer que estos componentes funcionen se conoce como Desarrollo de Software Basado en Componentes.

Desarrollo basado en componentes

El modelo de desarrollo basado en componentes incorpora muchas de las características del modelo espiral. Es evolutivo por naturaleza y exige un enfoque interactivo para la creación del software. Sin embargo, el modelo de desarrollo basado en componentes configura aplicaciones desde componentes preparados de software (clases).

Naturaleza y exige un enfoque interactivo para la creación del software. Sin embargo, el modelo de desarrollo basado en componentes configura aplicaciones desde componentes preparados de software (clases).

El modelo de desarrollo basado en componentes conduce a la reutilización del software, y la reutilización proporciona beneficios a los ingenieros de software. Según estudios de reutilización, QSM Associates, Inc. Informa que el ensamblaje de componentes lleva a una reducción del 70 % del ciclo de desarrollo un 84% del coste del proyecto y un índice de productividad del 26.2. No hay duda que el ensamblaje de componentes proporciona ventajas significativas para los ingenieros del software.

El proceso unificado de desarrollo de software representa un número de modelos de desarrollo basado en componentes que han sido propuestos en la industria. El lenguaje de modelado unificado define los componentes. Utilizando una combinación del desarrollo incremental e interactivo, el proceso unificado define la función del sistema aplicando un enfoque basado en escenarios.

El desarrollo de software basado en componentes se ha convertido actualmente en uno de los mecanismos más efectivos para la construcción de grandes sistemas y aplicaciones de software.

Una vez que la mayor parte de los aspectos funcionales de esta disciplina comienzan a estar bien definidos, la atención de la comunidad científica comienza a centrarse en los aspectos extra-funcionales y de calidad, como un paso hacia una verdadera ingeniería. En este artículo se discuten precisamente los aspectos de calidad relativa a los componentes de software y a las aplicaciones que con ellos se construyen, con especial énfasis en los estándares internacionales que los definen y regulan, y en los problemas que se plantean en este tipo de entornos.

Beneficios del Desarrollo de Software Basado en Componentes

El uso de este paradigma posee algunas ventajas:

1. Reutilización del software. Nos lleva a alcanzar un mayor aprovechamiento de los códigos generados

2. Simplifica las pruebas. Permite que las pruebas sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados.

3. Simplifica el mantenimiento del sistema. Cuando existe un débil acoplamiento entre componentes, el desabollador es libre de actualizar y/o agregar componentes según sea necesario, sin afectar otras partes del sistema.

4. Mayor calidad. Dado que un componente puede ser construido y luego mejorado continuamente por un experto u organización, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo.

La Notación de Componentes

Un componentes, es implementado por una o más clases/objetos del sistema.
Es una unidad autónoma que provee una o más interfaces.
Las interfaces presentan un contrato de servicios que el componente ofrece.

Los componentes pueden ser

- Archivos
- Código Fuente + cabeceras
- Librerías compartidas (DLLs)
- Ejecutables
- Paquetes

El Diagrama de Componentes

El diagrama de componentes muestra la relación entre componentes de software, sus dependencias, su comunicación su ubicación y otras condiciones.

Interfaces

Los componentes también pueden exponer las interfaces. Estas son los puntos visibles de entrada o los servicios que un componente está ofreciendo y dejando disponibles a otros componentes de software y clases. Típicamente, un componente está compuesto por numerosas clases y paquetes de clases internos. También se puede crear a partir de una colección de componentes más pequeños.

Los componentes y los Nodos

Un diagrama de despliegue muestra el despliegue físico del sistema en un ambiente de producción (o de prueba). Muestra dónde se ubican los componentes, en qué servidores, máquinas o hardware. Puede representar los enlaces de redes.

Restricciones

Las pre-condiciones especifican lo que debe ser verdadero antes de que un componente pueda realizar alguna función; las post-condiciones indican lo que debe ser verdadero después de que un componente haya realizado algún trabajo

y los invariantes especifican lo que debe permanecer verdadero durante la vida del componente.

Conclusión

Tenemos la fortuna de presenciar el nacimiento de una nueva forma de hacer software, que traerá beneficios inmensos para todos. El desarrollo de software basado en componentes desde siempre fue la idea revolucionaria que nos llevó a pensar que sí era posible el construir software de calidad en corto tiempo y con la misma calidad que la mayoría de las industrias de nuestro tiempo.

Al mirar hacia atrás, vemos los increíbles avances que hemos logrado en la comprensión de la forma correcta de reutilizar el software y el conocimiento existente, y nos asombramos cada vez más al darnos cuenta de que este solo es el inicio. El desarrollo de software basado en componentes se convirtió en el pilar de la Revolución Industrial del Software y se proyecta hoy en día en diversas nuevas formas de hacer software de calidad con los costos más bajos del mercado y en tiempos que antes eran impensables. Empresas como Microsoft entendieron el potencial de esta metodología hace años y hoy nos ofrecen nuevas

Iniciativas y herramientas que buscan llevar al proceso de construcción de software hacia un sitio privilegiado en el que debió colocarse desde un principio.

Análisis del riesgo

Se estudian todos los riesgos potenciales y se seleccionan una o varias alternativas propuestas para reducir o eliminar los riesgos.

En el proceso de planificación se debería identificar y medir la probabilidad de todos los riesgos potenciales y el impacto en la organización si aquella amenaza ocurriera

Ventajas:

El análisis del riesgo se hace de forma explícita y clara. Une los mejores elementos de los restantes modelos. - Reduce riesgos del proyecto - Incorpora objetivos de calidad - Integra el desarrollo con el mantenimiento

Desventajas:

Genera mucho tiempo en el desarrollo del sistema - Modelo costoso –Requiere experiencia en la identificación de riesgos

Inconvenientes

Genera mucho trabajo adicional. Cuando un sistema falla se pierde tiempo y coste dentro de la empresa. Exige una cierta habilidad en los analistas (es bastante difícil).

3.2.5 Modelo de desarrollo en espiral:

Toma las ventajas del modelo de desarrollo en cascada y el de prototipos añadiéndole el concepto de análisis de riesgo.

Se definen cuatro actividades:

Planificación: en la que se recolectan los requisitos iníciales o nuevos requisitos a añadir en esta iteración.

Análisis de riesgo; basándonos en los requisitos decidimos si somos capaces o no de desarrollar el software y se toma la decisión de continuar o no continuar.

Ingeniería, en el que se desarrolla un prototipo basado en los requisitos obtenidos en la fase de planificación.

Evaluación del cliente: el cliente comenta el prototipo. Si está conforme con él se acaba el proceso, si no se añaden los nuevos requisitos en la siguiente iteración.

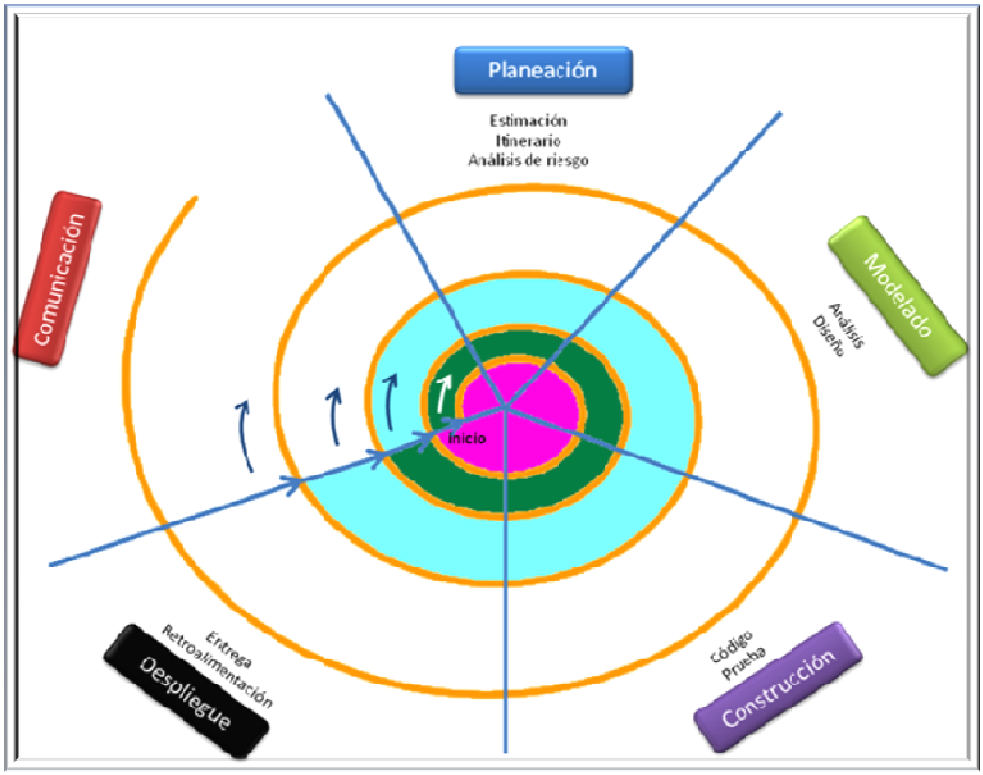


Figura 6: Modelo de ciclo de vida en espiral

El esquema del ciclo de vida para estos casos puede representarse por un bucle en espiral, donde los cuadrantes son, habitualmente, fases de **especificación, diseño, realización y evaluación** (o conceptos y términos análogos).

En cada fase el producto gana "madurez" (aproximación al final deseado) hasta que en una iteración se logre el objetivo deseado y este sea aprobado se termina las iteraciones.

3.2.6 Modelo de desarrollo incremental:

Permite construir el proyecto en etapas incrementales en donde cada etapa agrega funcionalidad

Estas etapas, consisten en **requerimientos, diseño, codificación, pruebas y entrega**. Permite entregar al cliente un producto más rápido en comparación del modelo en cascada.

- Reduce los riegos ya que provee visibilidad sobre el progreso de las nuevas versiones.
- Provee retroalimentación a través de la funcionalidad mostrada.
- Permite atacar los mayores riesgos desde el inicio.
- Se pueden hacer implementaciones parciales si se cuenta con la suficiente funcionalidad.
- Las pruebas y la integración es constante.
- El progreso se puede medir en periodo cortos de tiempo.
- Resulta más sencillo acomodar cambios al acortar el tamaño de los incrementos.
- Se puede planear en base a la funcionalidad que se quiere entregar primero
- Por su versatilidad requiere de una planeación cuidadosa tanto a nivel administrativo como técnico.

A Favor

La solución se va mejorando en forma progresiva a través de las múltiples iteraciones

Incrementa el entendimiento del problema y de la solución por medio de los refinamientos sucesivos

En contra:

- ✓ Requiere de mucha planeación, tanto administrativa como técnica
- ✓ Requiere de metas claras para conocer el estado del proyecto.
- ✓ Es un proceso de desarrollo de software, creado en respuesta a las debilidades del modelo tradicional de cascada.

Para apoyar el desarrollo de proyectos por medio de este modelo se han creado Framework (entornos de trabajo), de los cuales los dos más famosos son el **Rational Unified Process (RUP)** y el **Dynamic Systems Development Method.** El desarrollo incremental e iterativo es también una parte esencial de un tipo de Programación conocido como Extreme Programming y los demás frameworks de desarrollo rápido de software.

CAPITULO IV
RUP (RATIONAL UNIFIED PROCESS)
PROCESO UNIFICADO DE RATIONAL

4. Introducción:

RUP es una metodología sólida, con documentación que apoya el ciclo de vida evolutivo incremental, además de orientarse al desarrollo de componentes secundando el desarrollo orientado a objetos[6] RUP es un proceso de ingeniería de software que provee un enfoque disciplinado para la asignación de tareas y responsabilidades dentro de una organización. Su principal objetivo es asegurar la producción de software de alta calidad que satisfaga las necesidades de sus usuarios finales dentro de un presupuesto y tiempo predecibles

Debido a las características que posee de ser una herramienta flexible, le permite un marco de trabajo más amplio el cual puede ser adaptado tanto a empresas grandes como pequeñas y puede ser modificada para ajustarse a la forma de trabajo de una compañía

El Proceso Unificado tiene dos dimensiones (Figura 7):

- Un eje horizontal que representa el tiempo y muestra los aspectos del ciclo de vida del proceso a lo largo de su desenvolvimiento
- Un eje vertical que representa las disciplinas, las cuales agrupan actividades de una manera lógica de acuerdo a su naturaleza.

La primera dimensión representa el aspecto dinámico del proceso conforme se va desarrollando, se expresa en términos de fases, iteraciones e hitos (milestones).

[6] Kruchten

La segunda dimensión representa el aspecto estático del proceso: cómo es descrito en términos de componentes del proceso, disciplinas, actividades, flujos de trabajo, artefactos y roles.

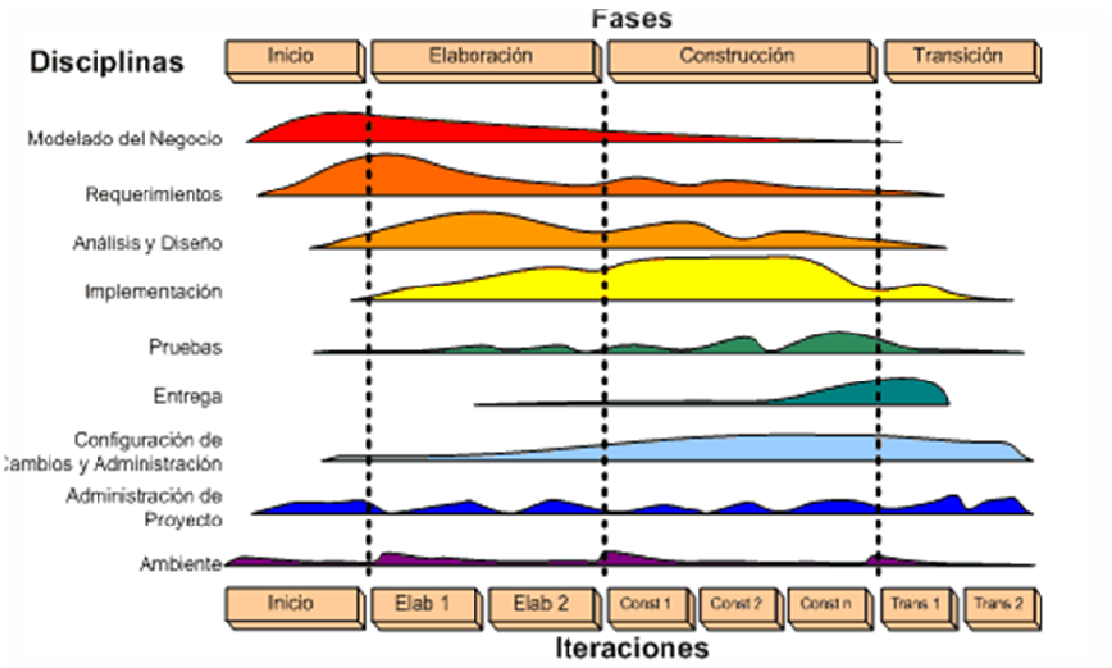


Figura 7 Dimensiones de RUP

4.1 Características de RUP

- Interactivo. Refinamiento sucesivo
- Controlado. Gestión de requisitos y control de cambios
- Construcción de modelos
- Centrado en arquitectura
- Desarrollo de software basado en componentes
- Conducido por los casos de uso
- Soporta técnicas OO (Orientadas a objetos) uso del UML
- Configurable
- Fomenta al control de calidad del software
- Soportado por herramientas

- Reconoce que las necesidades del usuario y sus requerimientos no se pueden definir completamente al principio
- Permite evaluar tempranamente los riesgos en lugar de descubrir problemas en la integración final del sistema. Reduce el costo del riesgo a los costos de un solo incremento
- Acelera el ritmo del esfuerzo de desarrollo en su totalidad debido a que los desarrolladores trabajan para obtener resultados claros a corto plazo
- Distribuye la carga de trabajo a lo largo del tiempo del proyecto ya que todas las disciplinas colaboran en cada iteración. Facilita la reutilización del código teniendo en cuenta que se realizan revisiones en las primeras iteraciones lo cual además permite que se aprecien oportunidades de mejoras en el diseño
- El proceso de desarrollo está dividido en Fases a lo largo del tiempo cada una de las cuales tiene objetivos específicos y un conjunto de "artefactos" definidos que deben alcanzarse. La duración de cada fase depende del equipo y del producto a generar. A su vez, cada fase puede tener una o más iteraciones y cada iteración sigue el modelo en cascada pasando por las distintas disciplinas. Cada iteración termina con una liberación del producto.

4.2 Principio de desarrollo:

Basado en cinco principios:

4.2.1 Adaptar el Proceso:

El proceso deberá adaptarse a las características propias del proyecto u organización, El tamaño del mismo, así como su tipo o las regulaciones que lo condicionen, influirán en su diseño específico. También se deberá tener en cuenta el alcance del proyecto.

4.2.2 Balancear prioridades:

Los requerimientos de los diversos participantes pueden ser diferentes, contradictorios o disputarse recursos limitados. Debe encontrarse un balance que satisfaga los deseos de todos.

4.2.3 Demostrar valor iterativamente:

Los proyectos se entregan, aunque sea de un modo interno, en **etapas iteradas**. En cada iteración se analiza la opinión de los inversores, la estabilidad y calidad del producto, y se refina la dirección del proyecto así como también los riesgos involucrados

4.2.4 Elevar el nivel de abstracción:

Este principio dominante motiva el uso de conceptos reutilizables tales como patrón del software, lenguajes 4GL[7] o marcos de referencia (frameworks) por nombrar algunos. Esto evita que los ingenieros de software vayan directamente de los requisitos a la codificación de software a la medida del cliente, sin saber con certeza qué codificar para satisfacer de la mejor manera los requerimientos y sin comenzar desde un principio pensando en la reutilización del código. Un alto nivel de abstracción también permite discusiones sobre diversos niveles y soluciones arquitectónicas. Éstas se pueden acompañar por las representaciones visuales de la arquitectura, por ejemplo con el lenguaje UML.

4.2.5 Enfocarse en la Calidad:

El control de calidad no debe realizarse al final de cada iteración, sino en todos los aspectos de la producción. El aseguramiento de la calidad forma parte del proceso de desarrollo y no de un grupo independiente

[7] Lenguaje de cuarta generación, estos lenguajes poseen un alto nivel de abstracción

4.3 Ciclo de vida de RUP

El ciclo de vida RUP es una implementación del Desarrollo en espiral(tratado en el capítulo III sección 3.2.5) Fue creado ensamblando los elementos en secuencias semi-ordenadas. El ciclo de vida organiza las tareas en fases e iteraciones.

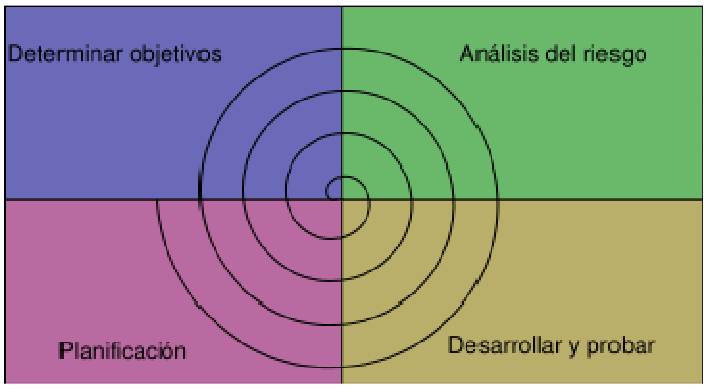


Figura 8. Ciclo de vida de RUP, basado en el ciclo de vida en espiral

4.4 Proceso

El Proceso Unificado es un proceso porque "define quién está haciendo qué, cuándo lo hace y cómo alcanzar cierto objetivo, en este caso el desarrollo de software" [Jacobson 1998]. Según [Booch 1998], los conceptos clave del Proceso Unificado son:

Fase e iteraciones	**¿Cuándo se hace?**
Flujos de trabajo de procesos (actividades y pasos)	**¿Qué se está haciendo?**
Artefactos (modelos, reportes, documentos)	**¿Qué se produjo?**
Trabajador: un arquitecto	**¿Quién lo hace?)**

4.5 Fases

La metodología RUP, llamada así por sus siglas en inglés Rational Unified Process, divide en 4 fases el desarrollo del software. Cada Fase tiene definido un conjunto de objetivos y un punto de control especifico

Fase	Objetivos	Puntos de Control
Inicio	• Definir el alcance del proyecto • Entender que se va a construir	Objetivo del proyecto
Elaboración	• Construir una versión ejecutable de la arquitectura de la aplicación • Entender cómo se va a construir	Arquitectura de la Aplicación
Construcción	• Completar el esqueleto de la Aplicación con la funcionalidad • Construir una versión Beta	Versión Operativa Inicial de la Aplicación
Transición	• Poner a disposición la aplicación para los usuarios finales • Construir la versión Final	Liberación de la versión de la Aplicación

Cada una de estas etapas es desarrollada mediante el ciclo de iteraciones, la cual consiste en reproducir el ciclo de vida en cascada a menor escala. Los Objetivos de una iteración se establecen en función de la evaluación de las iteraciones precedentes.

4.5.1 Fase de Inicio:

Durante la fase reinicio se desarrolla una descripción del producto final, y se presenta el análisis del negocio. Esta fase responde las siguientes preguntas:

¿Cuáles son las principales funciones del sistema para los usuarios más importantes?

¿Cuáles podría ser la mejor arquitectura del sistema?

En estas fases se identifican y priorizan los riesgos más importantes

Artefactos que típicamente sobreviven en esta fase

- Un enunciado de los mayores requerimientos planteados generalmente como casos de uso
- Un boceto inicial de la arquitectura
- Una descripción de los objetivos del proyecto
- Una versión muy preliminar del plan del proyecto
- Un modelo de negocio
- Se establece caso de negocio y alcance de proyecto.
- Un documento de visión general
- Plan de proyecto.
- Modelo inicial de casos de uso
- Identificación inicial de riesgos.
- Uno o más prototipos.
- Marca de Objetivos.
- Se establece el alcance y la estimación de tiempo y costo.

4.5.2 Fase de elaboración:

Durante la fase de elaboración se especifican en detalle la mayoría de los casos de uso del producto y se diseña la arquitectura

- Las iteraciones en la fase de elaboración
- Establecen una firme compresión del problema a solucionar

- Establece la fundación arquitectural para el software
- Establece un plan detallado para las siguientes iteraciones
- Elimina los mayores riesgos
- El resultado de esta fase es la línea base de la arquitectura
- En esta fase se construyen típicamente los siguientes artefactos
- El cuerpo básico del software en la forma de un prototipo arquitectural
- Casos de prueba
- La mayoría de los casos de uso (80%) que describen la funcionalidad del sistema
- Analizar el dominio del problema
- Eliminar los elementos de mayor riesgo para el desarrollo exitoso del proyecto
- Se realizan pruebas de riesgos.
- Analizar el dominio del problema
- Eliminar los elementos de mayor riesgo para el desarrollo exitoso del proyecto
- Marca de Arquitectura**.**
- Se realizan pruebas de riesgos.

Un plan detallado para las siguientes iteraciones:

La fase de elaboración finaliza con el hito de la arquitectura del ciclo de vida, este hito se alcanza cuando el equipo de desarrollo y los stakeholders llegan a un acuerdo sobre

- Los casos de uso que describen la funcionalidad del sistema
- La línea base de la arquitectura
- Los mayores riesgos han sido mitigados
- El plan de proyecto

4.5.3 Fase de construcción:

Durante la fase de construcción se crea el producto. La línea base de la arquitectura crece hasta convertirse en el sistema completo

Al final de esta fase, el producto contiene todos los casos de uso implementados, sin embargo puede que no esté libre de defectos

Los artefactos producidos en esta fase son:

- El sistema software
- Los casos de prueba
- Los manuales de usuario
- Los componentes se desarrollan e incorporan al producto.
- Todo es probado para eliminar posibles errores y riesgos.
- Marca de Capacidad.
- Se obtiene un producto Beta que debe ser puesto en ejecución para que los usuarios den retroalimentación.

La fase de construcción finaliza con el hito de capacidad operativa inicial, este hito se alcanza cuando el equipo de desarrollo y los stakeholders llagan a un acuerdo sobre

- El producto es estable para ser usado
- El producto provee alguna funcionalidad de valor
- Todas las partes están listas para comenzar la transición

4.5.4 Fase de transición

La fase de transición cubre el período durante el cual el producto se convierte en la versión beta

Sin embargo las características se agregan a un sistema que el usuario se encuentra utilizando activamente (ambiente de desarrollo)

Los artefactos construidos en esta fase son el mismo que en la fase de construcción. El equipo se encuentra ocupando fundamentalmente en corregir y extender la funcionalidad del sistema desarrollado en la fase anterior.

- El objetivo es realizar el lanzamiento del software desarrollado a los usuarios.
- Pruebas Beta para validar el producto con la retroalimentación del usuario.
- Conversión de bases de datos.
- Enviar el producto a otros lados donde también se va a usar el producto.
- Marca de Producto**.**
- Usuarios satisfechos.
- Verificación de gastos.

La fase de transición finaliza con el hito de lanzamiento del producto

Este hito se alcanza cuando el equipo de desarrollo y los stakeholders llagan a un acuerdo sobre:

☑ Se han alcanzado los objetivos fijados en la fase de inicio

☑ El usuario está satisfecho

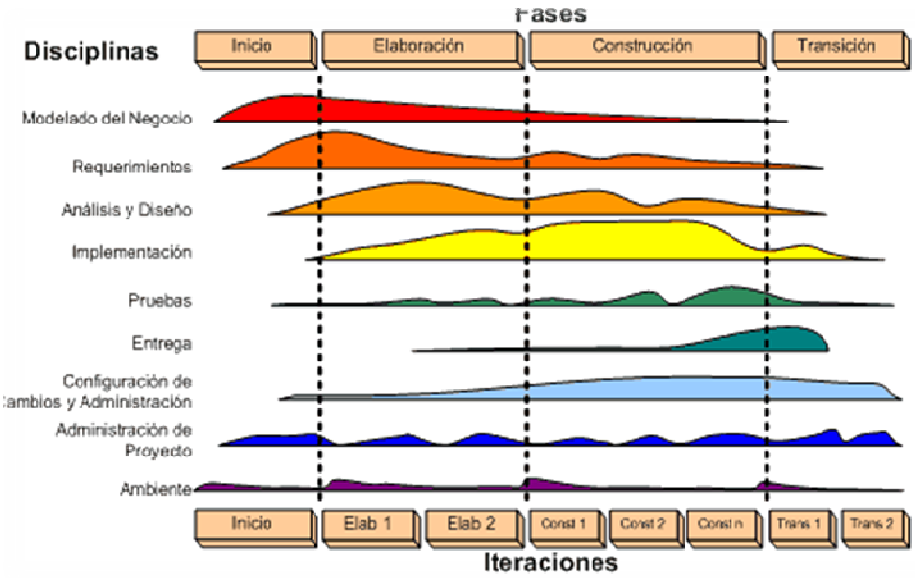


Figura 9 Fases de RUP

4.6 Grado de conocimiento de RUP en El Salvador

Para obtener ciertos datos se ha utilizado la encuesta como herramienta de medición, se ha dividido en dos tipos de encuestas una para estudiantes y otra para el sector profesional, la diferencia estriba en el grado de utilización o necesidad de uso de este tipo de herramientas.

Por la naturaleza del grupo de estudio, la determinación de la muestra utilizada se obtuvo de la **Fórmula para muestras en poblaciones finitas**, retomada por Laura Fisher y Alma Navarro de su Libro titulado "Introducción a la Investigación de Mercados ", la cual se detalla a continuación:

$$n = QNpq/ e^2(N-1) + Q^2pq$$

Donde:

Q = nivel de confianza = **95 %**

N = universo o población = **100**

p = probabilidad a favor = **50%**

q = probabilidad en contra = **50 %**

e = error de estimación (precisión de los resultados) = **5 %**

n = número de elementos (tamaño de la muestra)

Encuesta a estudiantes:

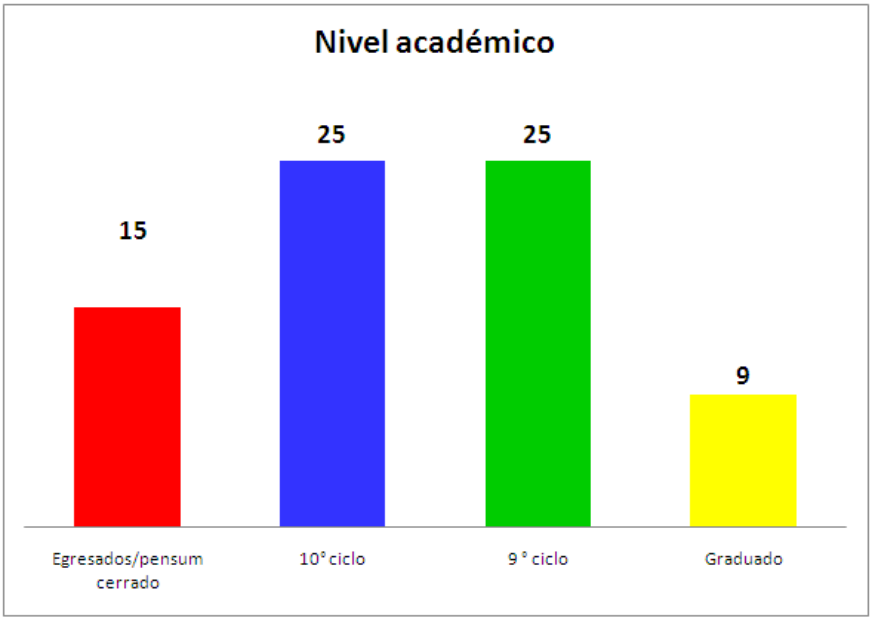
Para esta se determino un grupo representativo de 74 estudiantes de la Universidad Don Bosco, dicho número se obtuvo a través de la **Fórmula para muestras en poblaciones finitas,** anteriormente mencionada

ENCUESTA SOBRE EL GRADO DE CONOCIMIENTOS DE RATIONAL UNIFIED PROCCES[8]

Estudiantes Universitarios

Espacio muestral: 74 estudiantes de la Universidad Don Bosco, en diferentes etapas de sus carreras

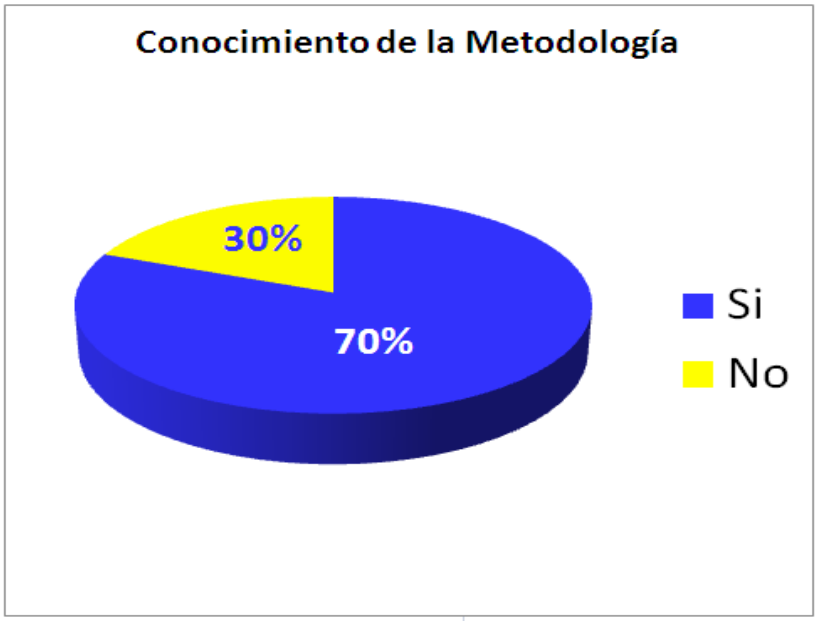
Nivel Academico	Cantidad
Egresados/pensum cerrado	15
10° ciclo	25
9 ° ciclo	25
Graduado	9



¿Sabe que es una metodología de desarrollo de software?

[8] Ver Anexo 1

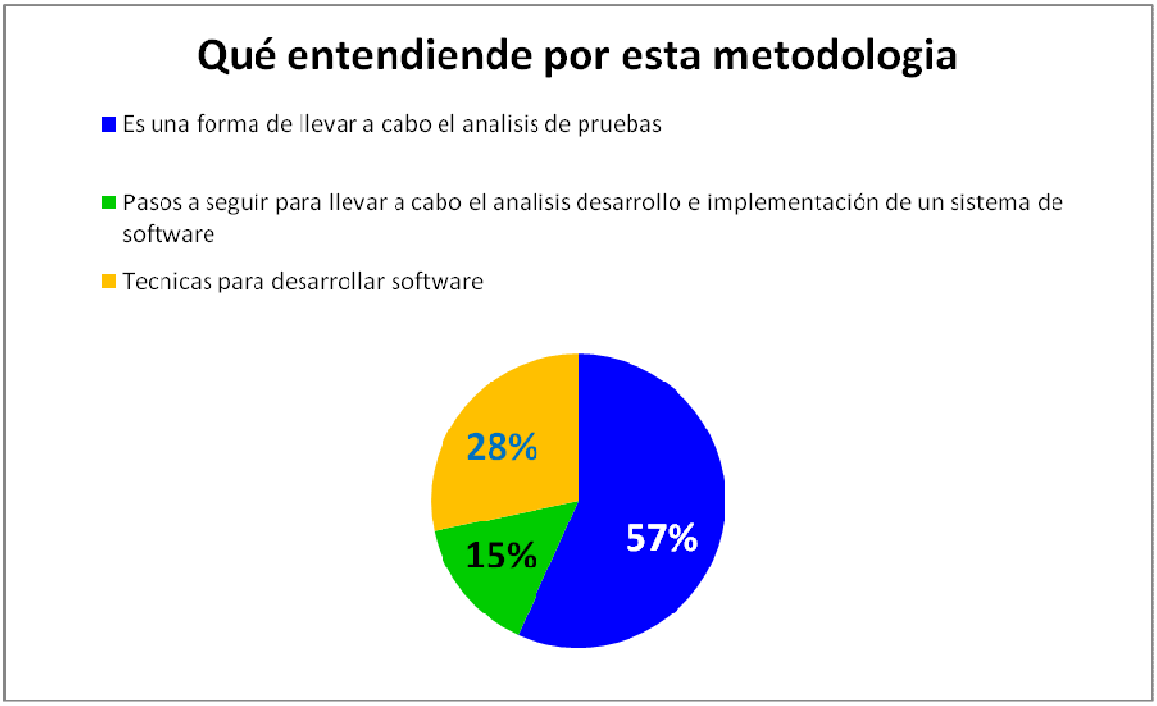
Conocimiento de la metodología	Cantidad
Si	60
No	14



60 de un total de 74 encuestados, representando un 70%, saben que es una metodología de desarrollo de software

¿Qué entiende por metodología de desarrollo de software?

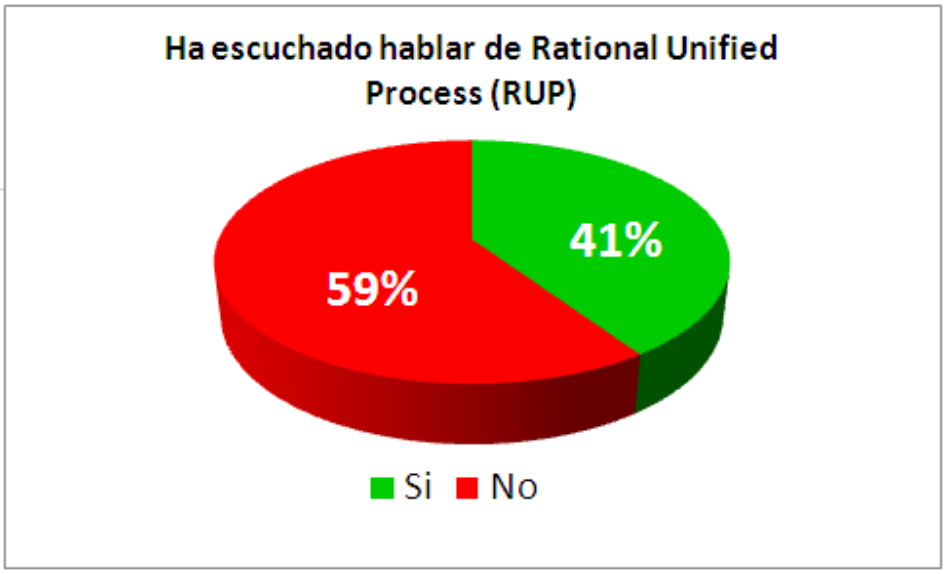
Concepto	Cantidad
Es una forma de llevar a cabo el análisis de pruebas	34
Pasos a seguir para llevar a cabo el análisis desarrollo e implementación de un sistema de software	9
Técnicas para desarrollar software	17



El 57% de personas que saben que es una metodología de desarrollo de software lo asocian, a una forma de llevar a cabo el análisis de pruebas, el 15% dicen que son pasos a seguir para desarrolla e implementar un sistema, todos concluyen que las metodologías nos ayudan a generar software de una forma ordenada

¿Ha escuchado hablar de Rational Unified Process RUP ?

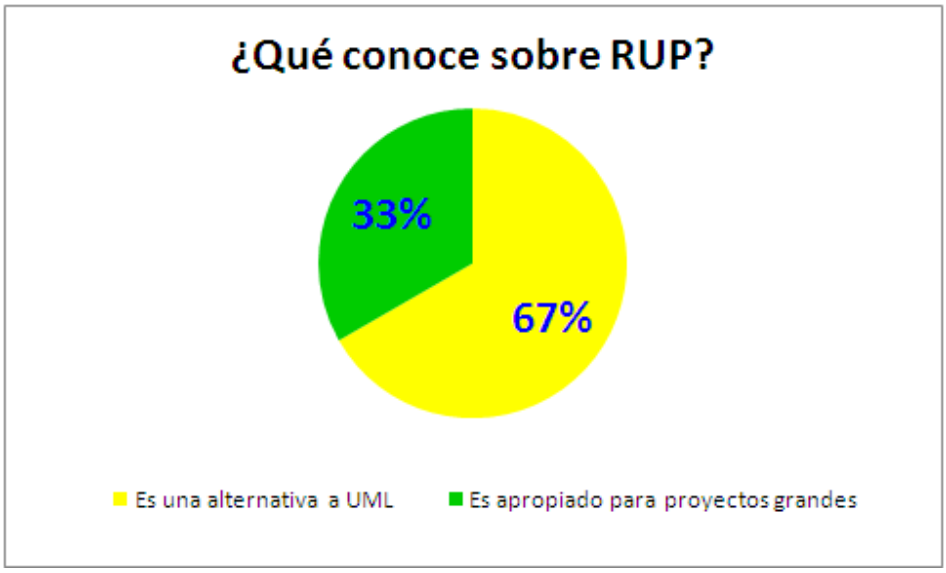
Ha escuchado hablar de Rational Unified Process (RUP)	Cantidad
Si	30
No	44



De un 70% de encuestados que dicen conocer que es una metodología de desarrollo de software, únicamente el 41 % conoce sobre RUP

¿Qué conoce sobre RUP?

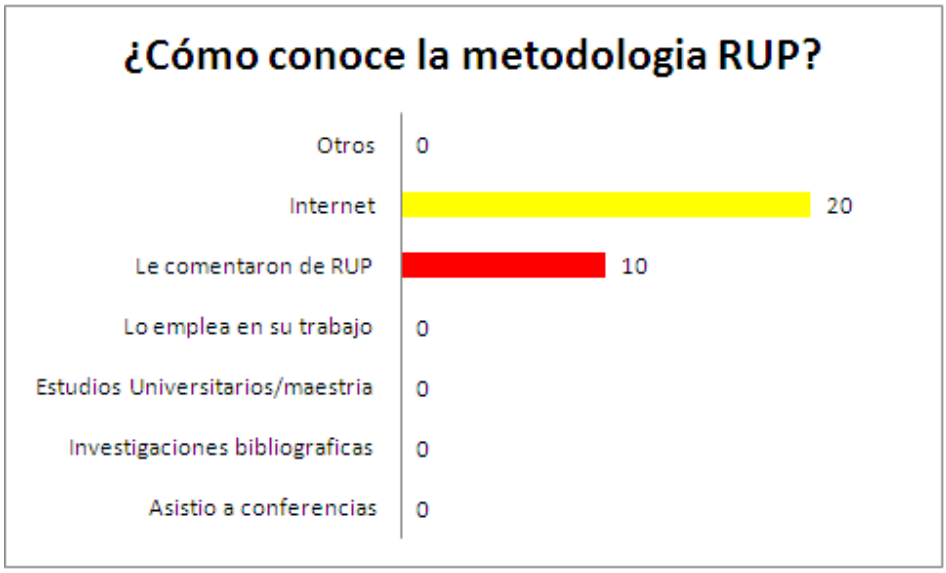
¿Qué conoce sobre RUP?	Cantidad
Es una alternativa a UML	20
Es apropiado para proyectos grandes	10



De un total de 30 personas que dijeron conocer RUP 20 consideran que RUP es utilizado en empresas grandes, mientras que 10 personas de un total de 30 (33%) lo asocian con una alternativa de UML

¿Cómo conoce esta metodología?

¿Cómo conoce la metodologia RUP?	Cantidad
Asistio a conferencias	0
Investigaciones bibliograficas	0
Estudios Universitarios/maestria	0
Lo emplea en su trabajo	0
Le comentaron de RUP	10
Internet	20
Otros	0



Internet resulto ser, el medio por medio del cual 20 de las 30 personas que respondieron conocer sobre RUP, lo vieron, mientras que el resto de encuestados dicen saber de la metodología porque les comentaron sobre esta

Sector Profesional

Para el sector profesional[9] se determino un número de tres encuestas siendo esta la cantidad de empresas de mayor reconocimiento en el área de fabricantes de software, estos según datos de la cámara de comercio e industria de El Salvador .En el siguiente cuadro se detallan los resultados de las encuestas a este sector

Pregunta	Respuestas		Conclusión
sabe que es una metodología de desarrollo de software	**Si**	**No**	Todos los encuestados del sector profesional, saben que es una metodología de desarrollo de software

[9] Ver Anexo 2.

	3	0	
Que entiende por metodología de desarrollo de software	Una manera de hacer las cosas para la construcción de software	Es un proceso ordenado que nos perite desarrollar software con resultados predecibles	Dos de los tres encuestados consideran que las metodologías de desarrollo de software nos ayudan a tener resultados predecibles, y lograr la producción de software de calidad
	1	2	
Utiliza en su trabajo alguna metodología de desarrollo de software	**Si**	**No**	Todos coinciden en el empleo de alguna metodología de desarrollo de software en su trabajo
	3	0	
Ha escuchado hablar de RUP	**Si**	**No**	Todos los encuestados en el sector profesional afirman conocer RUP y emplearlo en su trabajo
	3	0	
Que conoce sobre RUP	Proceso unificado de 4 pasos Inserción, elaboración, construcción ,transición, disciplinas, administración de proyectos, subcontratación	Herramienta que define claramente los actores y las responsabilidades de los mismos en el desarrollo del proyecto	En conclusión los encuestados coinciden en que el empleo de RUP, nos permite un mejor control de cada una de las etapas de desarrollo de los sistemas, conociendo claramente los procesos, actores etc.
	2	**1**	
¿Cómo conoce sobre RUP?	Estudios Universitarios/Maestría	Lo emplea en su trabajo	Dos de los tres encuestados conocen RUP, porque lo emplean es su trabajo uno de ellos dice haberlo conocido en sus estudios de superiores, y ahora lo emplea en su trabajo
	1	2	

Conclusiones de las encuestas.

Como podemos observar en los resultados solamente el 41 % de los encuestados conoce sobre RUP, sin embargo la gran mayoría de los que dicen conocer RUP lo asocian directamente a UML, dos de ellos consideran que RUP es más apropiado para proyectos de gran tamaño.

Veinte de las treinta personas que respondieron conocer de RUP, lo han visto en Internet y las otras diez personas les comentaron acerca de dicha metodología, cuarenta y cinco de los setenta y cuatro encuestados consideran importante que se impartan temas relacionados a esta y otras metodologías de desarrollo de software en la universidad

En el otro tipo de encuesta orientada para el sector profesional, desempeñando funciones especificas de software, se tomo un espacio muestral de tres personas de que representan las tres empresas dedicadas a la producción de software [10]específicamente, entre ellas SVSOFT S.A de C.V, todos dicen conocer la herramienta, y utilizarla en sus proceso de producción de software

Como podemos determinar, si bien es cierto esta herramienta, no se ve en la Universidad Don Bosco, está siendo utilizada por el sector empresarial, principalmente las empresas dedicadas a la producción de software

[10] Número de empresas productoras de software según datos de la cámara de Industria y comercio de El Salvador http://www.camarasal.com/cgibin/camarasal/bueno01?fabricante=on&producto=SOFTWARE&Submit=Buscar

CAPITULO V
ELEMENTOS DE RUP

5 Elementos de RUP:

Los elementos del RUP son:

- **Actividades**, Son los procesos que se llegan a determinar en cada iteración.
- **Trabajadores**, Vienen hacer las personas o entes involucrados en cada proceso.
- **Artefactos**, Un artefacto puede ser un documento, un modelo, o un elemento de modelo.
- **Disciplinas primarias y secundarias o de apoyo:** Una disciplina es una colección de actividades relacionadas con un área de atención dentro de todo el proyecto.

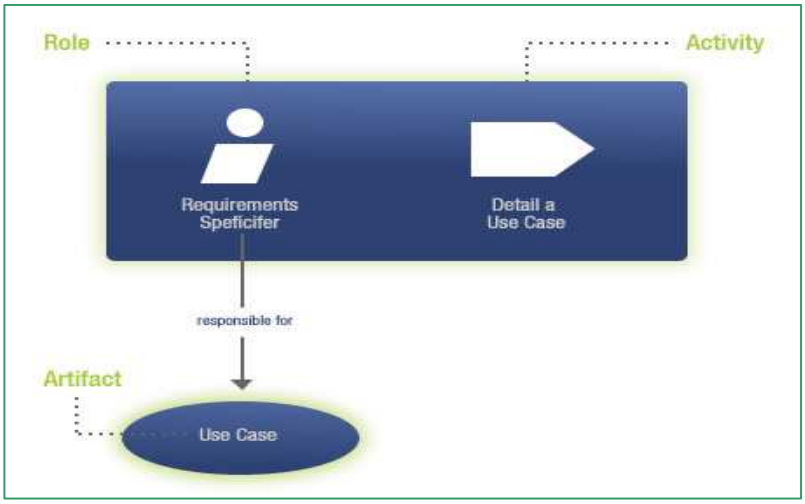


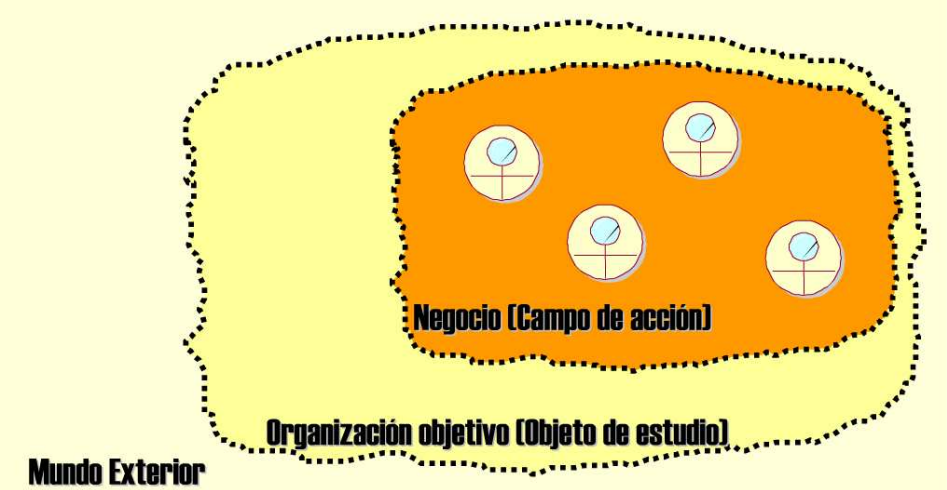
Figura 10 Elementos de RUP

5.1 Actividad:

Los trabajadores realizan actividades. Una actividad es algo que se realiza para proveer un resultado de valor en el contexto de un proyecto

5.2 Trabajador (ROL):

Un trabajador o rol, define un comportamiento o responsabilidades de un individuo o grupo de individuos trabajando en equipo, en el contexto de una organización de ingeniería de software



Sugerencias para identificar adecuadamente los trabajadores del negocio:

- ✓ Son roles (Humanos, software o hardware) no personas con nombres propios
- ✓ Se encuentran dentro de las fronteras del negocio o campo de acción
- ✓ No deben representar áreas, departamentos o parte de una organización sino roles de ejecución
- ✓ Cada trabajador debe participar en al menos un caso de uso del negocio. Si no participa en ningún proceso debe ser eliminado del modelo

5.3 Artefactos

Las actividades tienen artefactos de entrada y de salida, un artefacto es un producto de trabajo de un proceso: los trabajadores utilizan artefactos para realizar actividades y producen artefactos como resultado de sus actividades. Los artefactos son responsabilidad de un único trabajador y promueven la idea de que toda pieza de información en el proceso debe ser responsabilidad de un rol especifico, un trabajador es el propietario de un artefacto, pero otros trabajadores pueden usarlo y tal vez modificarlo si tienen permisos para ello

5.4 Actividades Primarias y secundarias o de (apoyo):

Una disciplina es una colección de actividades relacionadas con un área de atención dentro de todo el proyecto. El grupo de actividades que se encuentran dentro de una disciplina principalmente son una ayuda para entender el proyecto desde la perspectiva clásica de cascada.

Las disciplinas son:

a) Modelado de Negocios,
b) Requerimientos,
c) Análisis y Diseño,
d) Implementación,
e) Pruebas, Transición,
Actividades de poyo
f) Configuración y Administración del Cambio,
g) Administración de Proyectos y Ambiente.

a) Modelado de Negocios

Los propósitos que tiene el Modelo de Negocios son:

- ✓ Entender los problemas que la organización desea solucionar e identificar mejoras potenciales.
- ✓ Medir el impacto del cambio organizacional.
- ✓ Asegurar que clientes, usuarios finales, desarrolladores y los otros participantes tengan un entendimiento compartido del problema.
- ✓ Derivar los requerimientos del sistema de software, necesarios para dar soporte a los objetivos de la organización.
- ✓ Entender como el sistema a ser desarrollado entra dentro de la organización.

b. Requerimientos:

Esta disciplina tiene el propósito de:

- ✓ Establecer y mantener un acuerdo con los clientes y los otros interesados acerca de que debe hacer el sistema.
- ✓ Proveer a los desarrolladores del sistema de un mejor entendimiento de los requerimientos del sistema.
- ✓ Definir los límites (o delimitar) del sistema.
- ✓ Proveer una base para la planeación de los contenidos técnicos de las iteraciones.
- ✓ Proveer una base para la estimación de costo y tiempo necesarios para desarrollar el sistema.

  Definir una interfaz de usuario para el sistema, enfocada en las necesidades y objetivos del usuario.

c. Análisis y Diseño:

El propósito del Análisis y Diseño es:

- ✓ Transformar los requerimientos a diseños del sistema.
- ✓ Desarrollar una arquitectura robusta para el sistema.
- ✓ Adaptar el diseño para hacerlo corresponder con el ambiente de implementación y ajustarla para un desempeño esperado.

d. Implementación:

El propósito de la implementación es:

- ✓ Definir la organización del código, en términos de la implementación de los subsistemas organizados en capas.
- ✓ Implementar el diseño de elementos en términos de los elementos (archivos fuente, binarios, ejecutables y otros)
- ✓ Probar los componentes desarrollados como unidades.
- ✓ Integrar los resultados de los implementadores individuales en un sistema ejecutable.

La disciplina de implementación limita su alcance a como las clases individuales serán probadas. Las pruebas del sistema son descritas en futuras disciplinas.

e. Pruebas:

Esta disciplina actúa como un proveedor de servicios a las otras disciplinas en muchos aspectos. Pruebas se enfoca principalmente en la evaluación y aseguramiento de la calidad del producto, desarrollado a través de las siguientes prácticas:

- ✓ Encontrar fallas de calidad en el software y documentarlas.
- ✓ Recomendar sobre la calidad percibida en el software.
- ✓ Validar y probar las suposiciones hechas durante el diseño y la especificación de requerimientos de forma concreta.
- ✓ Validar que el software trabaja como fue diseñado.
- ✓ Validar que los requerimientos son implementados apropiadamente.

f. Transición

Esta disciplina describe las actividades asociadas con el aseguramiento de la entrega y disponibilidad del producto de software hacia el usuario final.

Existe un énfasis en probar el software en el sitio de desarrollo, realización de pruebas beta del sistema antes de su entrega final al cliente.

g. Administración y Configuración del Cambio:

Consiste en controlar los cambios y mantener la integridad de los productos que incluye el proyecto.

Incluye:

- Identificar los elementos configurables.
- Restringir los cambios en los elementos configurables.
- Auditar los cambios hechos a estos elementos.
- Definir y mantener las configuraciones de estos elementos.
- Los métodos, procesos y herramientas usadas para proveer la administración y configuración del cambio pueden ser consideradas como el sistema de administración de la configuración.

h. Administración de Proyectos:

El propósito de la Administración de Proyectos es:

- ✓ Proveer un marco de trabajo para administrar los proyectos intensivos de software.
- ✓ Proveer guías prácticas para la planeación, soporte, ejecución y monitoreo de proyectos.
- ✓ Proveer un marco de trabajo para la administración del riesgo.

i. Ambiente:

Se enfoca en las actividades necesarias para configurar el proceso al proyecto. Describe las actividades requeridas para desarrollar las líneas guías de apoyo al proyecto.

El propósito de las actividades de ambiente es proveer a las organizaciones de desarrollo de software del ambiente necesario (herramientas y procesos) que den soporte al equipo de desarrollo.

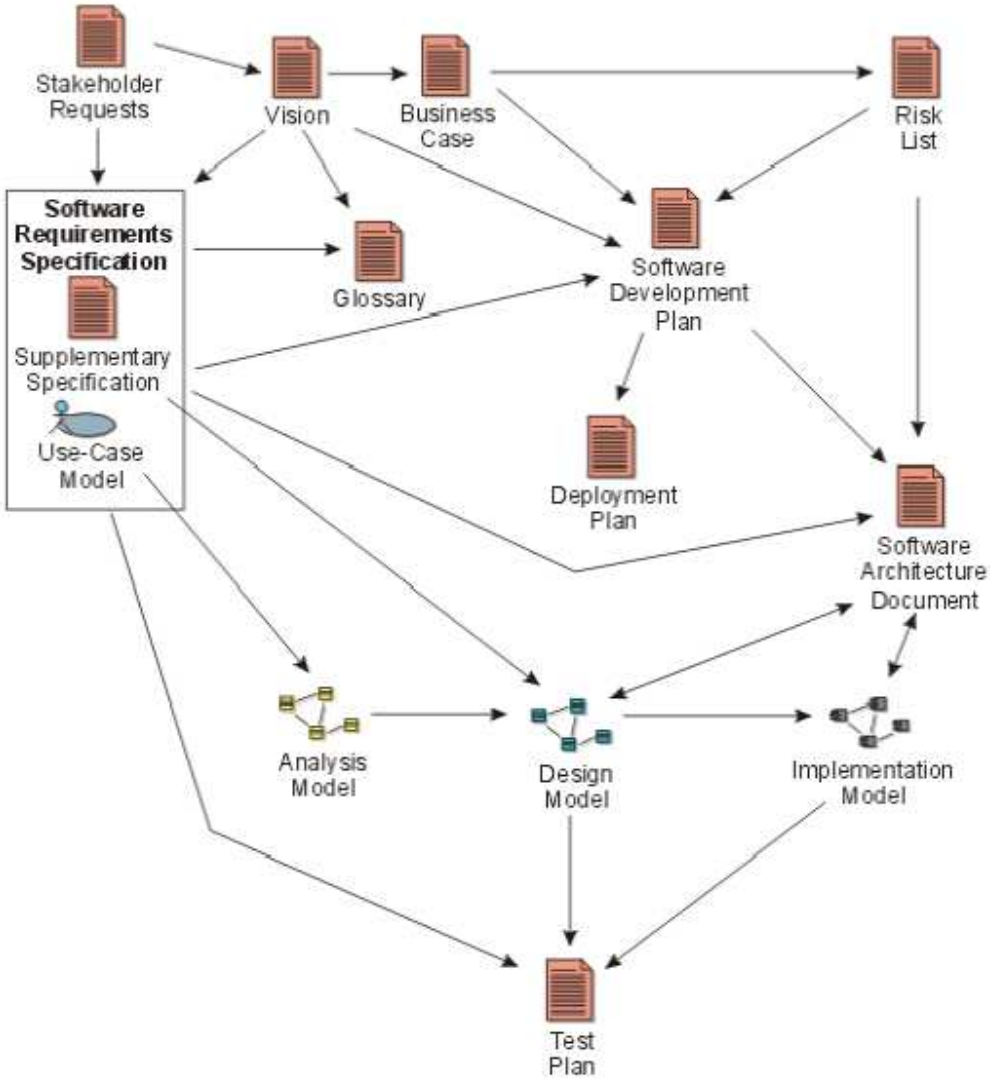


Figura 11 Principales Artefactos utilizados en el proceso de flujo de información

La figura 12 muestra como la información fluye a través de la fase de requisitos vía los artefactos,

Actividad	Trabajador	Resp. De artefactos (salida)	Artefactos (entradas)
REQUISTOS			
1 Encontrar actores y casos de uso	Analista de sistemas	Modelo de casos de uso (esbozado) Glosario	Modelo del negocio o modelo del dominio. Requisitos adicionales Lista de características
2 Priorizar casos de uso	Arquitecto	Descripción de la arquitectura (Vista del mod. De caso de uso)	Modelo de casos de uso (esbozado)
3 Detallar casos de uso	Especificador de casos de uso	Casos de uso (detallado)	Modelo de casos de uso (esbozado) Requisitos adicionales Glosario
4 Estructurar el modelo de casos de uso	Analista de sistemas	Modelo de casos de uso (Estructurado)	Modelo de casos de uso (esbozado) Requisitos adicionales Caso de uso (detallado) Glosario
5 Prototipar interfaz de usuario	Diseñador de interfaz de usuario	Prototipo de interfaz de usuario	Modelo de casos de uso Requisitos adicionales Caso de uso (detallado) Glosario

Figura 12 Actividades, trabajador y artefactos en la fase de requisitos

Ejemplo utilizando modelado del negocio

Para este caso se tomara como ejemplo una empresa de deportes, la cual interactúa con muchos clientes

El modelado del negocio se basa en los diagramas principales Modelado de casos de uso (Figura 13) , Modelado de dominio (figura 14) y Modelado de objetos del negocio (Figura 15)

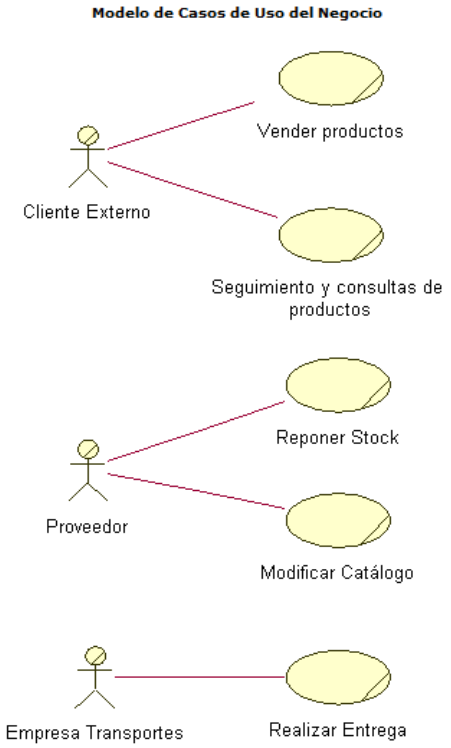


Figura 13. Modelado de casos de uso del negocio

Los modelos de objetos del dominio están asociados a cada uno de los casos de uso del negocio. Por ser de mayor prioridad para la empresa, el caso de uso para el cual se desarrolló el modelo de objetos fue el del caso de uso del negocio "Vendedor productos"

Figura 14. Modelado del dominio

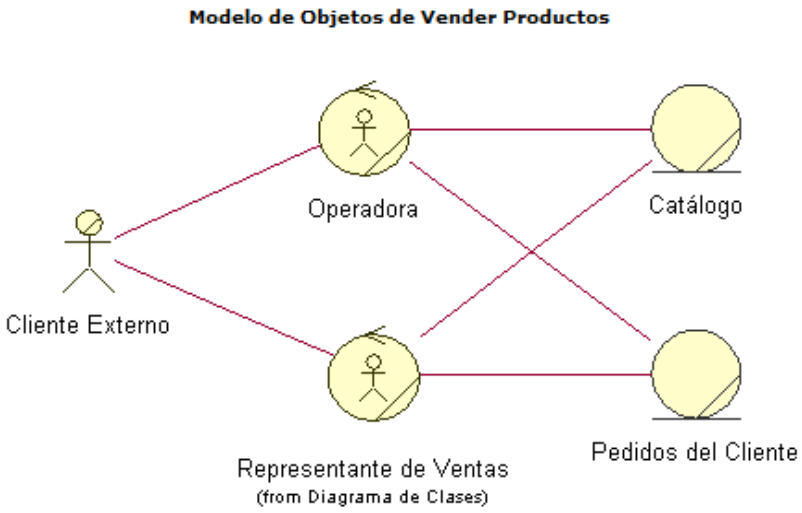


Figura 15. Modelo de objetos de vendedor productos

CAPITULO VI
CASO PRÁCTICO EMPLEANDO RUP

6 Introducción:

Volvo information Technology (VOLVO IT) es una filial de AB volvo, uno de los grupos industriales más grandes de la región Nórdica, Volvo IT provee todo tipo de soluciones industriales IT (Information Technology) en una variedad de ambientes técnicos, la compañía fue fundada en 1998, a través de una fusión de todos los recursos IT de diferentes compañías del grupo VOLVO

6.1Caso VOLVO IT

6.1.1 Experiencia con los proyectos RUP

"Implementar y empezar a usar un nuevo proceso común para el desarrollo de aplicaciones implica una gran inversión para una compañía como VOLVO IT. Por consiguiente fue muy importante para nosotros recibir feedback, de cómo el RUP en sí y el soporte fueron recibidos por los miembros del equipo del proyecto RUP y los clientes ", esta son las palabras de las personas que estuvieron a cargo de la implementación del proyecto de RUP en volvo IT

Para medir lo anterior utilizaron cuestionarios, que al mismo tiempo ayudaron a evaluar el soporte y la herramienta RUP, la evaluación se realizo con clientes representativos, el administrador del proyecto y el equipo del proyecto cuando el proyecto RUP termino, el cuestionario elaborado por VOLVO se centro en cinco áreas diferentes:

- ☑ RUP vs. "La forma tradicional de trabajar"
- ☑ Desarrollar procesos
- ☑ Entrenamiento
- ☑ Acompañamiento y soporte
- ☑ EL enfoque iterativo

Para cada área se utilizaron diversas preguntas a fin de encontrar las opiniones de todos los involucrados, una vez se obtuvo el resultado fue compilado y se presento al equipo de seguimiento de MAPS (Method and process for system development)

El objetivo que perseguía VOLVO IT era que al menos un 80% de los encuestados estuvieran satisfechos (nivel 3 o 4 en una escala de 1 a 4). Los resultados excedieron los objetivos en algunos de estos casos en el 95%

De entre todos los datos reflejados en los resultados de las encuestas podemos decir que los puntos más importantes son:

- El foco en los requerimientos y los riegos durante el proyecto se estima especialmente
- Se espera que el costo de mantenimiento de la aplicación sea más bajo en comparación con mantener una aplicación desarrollada de la manera tradicional
- Implementar un nuevo proceso de desarrollo de aplicación es una inversión en competencia, y debe ser considerada como una mejora a largo plazo

6.1.2 Efectos de usar RUP

Para poder medir los efectos de usar RUP, se tomaron en cuenta varios aspectos entre ellos tenemos el número total de desarrolladores (Programadores) que fueron entrenados en RUP fue de aproximadamente 1000 personas, lo que equivale a una inversión en el rango de 5 – 10 millones de dólares, así una de las preguntas de la alta gerencia no fue una sorpresa “Puede usted probar que RUP es una buena inversión ” A menos que los efectos positivos de usar RUP en la primera ronda del proyecto pudieran ser documentados, porque deberíamos continuar ?

Lo que se pudo determinar

El cuestionario fue diseñado para proporcionar un feedback de los proyectos RUP pilotos evidentemente no sería considerado una prueba objetiva, es normal ser positiva cuando se está entre los primeros que trabajan con un nuevo proceso y usando nuevas herramientas

Solo mirando la precisión de entrega (Tiempo actual y costo comparado con estimaciones iníciales), de los proyectos usando RUP no sería suficiente para juzgar los efectos de RUP, por el contrario el enfoque interactivo permite al cliente y el proveedor estar de acuerdo con cambios en el alcance y las prioridades a medida que el conocimiento del problema y la solución aumenta

Para evaluar la capacidad del proceso, evaluando la manera tradicional de trabajo" de un equipo involucrado en un proyecto, y luego comparando estos datos con los resultados de una evaluación del mismo equipo "trabajando con RUP" se esperaba decir que la capacidad de proceso fue mejorada. Para hacer estas mediciones se empleo el framework SPICE (Software Process Improvement and capability Eterminatio)

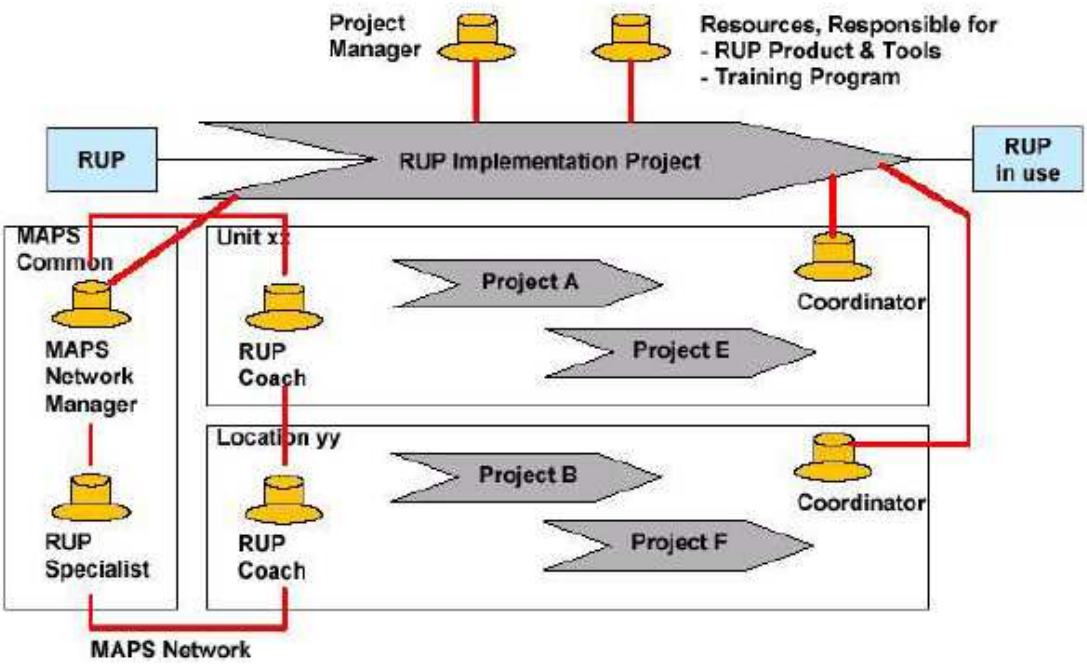


Figura 16 Organización de la implementación del proyecto RUP

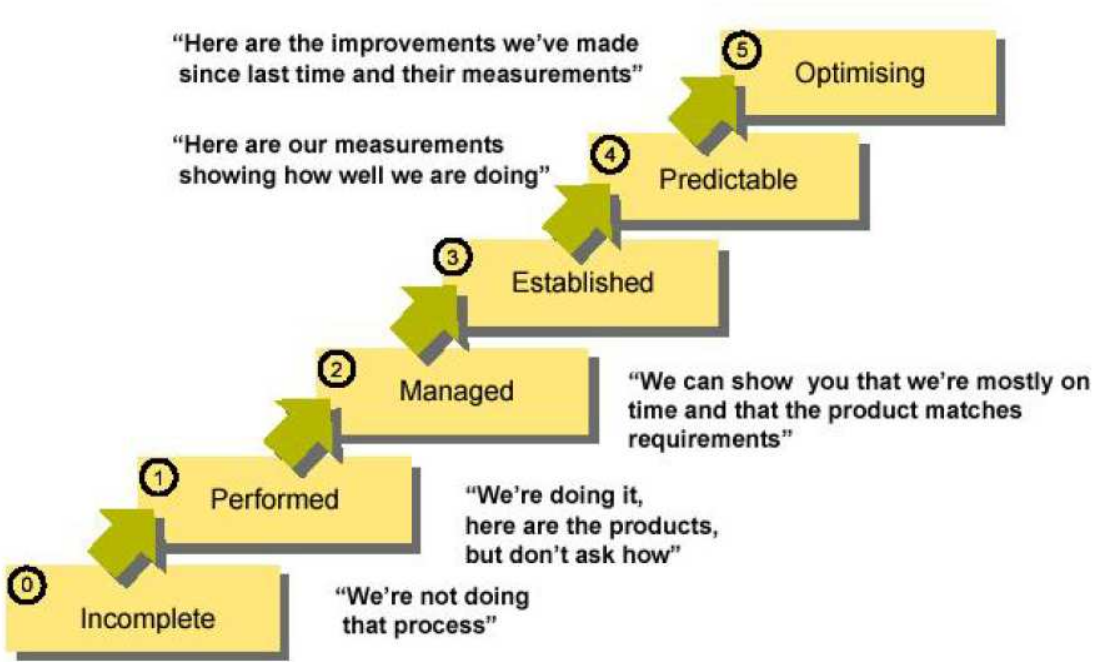


Figura 17, niveles de capacidad de SPICE

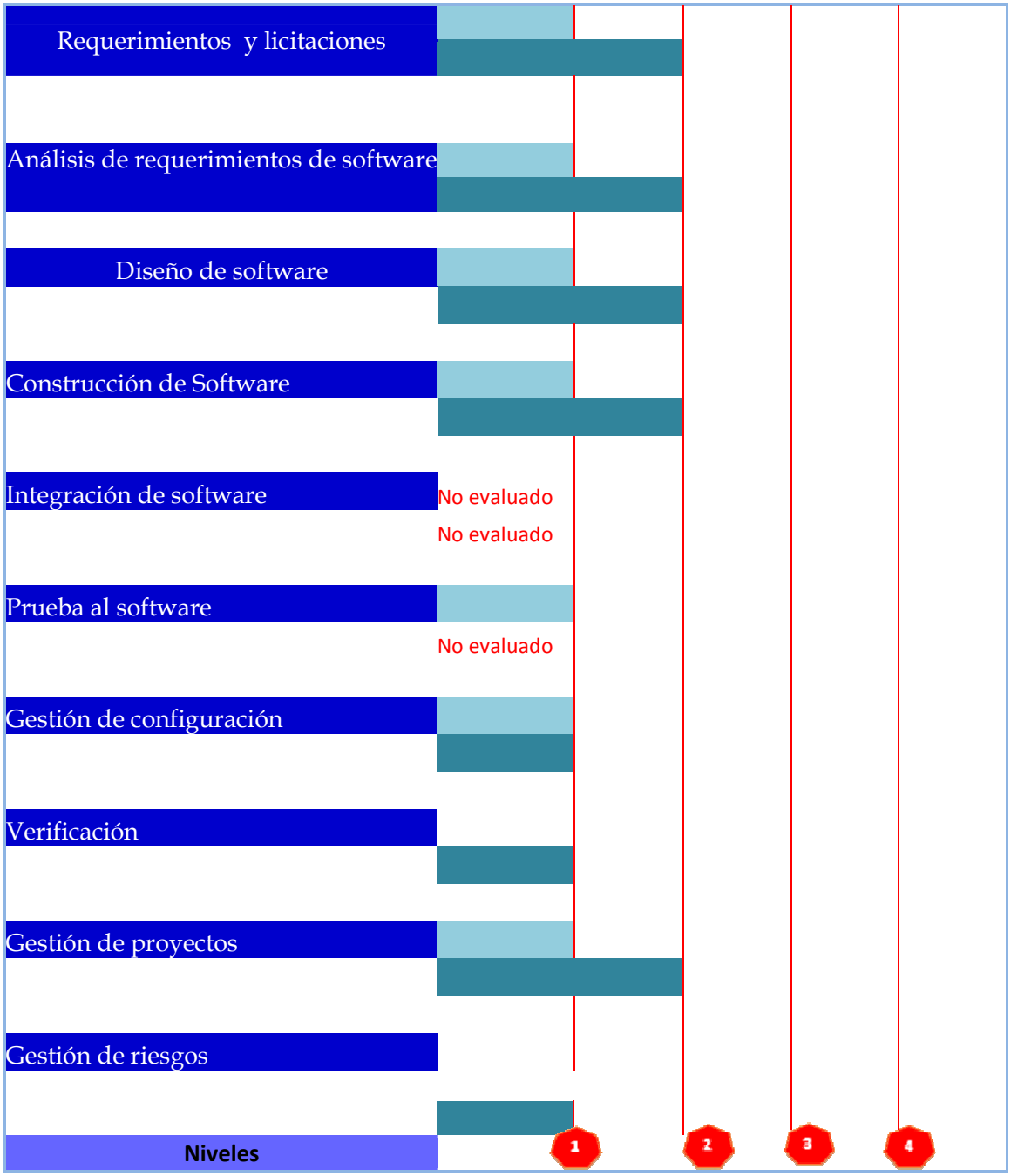

Requerimientos y licitaciones
Análisis de requerimientos de software
Diseño de software
Construcción de Software
Integración de software
No evaluado
No evaluado
Prueba al software
No evaluado
Gestión de configuración
Verificación
Gestión de proyectos
Gestión de riesgos
Niveles
1
2
3
4

Figura 18. Procesos normales Vrs procesos utilizando RUP

El diagrama muestra la capacidad incremento de nivel 1 a nivel 2 en los procesos de requerimiento, análisis, diseño y administración del proyecto, los procesos de verificación y administración de riesgo no fueron ejecutados del todo de la forma tradicional de trabajo. “trabajando con RUP” estos fueron clasificados en nivel 1

Nota: los niveles más óptimos son los niveles 4 y 3,y en menor escala niveles 2 y 1

CONCLUSIONES

- Se creó un documento en el que se abordan de una forma general los conceptos de ingeniería de software, las metodologías de desarrollo de software
- Se determino a través de encuestas que son muy pocos los estudiantes y profesionales del área de informática que conocen sobre Rational Unified Process (RUP), y la mayor parte de los que dijeron conocerla es a nivel general
- El tener un enfoque disciplinado para la asignación de tareas y responsabilidades dentro de una organización; nos llevan al cumplimiento de uno de los objetivos que se sigue, el cual es la producción de software de alta calidad que satisfaga las necesidades de sus usuarios finales dentro de un presupuesto y tiempo predecibles
- Se realizo una descripción de forma general de los elementos que componen RUP, sus características, ciclo de vida, y su lógica de funcionamiento
- Se conoció el caso de éxito de VOLVO IT, el cual nos muestra los beneficios que esta compañía logro luego de la implementación de RUP, si bien es cierto los primeros meses se realizaron inversiones en cuanto a entrenamientos para el personal involucrado, las ganancias que se obtienen por tener un sistema ordenado y con roles bien definidos, represento una gran ganancia en tiempo y dinero para esta compañía

BIBLIOGRAFIA

- Amador Duran Toro, Beatriz Bernrdez Jiménez, **"Metodología para el análisis de de requisitos de sistemas de software versión 2.2",** Universidad de Sevilla, departamento de Lenguajes y Sistemas informáticos, escuela Técnica superior de Ingeniería Informática, Diciembre de 2001
- Roger S.Pressman . **"Ingeniería del Software: un enfoque practico",** de segunda edición, Editorial McGraw Hill. 1990
- Manuel Díaz Rodríguez, Antonio Moña Gómez, **"Ingeniería de Software Especificación",** Departamento de Lenguajes de Ciencias de la computación, Universidad de Málaga

GLOSARIO

Abstracción: Consideración aislada de las cualidades de un objeto o del mismo objeto en su pura esencia o noción

Artefacto: Producto del trabajo de un proceso

Atributo: característica de un archivo, carpeta, sistema que define su función o su naturaleza

Arquitectura: En informática, se refiere a la forma de estructurar una computadora, un sistema operativo, un microprocesador, un software, etc.

Aplicación: Programa informático que permite a un usuario utilizar una computadora con un fin específico. Las aplicaciones son parte del software de una computadora, y suelen ejecutarse sobre el sistema operativo

BUGS: También conocidos como holes o agujeros. Defecto en un software o un hardware que no ha sido descubierto por los creadores o diseñadores de los mismos

Calidad: Es la totalidad de los rasgos y características de un producto o servicio que se sustenta en su habilidad para satisfacer las necesidades establecidas implícitas

Caso de uso: secuencia de acciones realizada por un sistema que produce un resultado observable de valor para un actor particular

Código Fuente: Texto escrito en un lenguaje de programación específico y que puede ser leído por un programador. Debe ser traducido a lenguaje maquina

Compilador: Proceso de traducción de un código fuente (escrito en un lenguaje de programación de alto nivel) a lenguaje máquina (código objeto) para que pueda ser ejecutado por la computadora.

Costes: o costo, es el gasto económico que representa la fabricación de un producto o la presentación de un servicio.

Cualitativo: Referente a las cualidades que posee un objeto

Cuantitativo: Adjetivos que le dan un sentido que se puede expresar numéricamente

Cognitivo: Este término s utilizado por la psicología moderna, concediendo mayor importancia a los aspectos intelectuales que a los afectivos y emocionales, en este sentido se tiene un doble significado: primero, se refiere a una representación conceptual de los objetos. La segunda, es la comprensión o explicación de los objetos.

Extreme Programming: Es el más destacado de los procesos ágiles de desarrollo de software

Etapas: Parte de un proceso, pasos en los que se beben cumplir objetivos específicos

Encapsular: El encapsulamiento es el proceso por el cual los datos que se deben enviar a través de una red se deben colocar en paquetes que se puedan administrar y rastrear. El encapsulado consiste pues en ocultar los detalles de implementación de un objeto, pero a la vez se provee una interfaz pública por medio de sus operaciones permitidas. Considerando lo anterior también se define el encapsulado como la propiedad de los objetos de permitir el acceso a su estado únicamente a través de su interfaz o de relaciones preestablecidas con otros objetos.

Framework: es una estructura de soporte definida, mediante la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software

Fiabilidad: Probabilidad de que una cosa funcione bien o sea segura

Interactivo: hace referencia a los programas que aceptan y responden entradas en datos y comandos por parte de los humanos. La interactividad está muy relacionada a la interfaz de un programa

Intranet: Red entre computadoras montada para el uso exclusivo dentro de una empresa u hogar. Se trata de una red privada que puede o no tener acceso a Internet. Sirve para compartir recursos entre computadoras

Internet: Conocida como la red de redes, pues se trata de una de las redes más grandes con un estimado de mil cien millones de usuarios (2007). Para funcionar utiliza el conjunto de protocolos TCP/IP..

Metodología: Conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar nuevo software

Mitigar: Moderar, aplacar, disminuir o suavizar algo

Orientado a Objetos: Tipo de lenguaje de programación basado en la idea de encapsular estado y operaciones en objetos. En general, la programación se resuelve comunicando dichos objetos a través de mensajes (programación orientada a mensajes).

Paradigma: Un modelo o patrón, esta palabra se usa para denominar elementos que siguen algún diseño o modelo

Procedimental: Relacionado con un procedimiento

Programación: es un proceso por el cual se escribe (en un lenguaje de programación), se prueba, se depura y se mantiene el código fuente de un programa informático.

Reutilización: es el uso de software existente para desarrollar un nuevo software. La reutilización de código ha sido empleada desde los primeros días de la programación. Los programadores siempre han reutilizado partes de un código, planillas, funciones etc.

Requisitos: Describen los servicios que se esperan del sistema

Sistemas evolutivos: sistemas capaces de construir y mantener en tiempo real una imagen del ambiente que los rodea, como un ser que construye su propia imagen de la realidad y la utiliza para interactuar con su entorno

Sistemático: Método de ordenación, organización o clasificación de los elementos

Sistema operativo: Sistema tipo software que controla la computadora y administra los servicios y sus funciones como así también la ejecución de otros programas compatibles con éste. Ejemplo de familias de sistemas operativos Windows, Linux, Mac, etc.

Universo del discurso: Descripción abstracta y general de la parte o sector del universo real que el contenido de la base de datos va a representar

.

ANEXOS

RATIONAL UNIFIED PROCESS (RUP)

Encuesta sobre conocimientos de RUP

Objetivo:
La presente encuesta pretende medir el grado de conocimiento que las personas del área de informática tienen acerca de la metodología de desarrollo de software RUP

INFORMACION PERSONAL:

Nombre: _____

Universidad: _____

Carrera: _____

Ciclo Actual: _____

Esta trabajando en estos momentos:

Si ___ No ___

Nombre de la Empresa: _____

Cargo: _____

METODOLOGIA DE DESARROLLO DE SOFTWARE EN GENERAL:

1) **Sabe que es una metodología de desarrollo de Software**

Si ___ No ___

NOTA: Si su repuesta es Si continúe con la pregunta 2, de lo contrario diríjase a la pregunta 9

2) ¿Que entiende por metodología de desarrollo de software?

3) **¿Vio en la Universidad un contenido sobre metodologías de desarrollo de software?**

4) **En que materia vio este contenido**

5) **Alguna vez utilizo alguna metodología de desarrollo de software (UML, eXtremme, programming, RUP) para crear los proyectos de cátedra,**

6) **¿En que Materia?**

7) **¿En su trabajo utiliza alguna herramienta de ingeniería de software para sus proyectos?**

Si___ NO___

8) **Considera importante el utilizar una metodología que le permita optimizar los recursos materiales y humanos**

9) **Considera importante que se imparta en la Universidad contenidos de esta clase ?**

Muy Importante _____

Importante _____

No muy importante _____

No sabe _____

Irrelevante _____

RATIONAL UNIFIED PROCESS:

1) Ha escuchado hablar de Rational Unified Process (RUP)?

Si___ No ___

2) ¿Qué conoce sobre RUP?

3) **¿Como conoce esta metodología?**

Asistió a Conferencias _____ Lo emplea en su trabajo: _____

Investigaciones bibliográficas: _____ Le comentaron de RUP _____

Estudios Universitarios/maestría _____ Internet: : _____

Otros: _____

4) Estaría dispuesto a adoptar esta metodología de desarrollo para los proyectos de su empresa/cátedra

5) Considera importante que se imparta el contenido de esta metodología en la Universidad

Si___ No ___

ANEXO 2

RATIONAL UNIFIED PROCESS (RUP)

Encuesta sobre conocimientos de RUP

Objetivo:
La presente encuesta pretende medir el grado de conocimiento que las personas del área de informática tienen acerca de la metodología de desarrollo de software RUP

INFORMACION PERSONAL:

Nombre: _____

Empresa: _____

Cargo: _____

METODOLOGIA DE DESARROLLO DE SOFTWARE EN GENERAL:

2) Sabe que es una metodología de desarrollo de Software

Si ___ No___

NOTA: Si su repuesta es SI continúe con la pregunta 2, de lo contrario diríjase a la pregunta 6

3) ¿Que entiende por metodología de desarrollo de software?

Una manera de hacer las cosas para la construcción de software

4) Utiliza en su trabajo alguna metodología de desarrollo?

Si ___ No ___

5) ¿Qué herramienta de desarrollo de software utiliza?

6) Considera importante el utilizar una metodología que le permita optimizar los recursos materiales y humanos

7) **Que metodología, herramienta o recurso de ingeniería de software, utiliza en sus proyectos**

RATIONAL UNIFIED PROCESS:

NOTA: Responda esta parte si conoce sobre RUP

6) **Ha escuchado hablar de Rational Unified Process (RUP)?**

Si___ No ___

7) **¿Qué conoce sobre RUP?**

8) **Como conoce esta metodología**

Asistió a Conferencias _____ Lo emplea en su trabajo: _____

Investigaciones bibliográficas: _____ Le comentaron de RUP _____

Estudios Universitarios/maestría _____ Internet: : _____

Otros: _____

9) **¿Cuáles cree que serian las ventajas de usar una herramienta como esta en su organización?**

10) **Estaría dispuesto a adoptar esta metodología de desarrollo para los proyectos de su empresa**