UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN

Autor: JOSÉ SERGIO PAZ LUCAS

Ingeniería del software		Clave: 1764
Plan: 2005		Créditos: 8
Licenciatura: Informática		Semestre: 7º
Área: Informática (Gestión de la información)		Hrs. Asesoría: 4
Requisitos: Ninguno		Hrs. Por semana: 4
Tipo de asignatura:	Obligatoria (x)	Optativa ( )

Objetivo general de la asignatura

Al finalizar el curso, el alumno integrará los conocimientos previos de análisis y diseño de sistemas para el desarrollo de software de calidad, además de obtener las metodologías, técnicas y herramientas para desarrollar sistemas informáticos en el tiempo y costos establecidos.

Temario oficial (64 horas sugeridas)

1. Fundamentos de la ingeniería de software (6 horas)
2. Software (6 horas)
3. Administración de proyectos (8 horas)
4. Administración de requerimientos (6 horas)
5. Análisis de software (12 horas)
6. Diseño de software (10 horas)
7. Verificación y validación del software (4 horas)
8. Mantenimiento del software (4 horas)
9. Administración de la configuración y de cambios (4 horas)
10. Administración de la calidad (4 horas)

Introducción

No podemos negar la participación que tiene actualmente el software en muchos de los aspectos de nuestra vida y dada esta importancia resulta necesario llevar a cabo prácticas que contribuyan al desarrollo de buen software. En esta materia se mostrará al alumno la importancia de la ingeniería de software como parte de su labor profesional dentro de la informática. Al alumno se le presentarán los caminos que pueden dar solución a problemas inherentes al desarrollo de software utilizando los conocimientos generados por la industria de software. La materia se encuentra dividida en diez temas:

En el **tema uno** (Fundamentos de la ingeniería de software) se realizará una presentación al alumno del origen y características que tiene la ingeniería del software, de manera de que él sea capaz de reconocer su importancia dentro de la informática. En su conjunto será un marco de referencia que le permitirá diferenciarla de otras disciplinas relacionadas a la computación.

El **tema dos** (Software) abarca un análisis relacionado con las diferentes clasificaciones que se pueden hacer al software y a las características que podrían o no ser inherentes al software. También se abordarán los principios de la ingeniería de software y las herramientas CASE de las que se puede hacer uso.

El enfoque del **tema tres** (Administración de proyectos) partirá del valor de esta disciplina en el desarrollo de sistemas informáticos, los elementos que la integran y se hará mención de las mejores prácticas.

En el **tema cuatro** (Administración de requerimientos) se hará énfasis en la importancia que tiene el manejo de los requerimientos para que un proyecto de desarrollo de software se construya de manera correcta. Internamente se presentarán las diferentes opciones que permitan la captura, manejo, actualización y clasificación de los diferentes tipos de requerimientos que se presentan dentro de un proyecto.

Para el **tema cinco** (Análisis de software) se realizará un recorrido por esta fase del desarrollo de software, la cual se realiza desde sus diferentes perspectivas.

Dentro del **tema seis** (Diseño de software) se presentarán los diferentes recursos que pueden emplearse en el modelado de sistemas informáticos, de manera que permitan una representación clara del sistema en esta fase del desarrollo.

En el **tema siete** (Verificación y validación de software) se enseñará la diferenciación de estos dos conceptos y su lugar dentro del ciclo de desarrollo. Por otra parte se presentarán los diferentes recursos que pueden aplicarse para llevar a cabo dichas actividades.

Todas las cuestiones relacionadas posteriores a la puesta en operación del sistema se verán dentro del **tema ocho** (Mantenimiento del software).

La posibilidad de cambios durante todo el desarrollo y mantenimiento del sistema es algo común, por lo que es recomendable llevar un registro y seguimiento de los componentes que integran el sistema. Estas cuestiones se desarrollarán en el **tema nueve** (Administración de la configuración y de cambios).

Como punto final en el **tema diez** (Administración de la calidad) se desarrollará el problema de la calidad en el desarrollo de software y se presentarán algunas de las propuestas que han realizado.

TEMA 1. FUNDAMENTOS DE LA INGENIERÍA DE SOFTWARE

Objetivo particular

Al finalizar este tema el alumno será capaz de reconocer el origen y la importancia de la ingeniería del software para diferenciarla de otras disciplinas e identificar los elementos que la integran.

Temario detallado

1.1 Historia
1.2 Crisis del software
1.3 ¿Qué es la IS, ciencia, arte, disciplina o proceso?
1.4 Objetivo de IS
1.5 Las cuatro P de la IS
1.6 Proceso de IS
1.7 Relación de la IS con las demás asignaturas de la Licenciatura en Informática
1.8 Sistema y sistema informático
1.9 Metodología, técnicas y herramientas
1.10 Código de ética ACM/IEEE
1.11 Ciclo de vida de Sistemas y Modelos

Introducción

La ingeniería de software surge como una necesidad de responder a problemas relacionados al desarrollo de software. Con la finalidad de solventar los problemas detectados se han instrumentado varias propuestas que en su conjunto ayudan a reducir de manera considerable los problemas planteados. Emplear y desarrollar las mejores prácticas propuestas por la ingeniería de software nos posibilita una mejor comprensión hacia a la construcción de buen software.

1.1. Historia

La historia de la ingeniería de software se encuentra ligada a la evolución y madurez de la programación de software. Al inicio el problema radicaba en colocar una secuencia de instrucciones dentro de una computadora para que hicieran algo útil. Con la aparición de lenguajes de alto nivel y la reducción de costos de las computadoras se amplió el acceso a la programación, lo que posibilitó la construcción gradual de la profesión. No existían grandes desarrollos de software sino hasta mediados de 1960, de los cuales se puede mencionar el sistema operativo OS 360 para las computadoras IBM 360. Una vez que el software se emplea para resolver problemas más complejos es entonces cuando se hacen evidentes las dificultades para emplear las técnicas de desarrollos pequeños a desarrollos más grandes y complejos. Los problemas que se presentaban no eran los mismos que se presentaban en los inicios de la programación. Con la finalidad de resolver los problemas del desarrollo de software se optó por adquirir la perspectiva con la que las ingenierías construyen otros sistemas complejos. Si bien es cierto que el nacimiento de la ingeniería de software se da con la programación, es importante considerar otros factores que han influido también de manera importante en su crecimiento como: la disminución de los costos en el hardware y el aumento de los costos en el software, el cambio de perspectiva del software como un ciclo de vida y no solamente como código, la presencia del software en nuestra sociedad.

Claramente se puede observar que conforme el desarrollo de software se fue haciendo más complejo los problemas se hicieron más evidentes. A continuación se presenta un cuadro que ilustra la historia de la ingeniería del software por fases.

Fase	Descripción
Primera Fase. Los albores (1945-1955)	→ Programar no es una tarea diferenciada del diseño de una máquina → Uso de lenguaje máquina y ensamblador
Segunda Fase. El florecimiento (1955-1965)	→ Aparecen multitud de lenguajes → Se pensaba que era posible hacer casi todo
Tercera Fase. La crisis (1965-1970)	→ Desarrollo inacabable de grandes programas → Ineficiencia, errores, coste impredecible → Nada es posible
Cuarta Fase. Innovación conceptual (1970-1980)	→ Fundamentos de programación → Verificación de programas → Metodologías de diseño
Quinta Fase. El diseño es el problema (1980-?)	→ Entornos de programación → Especificación formal → Programación automática

Cuadro 1.1. Historia del desarrollo de software

1.2 Crisis del software

El término Ingeniería del Software fue utilizado por Fritz Bauer en la primera conferencia sobre desarrollo de software patrocinada por el Comité de Ciencia de la OTAN celebrada en Garmisch (Alemania), en octubre de 1968.

El término “crisis del software” identifica la precaria situación en la que se encontraba el desarrollo de software a comparación de otras disciplinas ante la demanda de nuevos sistemas.

Entre los problemas que acompañan al desarrollo de software podemos encontrar:

- Retrasos considerables en la planificación
- Poca productividad
- Elevadas cargas de mantenimiento
- Demandas cada vez más desfasadas con las ofertas
- Baja calidad y fiabilidad del producto
- Dependencia de los realizadores

1.3 ¿Qué es la IS, ciencia, arte, disciplina o proceso?

Es un campo de la computación que tiene que ver con la construcción de sistemas de software los cuales son grandes y complejos, son construidos por grupos de ingenieros. Resulta difícil determinarla en sólo una clasificación debido a que se cuenta con varias perspectivas de lo que es la ingeniería de software por los varios contextos en los que ha participado.

Existen dos perspectivas dentro de la ingeniería de software: una que tiene como objeto de estudio a los problemas teóricos propios de la computación, mientras que la otra perspectiva se encarga de la construcción de software.

Un programador escribe un programa de software, mientras un ingeniero de software escribe componentes de software e integra otros elementos que serán combinados, bajo una perspectiva de calidad, para solucionar un problema. La primera actividad se puede considerar como individual, mientras que la última es esencialmente una actividad de equipo. La programación es sólo una actividad dentro de la ingeniería de software.

El ingeniero del software emplea sus conocimientos multidisciplinarios para solucionar problemas y a diferencia de otras disciplinas de las ciencias de la computación emplea los conocimientos teóricos que desarrollan las otras

disciplinas para solucionar problemas que pueden no pertenecer por naturaleza a un problema de la computación. El desarrollo de software implica responsabilidades profesionales y sociales.

La ingeniería de software implica requiere disciplina para llevar a cabo las actividades inherentes al trabajo diario que le permitirán mejorar de manera constante el trabajo que realiza a nivel individual. Este esfuerzo representa un aspecto importante con miras a realizar un producto de calidad como resultado de participación de muchas personas.

La ingeniería de software emplea recursos teóricos y notaciones científicas para el desarrollo de aplicaciones, pero también utiliza modelos con la finalidad de representar aspectos de la realidad de manera sistemática. En este aspecto la ingeniería de software es una ciencia que estudia de manera sistemática los problemas del software. Como resultado de esta perspectiva son los métodos, técnicas, metodologías y herramientas que pueden utilizarse.

La ingeniería de software es un proceso si se considera que para el desarrollo de software se requieren entradas, pasos establecidos y salidas. Este esquema de trabajo facilita el entendimiento de la forma en la opera la construcción de software para solucionar un problema.

La ingeniería de software es un arte en cuanto a la identificación de elementos a utilizar dependiendo del contexto, el manejo de las relaciones humanas, establecimiento de criterios de calidad, solución a problemas éticos, interpretación y solución de problemas basados en software, etc. Todos estos problemas se caracterizan por sobrepasar las dificultades técnicas y es por eso que se emplean recursos adicionales, como la creatividad y el buen juicio, para dar una solución oportuna.

La ingeniería de software emplea recursos de diferentes áreas del conocimiento dada la complejidad que implica la construcción de software. No

es posible determinar a la ingeniería de software en una sola perspectiva por la naturaleza de su objeto de estudio a comparación de las otras ingenierías. Esto demuestra que es posible verla como ciencia, arte, disciplina o proceso, todo dependerá del punto de vista que se tome y lo cual no las hace excluyentes una de otra sino complementarias.

La ingeniería de software es:
Es la aplicación de un planteamiento sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software.[1]

1.4 Objetivos de la IS

El objetivo de la IS es desarrollar software de alta calidad. Esto implica que el software que se está desarrollando cuente con ciertas características entre las que podemos encontrar: robustez, fácil de entender, fácil de mantener, etc.

Para lograr este objetivo se pueden emplear los modelos de referencia de los procesos de desarrollo de software con calidad.

Para entender lo que se refiere a software de calidad es importante revisar las diferentes perspectivas que se tienen en cuanto a la calidad.[2]

- McCall
- CMM
- ISO 9000
- ISO/IEC 9126

[1] "IEEE Standard Glossary of Software Engineering Terminology", IEEE, Piscataway, Nueva Jersey, Std. 610.12-1990.
[2] Algunos de estos modelos serán desarrollados en el tema diez.

1.5 Las cuatro P de la IS

Resulta importante reconocer que la ingeniería de software es posible por el esfuerzo humano, definir los objetivos y alcances de lo que se va a desarrollar, poner atención en la forma en la que se colocan los métodos y herramientas y, finalmente, en elaborar un plan sólido para desarrollar el producto. Una administración efectiva de un proyecto de software se enfoca en las cuatro P's:

- Personas. Se refiere a las personas que participan dentro de un proyecto de software y sus interacciones.
- Producto. Se refiere a los elementos que se entregan y que van más allá de la aplicación de software.
- Proceso. Proporciona un marco sobre el cual se puede establecer un plan claro para desarrollar software. Mediante él los proyectos producen productos de manera efectiva debido a que las actividades propuestas se pueden ajustar a las características del proyecto.
- Proyecto. Su objetivo es realizar un producto de software.

El resultado final de un proyecto de software es un producto, donde intervienen personas a través de un proceso de desarrollo de software que guía los esfuerzos de las personas implicadas en el proyecto.

1.6. Proceso de IS

Es un conjunto de actividades técnicas y administrativas realizadas durante la adquisición, desarrollo, mantenimiento y retiro de software[3].

Existe un acercamiento al proceso de ingeniería del software desde dos puntos de vista. El primer nivel tiene que ver con las cuestiones técnicas y administrativas dentro de los procesos del ciclo de vida durante la adquisición, desarrollo, mantenimiento y retiro del software. La segunda parte es un meta-

[3] IEEE. "Guide to the Software Engineering Body of Knowledge". IEEE Computer Society SWEBOK, 2004, p. 9-1.

nivel que comprende las definiciones, implementación, evaluación, medición, administración, cambio y mejora de los procesos mismos del ciclo de vida de software.

El término “proceso de ingeniería del software” se puede interpretar de varias formas que pueden llegar a confundir:

- La adición del artículo “El” a procesos de ingeniería del software puede dar a entender que sólo existe una forma para realizar las actividades, cuando en realidad existen varios procesos involucrados.
- Otro significado tiene que ver con la discusión general de los procesos relacionados a la ingeniería del software.
- El tercer significado tiene que ver con el conjunto actual de actividades desempañadas dentro de una organización, las cuales pueden ser vistas como un proceso dentro de una organización.

El proceso de ingeniería de software es importante no sólo para organizaciones grandes ya que todas las actividades han sido desempeñadas exitosamente por organizaciones pequeñas o individuales.

1.7. Relación de la IS con las demás asignaturas de la Licenciatura en Informática

Existe una relación entre las materias de la licenciatura en Informática y la ingeniería de software es que ambas se interesan por la solución de problemas a través del empleo de software. El complemento que realiza la ingeniería de software es la adición de recursos (modelos, técnicas y herramientas) para construir una solución de software con calidad.

La perspectiva de calidad es compatible con las materias abordadas por la informática y debe de verse como una adición que añade valor y permite ampliar la visión de sólo automatizar información.

1.8. Sistema y sistema informático

Por una parte: “Un sistema puede ser definido como un complejo de elementos interactuantes”.[4]

Sistema. Colección de componentes organizados para cumplir una función específica o un conjunto de funciones.[5]

Un sistema informático es un conjunto de aplicaciones de software que se ejecutan para proporcionar un resultado. Es un conjunto de software, hardware y personas.

Los sistemas de información tienen muchas cosas en común, la mayoría de ellos están formados por:

- Personas: es el componente esencial en cualquier sistema de información, producen y utilizan la información de sus actividades diarias para decidir lo que se debe hacer. Las decisiones pueden ser rutinarias o complejas.
- Procedimientos, los sistemas de información deben soportar diversas clases de actividades del usuario, por eso han de establecerse procedimientos que aseguren que los datos correctos llegan a las personas adecuadas en su momento justo.
- Equipo, es decir, los ordenadores con sus programas y todos los dispositivos necesarios.

1.9. Metodología, técnicas y herramientas

Para solucionar un problema en la ingeniería de software es posible recurrir a metodologías, técnicas y herramientas. La relación estrecha entre estos tres elementos hace posible mejorar la calidad del software.

---

[4] Bertalanffy, Ludwig von. *Teoría general de los sistemas*. p. 56.
[5] IEEE 610.12-1990.

Metodología

Propone un acercamiento para resolver una problemática empleando recursos organizados de manera particular. Es una perspectiva que contiene fases, procedimientos, reglas, técnicas, herramientas, documentación y aspectos de formación para los desarrolladores de sistemas de información.

Técnica

Método o procedimiento (con referencia a detalles prácticos o formales), o forma de usar habilidades básicas, en la representación de un trabajo artístico o realizando una operación mecánica o científica.[6]

Es el proceso que permite asegurar que algún aspecto de la aplicación o unidad funciona apropiadamente.[7]

Puede verse también como un procedimiento formal que produce algún resultado. Especialización en un tema en una perspectiva teórico-práctica. Describe la manera general la forma en la que pueden hacerse las cosas.

Herramienta

Es un instrumento que permite realizar las cosas de mejor forma. Es el medio sobre el cual se apoyarán las tareas con el fin de realizar una mejora en alguno de sus aspectos. Suelen ser los elementos más abundantes y puede existir más de uno que sirva de apoyo a una misma tarea. La selección de una herramienta deberá de ser el resultado de una estrategia que apuntale la dirección y los elementos necesarios para lograr los objetivos.

Es el vehículo para desempeñar un proceso de pruebas.[8]

---

[6] Hutcheson, Marnie L. *Software Testing Fundamentals: Methods and Metrics*. Indianapolis, Wiley, 2003.
[7] PERRY, William E. Effective methods for software testing. New York, Wiley, 1995. P. 361.
[8] PERRY, William E. Effective methods for software testing. New York, Wiley, 1995. P. 361.

1.10. Código de ética ACM/IEEE[9]

La ACM (*Association for Computing Machinery*) aprobó el código en noviembre de 1998 y la IEEE (*Institute of Electrical and Electronics Engineers*) *Computer Society,* en diciembre del mismo año.

> Código de Ética y Práctica Profesional 5.2
>
> PREÁMBULO
>
> La versión corta del código resume las aspiraciones a un alto nivel de abstracción; las cláusulas que se incluyen en la versión completa proporcionan ejemplos y detalles acerca de cómo estas aspiraciones modifican nuestra manera de actuar como profesionales de la ingeniería de software. Sin las aspiraciones los detalles pueden convertirse en tediosos y legalistas; sin los detalles las aspiraciones pueden convertirse en altisonantes pero vacías; juntas, las aspiraciones y los detalles forman un código cohesivo.
>
> Los ingenieros de software deberán comprometerse a convertir el análisis, especificación, diseño, implementación, pruebas y mantenimiento de software en una profesión respetada y benéfica. De acuerdo a su compromiso con la salud, seguridad y bienestar social, los ingenieros de software deberán sujetarse a los ocho principios siguientes:
>
> *Sociedad*. Los ingenieros de software actuarán en forma congruente con el interés social.
>
> *Cliente y empresario*. Los ingenieros de software actuarán de manera que se concilien los mejores intereses de sus clientes y empresarios, congruentemente con el interés social.
>
> *Producto.* Los ingenieros de software asegurarán que sus productos y modificaciones correspondientes cumplen los estándares profesionales más altos posibles.

[9] Esta es una versión corta del código de ética. La versión completa se puede encontrar en: http://www.acm.org/about/se-code#full, consultado el 18/05/09.

Juicio. Los ingenieros de software mantendrán integridad e independencia en su juicio profesional.

Administración. Los ingenieros de software gerentes y líderes promoverán y se suscribirán a un enfoque ético en la administración del desarrollo y mantenimiento de software.

Profesión. Los ingenieros de software incrementarán la integridad y reputación de la profesión congruentemente con el interés social.

Colegas. Los ingenieros de software apoyarán y serán justos con sus colegas.

Personal. Los ingenieros de software participarán toda su vida en el aprendizaje relacionado con la práctica de su profesión y promoverán un enfoque ético en la práctica de la profesión.

1.11. Ciclo de vida de Sistemas y Modelos

Ciclo de vida. Evolución de un sistema, producto, servicio, proyecto u otra entidad realizada por el hombre desde su concepción hasta el retiro.

Modelo de ciclo de vida. Marco de referencia de procesos y actividades relacionados al ciclo de vida que puede estar organizado en fases, el cual actúa como una referencia común para la comunicación y entendimiento.

Procesos del ciclo de vida del software. Marco de referencia que contiene procesos, actividades y tareas que serán aplicadas durante la adquisición de un producto de software o servicio y durante el abastecimiento, desarrollo, operación, mantenimiento y disposición de productos de software.

Estas son tres definiciones extraídas del estándar ISO/IEC 12207[10] son planteamientos generales, de manera específica se encuentran varios modelos que presentan las diferentes formas en las que es posible llevar a cabo la

[10] ISO/IEC. “Systems and software engineering – Software Life Cycle Processes”. IEEE std 12207:2008, 2008.

creación de un producto de software, también son conocidos como modelos de proceso de software. A continuación se presentan algunos de ellos.

Modelo de cascada. Contiene las actividades básicas que se pueden encontrar en la mayoría de los modelos y se encuentra dividido generalmente en fases: especificación de requerimientos, diseño de software, implementación, pruebas y mantenimiento.

Modelo basados en componentes. Su interés es en componentes reutilizables que ya existen y se enfoca principalmente a integrarlos y no tanto a un desarrollo completamente nuevo.

Modelo evolutivo. Son modelos que se adaptan a la evolución que sufren los requisitos del sistema en función del tiempo. Algunas actividades se entrelazan de manera que manera que se realice un desarrollo rápido que constantemente que se refina en varias versiones con las peticiones del cliente.

Modelos de métodos formales. Permiten especificar, desarrollar y verificar un sistema generando un modelo formal matemático.

Proceso unificado[11]. Es considerado un modelo híbrido porque integra varios de los elementos de los procesos genéricos. Integra una perspectiva dinámica con las fases iterativas, una perspectiva estática con los *workflows* y una perspectiva práctica con las buenas prácticas.

Métodos ágiles. Son modelos enfocados a desarrollos iterativos e incrementales. Requieren de la participación constante del usuario durante el proceso de desarrollo. Eliminan cualquier sobrecarga de trabajo en el desarrollo con la finalidad de hacer desarrollos más rápidos.

[11] Este modelo también conocido como RUP, es desarrollado más adelante en el tema cuatro.

Bibliografía del tema 1

Bertalanffy, Ludwig von. *Teoría general de los sistemas*. México, Fondo de cultura económica, 2003, 311 pp.

Hutcheson, Marnie L. *Software Testing Fundamentals: Methods and Metrics*. Indianapolis, Wiley, 2003. 408 pp.

Perry, William E. *Effective methods for software testing*. New York, Wiley, 1995, 1008 pp.

PFLEEGER, Shari Lawrence. *Ingeniería de software, Teoría y práctica*. México, Prentice Hall, 2002, 759 pp.

IEEE. "*IEEE Software Engineering Standards Collection* 1999 Edition. Volume 2: Process Standards". IEEE Computer Society Press, 1999.

IEEE. "Standard Glossary of Software Engineering Terminology". IEEE std 610.12-1990, 1990.

ISO/IEC. "Systems and software engineering – Software Life Cycle Processes". IEEE std 12207:2008, 2008.

Páginas de referencia

http://www.acm.org/about/se-code/

http://www.swebok.org/

Actividades de aprendizaje

A.1.1. Leer la versión completa del código de ética de la ACM/IEEE. La liga se encuentra en las referencias de este tema. Redactar un reporte de lectura con la opinión sobre las dificultades para la aplicación del código de ética de la ACM/IEEE en el ejercicio profesional.

A.1.2. Realizar un cuadro comparativo en la que se identifiquen las ventajas y desventajas de los modelos de ciclo de vida.

A.1.3. Investigar y elaborar un cuadro comparativo que permita identificar las características de las diferentes disciplinas computacionales: ciencias de la computación, ingeniería en computación, informática, ingeniería de software.

A.1.4. Realizar el análisis sobre los retos de la ingeniería de software con respecto a las demás ingenierías.

Cuestionario de autoevaluación

1. ¿Qué es la ingeniería de software?
2. ¿Cuándo y en qué contexto surge el término “crisis del software”?
3. ¿Cuál es el objetivo de la ingeniería de software?
4. ¿Cuál es la relación de la ingeniería de software y la informática?
5. ¿Cómo se diferencia un sistema de un sistema informático?
6. ¿Sobre qué elementos se puede apoyar la ingeniería de software para mejorar la calidad del software?
7. ¿Cuáles son los temas que trata el código de ética de la ACM/IEEE?
8. ¿Cuál fue el motivo para crear la ingeniería de software?
9. Mencione cuatro modelos de ciclo de vida de software.
10. ¿Qué es el ciclo de vida de software?

Examen de autoevaluación

I. Instrucciones. Responda cada una de las preguntas seleccionando una de las opciones que se presentan.

1. ¿En qué conferencia se dio a conocer el término de ingeniería de software?

a) OTAN
b) ONU
c) GNU
d) TLC

2. En qué fase, dentro de la historia de la IS, aparece el término crisis de software

a) Tercera
b) Primera
c) Novena
d) Ninguna de las anteriores

3. Según la definición de “crisis de software”, ¿cuál no sería una causa del origen de dicho término?

a) Poca productividad
b) Retrasos
c) Facilidad de uso
d) Baja calidad

4. Es uno de los principios del código de ética ACM/IEEE

a) Lealtad
b) Honor
c) Juicio
d) Amistad

5. Son los elementos comunes dentro de un sistema de información

a) Personas, procedimientos y equipos
b) Código de ética, software y manuales
c) Software y documentación
d) Personas y software

II. Instrucciones. Relacione las dos columnas colocando la letra de la columna derecha dentro de los paréntesis de la columna izquierda.

6.	( ) Se refiere a los elementos que se entregan y que van más allá de la aplicación de software.	a) Objetivo de la ingeniería de software
7.	( ) Aproximación lógica a la adquisición, el suministro, el desarrollo, la explotación y el mantenimiento del software	b) Metodología
8.	( ) Conjunto de filosofías, fases, procedimientos, reglas, técnicas, herramientas, documentación y aspectos de formación para los desarrolladores de sistemas de información.	c) Producto
9.	( ) Formados por un conjunto de fases o actividades en las que no tienen en cuenta la naturaleza evolutiva del software.	d) Ciclo de vida del software
10.	( ) Es desarrollar software de alta calidad.	e) Modelos tradicionales

TEMA 2. SOFTWARE

Objetivo particular

Al finalizar el tema el alumno obtendrá los conceptos que identifican y clasifican al software. También conocerá los principios que propone la ingeniería del software para desarrollar software de calidad y finalmente identificará las herramientas CASE (*Computer-Aided Software Engineering*) de las cuales puede hacer uso.

Temario detallado

2.1 Concepto
2.2 Clasificación
2.3 Características
2.4 Principios de la IS
2.5 Herramientas CASE

Introducción

El software es resultado final del esfuerzo que se realiza durante el proceso de desarrollo. Es importante reconocer la naturaleza del producto, de ello depende, en gran parte, para que se identifiquen los alcances y oportunidades que se pueden realizar con y en él. El desarrollo de software no es una actividad trivial; por eso existen esfuerzos que sirven de guía para el desarrollo de software de calidad. La versatilidad del software le permite que sea aplicado como apoyo en diferentes ámbitos dentro de la actividad humana.

2.1 Concepto

Software es un programa de computadora que desempeña un conjunto de funciones.

Son los programas de ordenador, los procedimientos y, posiblemente, la documentación asociada y los datos relativos a la operación del sistema informático.

El software es un conjunto de algoritmos ejecutados en una computadora. Un algoritmo es una secuencia de instrucciones lógicas y matemáticas con un objetivo específico.

Software es concebido también como:

- Las instrucciones (programas de computadora) que cuando son ejecutas proporcionan las funciones y desempeño deseados.
- Estructuras de datos que posibilitan manipular adecuadamente la información.
- Documentos que describen la operación y uso de los programas.

2.2 Clasificación

La evolución de la tecnología ha posibilitado que se desarrolle software cada vez más acorde con las actividades del ser humano. Dada la diversidad de escenarios en los que se lleva a cabo la aplicación del software una clasificación del software nos permite hacer una distinción de acuerdo a las diferentes perspectivas en las que se emplean:

- Software por función o de aplicación. Es el que se emplean de manera tal que la computadora llegue a ser una herramienta útil para el usuario final en alguna de las actividades que realiza.
- Software de utilería. Permite dar mantenimiento al equipo de cómputo.
- Software de entretenimiento. Es aquel que hace posible utiliza al equipo de cómputo como medio de entretenimiento.
- Software integrado. Reside sólo en la memoria y es utilizado para controlar productos y sistemas. Puede realizar funciones limitadas.

- Software para lenguaje de programación. Es el software enfocado a un lenguaje de programación generando todo un ambiente de desarrollo que facilite la generación y ejecución de código.
- Por sistema operativo. Se refiere a todo aquel software que se encarga de la administración de los recursos de un equipo de cómputo.
- Software personalizado. Permite personalizar las funciones según los gustos o tipo de actividades que realiza con en el equipo de cómputo.
- Software de negocios. Son todas aquellas que se utilizan para soportar las operaciones más importantes de un negocio en funciones como almacenamiento, análisis y procesamiento.

2.3 Características

En el software podemos identificar un conjunto de características que permiten identificarlo.

- No tiene existencia física.
- Se desarrolla, no es manufacturado.
- Es una herramienta, un medio.
- La existencia del software sólo se da con la ayuda del hardware o computadora.
- Su entorno cambia constantemente.
- Depende de los requerimientos.
- Por lo general se construye a la medida.
- El desarrollo de software es un trabajo intelectual.
- Algunas partes se pueden utilizar en varios proyectos.
- El software no se estropea, ni desgasta, sólo se hace obsoleto.
- En desarrollos grandes se dificulta su control.
- Es maleable.

Por otra parte, desde la perspectiva de los modelos de calidad, el software debe de tener ciertas características que permitan determinar su calidad, de ellas las más comunes se pueden encontrar las siguientes:
Correcto. Grado con el cual un sistema o componente se encuentra libre de defectos en sus especificaciones, diseño e implementación. Grado en el cual el software, documentación u otros elementos cumplen con los requerimientos especificados.

Confiabilidad. Capacidad de un sistema o componente para desempeñar sus funciones requeridas bajo condiciones establecidas durante un periodo de tiempo específico.

Flexibilidad. Es la facilidad con la cual un sistema o componente puede ser modificado para ser utilizado en aplicaciones o ambientes diferentes a los que inicialmente diseñados.

Reusabilidad. Es el grado con el cual un módulo de software u otro producto pueden utilizarse en más de un programa de computadora o sistema de software.

Portabilidad. La facilidad con la cual un sistema o componente puede transferirse entre varios ambientes de hardware o software.

Comparabilidad. Capacidad de dos o más sistemas o componentes para desempeñar sus funciones especificadas mientras comparten el mismo ambiente de hardware o software. Capacidad de dos o más sistemas o componentes para intercambiar información.

Integridad. Grado con el cual un sistema un sistema o componente previene accesos no autorizados o modificaciones a programas de computadora o datos.

Funcionalidad. Capacidad del software de contener las funciones acordadas y las necesidades implicadas cuando el software es utilizado bajo condiciones específicas.
Fiabilidad. Capacidad del software para mantener un nivel específico de desempeño cuando es utilizado en condiciones específicas.

Facilidad de uso. Capacidad del software para ser entendido, enseñable, usado y atractivo al usuario cuando es utilizado en condiciones específicas.

Eficiencia. La capacidad del software para proporcionar un desempeño adecuado, relacionado a la cantidad de recursos utilizados en ciertas condiciones.

Facilidad de mantenimiento. La capacidad del software para ser modificado. Las modificaciones pueden incluir las correcciones, mejoras o adaptaciones a los cambio del entorno y en los requerimientos y especificaciones funcionales.
Portabilidad. Capacidad del software para ser transferido de un ambiente a otro.

Robustez. El software es robusto cuando se comporta razonablemente en circunstancias que no fueron anticipadas en los requerimientos.

Desempeño. El software es eficiente si utiliza los recursos de la computadora de manera económica.

Verificabilidad. El software es verificable cuando sus propiedades pueden ser verificadas fácilmente.

Reparable. El software tiene esta cualidad cuando permite reparar sus defectos con una limitada cantidad de trabajo.

Adaptabilidad. El software evoluciona cuando es modificado en el tiempo para proveer nuevas funciones o para modificar las que tiene.

Interoperabilidad. Es la habilidad del software para coexistir con otros sistemas. Estas últimas características permiten evaluar al software y algunas de ellas (dependiendo del modelo) se llegan a dividir en sub-características y atributos. Los cuales pueden interactuar entre ellos. Con el tiempo, estas características han sido incorporadas por los modelos de calidad, lo cual no quiere decir que sean todas o que deban de utilizarse por completo cada vez que se realiza un desarrollo de software, ya que en algunos casos se dará preferencia a unas características sobre otras. Cada desarrollo tiene sus propias funciones y características e identificarlas es una habilidad que debe formarse por quienes se encargan de desarrollar software.

2.4 Principios de la IS

Los principios en la IS son aquellos que actúan como puntos de referencia para el desarrollo de software exitoso. Por una parte se encuentra el proceso de desarrollo y por la otra se hace referencia al producto, haciendo notar que ambos se encuentran relacionados y tienen la misma importancia.

Los principios son planteados de manera general con la finalidad de que puedan ser empleados, pero no al grado de guiar el desarrollo de software. Para que sean aplicados es necesario apoyarse de métodos y técnicas que permitan adoptar los principios al proceso o al producto.

Los principios son la base sobre la cual se posicionan las técnicas, herramientas, metodologías y métodos. La selección de principios y técnicas se encuentra determinada por las metas de calidad.

Los principios que se presentan a continuación no son los únicos y se aplican principalmente al proceso de desarrollo de software.

Rigor y formalidad. El rigor es lo que permite que se puedan construir productos confiables, controlar costos y aumentar la confianza en su confiabilidad. Pese a lo que se propone el rigor es un aspecto intuitivo que no puede ser definido de forma rigurosa. Dentro de los grados que existen en rigor, el más alto es la formalidad, este requiere que sea dirigido y evaluado por leyes matemáticas. La formalidad implica rigor, pero no necesariamente a la inversa. La formalidad es la base de la mecanización del proceso.

Separación de intereses. Posibilita la interacción con diferentes aspectos individuales de un problema, logrando la atención de cada una por separado. Algunas de las decisiones relacionadas a un proyecto de software deberán de tomarse por separado evitando relacionarlas una con otra.

Modularidad. Un sistema complejo de software que puede ser dividido en piezas o módulos permite separar los diferentes intereses cuando se abordan cuestiones que tienen que ver con los detalles de cada; además, se pueden estudiar todos los módulos en relación para conformar un sistema. La cohesión permite que exista una fuerte relación en cada uno de los módulos y la integración se encarga de la relación entre módulos.

Abstracción. Es el proceso en el que se identifican los aspectos importantes de un fenómeno. En cada abstracción se eliminan los detalles y se genera una vista del mundo enfocada a un propósito específico. Un ejemplo de abstracción es cuando se busca representar el funcionamiento de algunos fenómenos a través de modelos.

Anticipación al cambio. El software cambia constantemente por diferentes circunstancias, la habilidad del software de anticiparse a los cambios tiene que ver con el cómo y cuándo se realizarán los cambios y una vez que son identificados se deberá proceder de manera que los cambios sean realizados

fácilmente. La anticipación a los cambios requiere de un control sobre todos los elementos del desarrollo.

Generalidad. La generalidad es un principio fundamental si se tiene como objetivo el desarrollo de herramientas generales o paquetes para el mercado, ya que para ser exitosas deberán cubrir las necesidades de distintas personas. Estos productos de propósito general, *off-the-shelf,* por ejemplo los procesadores de texto, representan una tendencia general en el software; para cada área específica de aplicación existen paquetes generales que proveen soluciones estándares a problemas comunes que evitan de esta forma una solución especializada.

Acrecentamiento. Caracteriza un proceso en el que se procede de manera gradual o incremental. Las metas se alcanzan mediante aproximaciones sucesivas en el que cada aproximación es un incremento de la anterior. Una manera de aplicar este principio es identificando elementos útiles de una aplicación para desarrollarlos y entregarlos al cliente para obtener una retroalimentación más rápida. Este principio se fundamenta diciendo que en la práctica es poco probable que se cuenten con todos los requerimientos antes de que se inicie el desarrollo de la aplicación y que estos van surgiendo conforme se van presentando los avances de la aplicación, por esto entre más se vayan presentando partes de la aplicación y al obtener la retroalimentación del cliente será más fácil incorporar los cambios al proyecto.

2.5 Herramientas CASE

Un CASE (*Computer-Aided Software Engineering*) es una herramienta que permite organizar y controlar el desarrollo de software. Se utiliza especialmente en desarrollos de software largos que implican una mayor complejidad por la cantidad de componentes y personas involucradas. Este tipo de herramientas soporta los métodos y conceptos de programación.

Entre las ventajas que se pueden encontrar al utilizar las herramientas CASE podemos encontrar:

- Generación de una vista común entre todos los integrantes para cada una de las fases del desarrollo de software.
- Repositorio de los entregables en cada una de las fases.
- Organización
- Reducción de costos
- Aseguramiento de la calidad.
- Registro y presentación el avance del proyecto.
- Reducción del tiempo de procesamiento para el análisis.
- Automatización de algunas actividades dentro del proceso de desarrollo.
- Incremento de la productividad y la confiabilidad en el proceso de producción.

A continuación se presenta una clasificación de las herramientas.
Medio de interacción. La interacción del usuario con las herramientas ha evolucionado de manera que éstas sean más intuitivas y fáciles de usar para reducir de esta forma los errores. Un aspecto importante a considerar en la interacción con las computadoras es la sintaxis en los lenguajes de programación, ya que permite la adopción del lenguaje y reduciendo los errores en su uso.

Nivel de formalidad. En el desarrollo de software se ven involucrados varios tipos de documentos. En cada uno de ellos se puede definir en algún grado la sintaxis y la semántica. Mientras un compilador se encuentra definido de principio con una sintaxis y semántica formalmente definidas, el editor puede emplearse para cualquier lenguaje.

Dependencia en la fase del ciclo de vida. Existen herramientas de software que se encuentran ligadas a actividades específicas dentro del ciclo de vida del

software. La adopción de herramientas permite una transición entre las fases según sea el modelo que se haya seleccionado.

Dependencia en la aplicación o método. La selección de técnicas, métodos, modelo de ciclo de vida y cualidades, pueden verse influidas por las aplicaciones. Muchos métodos pueden ser más efectivos empleando una adecuada herramienta de soporte. Las herramientas pueden mejorar el método pero no pueden sustituirlo. De igual forma la calidad en el software no se puede mejorar sólo utilizando herramientas sin comprender su significado dentro de la metodología.

Grado de estandarización. Por una parte la estandarización mejora su aplicabilidad pero por otra parte detiene su evolución. Sin embargo, la estabilización de un método o lenguaje garantiza el soporte que se le da y también posibilita la opción de seguir utilizándolo en varios proyectos a la gente que lo utiliza.

Dependencia del lenguaje. Existen herramientas que se encuentran ligadas a un lenguaje y existen otras que soportan varios. Dentro de las primeras se encuentran los compiladores, que son especializados en un lenguaje de programación. Mientras que un procesador de palabras puede ser utilizado para editar cualquier lenguaje de programación.

Herramientas estáticas contra herramientas dinámicas. Hay herramientas que no requieren de la ejecución del objeto que emplea. Son utilizadas para crear, modificar, verificar la consistencia con respecto a una regla, medir ciertas propiedades estáticas o para detectar algunas restricciones. A estas herramientas se les denominan estáticas. Las herramientas dinámicas son las que requieren de la ejecución del objeto.

Herramientas de desarrollo contra componentes de producto final. Algunas herramientas se emplean para el desarrollo del producto final sin llegar a ser

parte de él. Mientras que otras herramientas de componentes de software pueden ser incluidas y llegar a ser parte del producto final.

Bibliografía del tema 2

Ghezzi, Carlo. *Fundamentals of Software Engineering*. Englewood Cliffs, Prentice-Hall, 2003.

ISO/IEC. “Information Technology – Software Product Quality”. International Standard 9126:2000(E), 2000.

Actividades de aprendizaje

A.2.1. Elabora un cuadro en el que se presenten las características comunes en cada una de las clasificaciones del software.

A.2.2. Investiga dos modelos de calidad y registra las características de calidad que propone cada una.

A.2.3. Realiza una lista en la que se detalle los beneficios de las herramientas CASE en desarrollos pequeños.

A.2.4. Realiza una investigación y encuentra otro de los principios de IS diferente a los expuestos en este tema. Registra la justificación y fuente de la información.

A.2.5. Realiza una investigación y registra dos ejemplos de herramienta CASE con sus respectivas características.

Cuestionario de autoevaluación

1. ¿Cuál es la definición de software que consideras más adecuada?, justifica tu respuesta.
2. ¿Por qué resulta difícil obtener una definición de software?
3. ¿Cuál es la finalidad de los principios de IS?
4. ¿Menciona tres características del software?
5. ¿Cuál es el significado de la palabra CASE?
6. ¿Menciona los principios de IS que se vieron en este tema?
7. ¿Cuál es la finalidad de las herramientas CASE?
8. ¿En qué clasificación de software entrarían las herramientas CASE?
9. ¿A qué nivel se encuentran planteados los principios de IS?
10. ¿En qué tipo de desarrollos de software son comúnmente utilizadas las herramientas CASE?

Examen de autoevaluación

Instrucciones. Relaciona las dos columnas colocando la letra de la columna derecha dentro de los paréntesis de la columna izquierda.

1.	( ) Son puntos de referencia generales para desarrollar software de calidad.	a) Nivel de formalidad
2.	( ) Es una herramienta que se puede utilizar durante el proceso de desarrollo de software.	b) Eficiencia
3.	( ) Es software que define el grado de sintaxis o semántica.	c) Definición de software
4.	( ) No requiere de la ejecución del objeto para realizar sus funciones.	d) Principios IS
5.	( ) Automatiza algunas actividades dentro del proceso de desarrollo.	e) CASE
6.	( ) Capacidad de proporcionar desempeño adecuado con respecto a los recursos utilizados en un momento determinado.	f) Herramientas estáticas
7.	( ) Conjunto de algoritmos ejecutados en una computadora.	g) Ventaja CASE
8.	( ) Software que tiene la finalidad de que la computadora se convierta en una herramienta para algunas de las actividades que realiza el usuario final.	h) Software por función o de aplicación
9.	( ) Software ligado a uno o varios lenguajes.	i) Funcionalidad
10.	( ) Capacidad para desempeñar las funciones acordadas en condiciones específicas	j) Dependencia del lenguaje

TEMA 3. ADMINISTRACIÓN DE PROYECTOS

Objetivo particular

Al finalizar este tema el alumno conocerá la importancia y características de un proyecto dentro del desarrollo de software y también conocerá una de las propuestas existentes para mejorar el trabajo de manera individual.

Temario detallado

3.1 ¿Qué es un proyecto?
3.2 Ciclo de vida del proyecto
3.3 Oficina de proyectos
3.4 PSP

Introducción

La forma de trabajo dentro del desarrollo se identifica con un proyecto, por eso resulta importante conocer las características de un proyecto junto con los diferentes recursos que se encuentran disponibles para que un proyecto alcance los objetivos establecidos.

3.1 ¿Qué es un proyecto?

Un proyecto es un esfuerzo temporal emprendido para crear un producto, servicio o resultado único. Esta estructura de trabajo es ampliamente utilizada y no es de uso exclusivo del desarrollo de software. Sus características nos permiten identificarlo y es por eso que se presentan a continuación.

Temporal. Se refiere a que cada proyecto tiene un inicio y un final definidos. El final se alcanza una vez que se cumplen los objetivos. La duración del tiempo proyecto es finito.

La naturaleza temporal de los proyectos se relaciona también con cuestiones como:

La oportunidad o ventana del mercado tiene un lapso de duración determinado. El equipo trabaja como unidad durante el periodo que dura el proyecto para posteriormente reasignar a los recursos una vez que el proyecto ha finalizado.

Productos, servicios o resultados únicos. Un proyecto genera entregables únicos.

Un producto o artefacto realizado es cuantificable y puede ser él mismo un elemento final o parte de otro elemento. El hecho de que los elementos se encuentren de manera repetida no cambia que el trabajo de un proyecto sea único.

Confección progresiva. Significa que el desarrollo se realiza por pasos y en su continuación se realiza por incrementos. La elaboración progresiva del proyecto necesita ser coordinada con el alcance.

Un proyecto se diferencia del trabajo operativo en que este último se mantiene funcionando de manera repetida, mientras que un proyecto se opera de manera temporal y única. Otra diferencia se encuentra en que los objetivos, los proyectos están sujetos a los objetivos planteados y una vez alcanzado dichos objetivos el proyecto finaliza, mientras que en la operación los objetivos se encuentran en relación con el sostén del negocio, los cuales pueden cambiar o definir nuevos, pero el trabajo continúa.

Los proyectos son empleados principalmente como medio para alcanzar un plan estratégico de una organización.

Un proyecto puede ser el resultado de alguno de los siguientes escenarios:

- Demanda del mercado
- Necesidad organizacional
- Solicitud de un cliente
- Ventaja tecnológica
- Disposición legal

La administración de proyectos es la aplicación de conocimientos, herramientas, habilidades y técnicas a las actividades del proyecto para cumplir con los requerimientos del mismo. Los procesos de la administración de proyectos comprenden cinco grupos: inicio, planificación, ejecución, monitoreo y control y cierre. Estos grupos no representan las fases de un proyecto, sino que es una forma de agrupar las áreas de conocimiento que se pueden utilizar.

En un proyecto se debe de realizar el balanceo entre la calidad, el alcance, el tiempo y el costo. En este esquema comúnmente se dice que de los cuatro elementos mencionados anteriormente la calidad se ve influenciada por los otros tres.

Dentro de algunas de las causas por las cuales fracasan los proyectos se encuentran:

1. El usuario no sabe lo que quiere (¿o nosotros no lo entendemos?).
2. Requerimientos y especificaciones incompletos.
3. Cambio en los requerimientos y especificaciones.
4. Carencia de participación por parte del usuario.
5. No existe documentación.
6. Falta de una metodología para la gestión de requisitos.

3.2 Ciclo de vida del proyecto

Se puede dividir los proyectos en fases con la finalidad de tener un mejor control con las apropiadas conexiones a las operaciones que realiza la organización. El conjunto de fases es conocido como ciclo de vida del proyecto. Las organizaciones identifican los ciclos de vida que aplican a todos sus proyectos.

Un ciclo de vida define las fases que se emplearán desde el inicio hasta el final del proyecto. La definición del ciclo de vida del proyecto le permite al administrador de proyecto en dónde colocar cada una de las fases. La transición entre fases involucra generalmente una transferencia técnica. Los productos de una fase son revisados antes de comenzar con las actividades de la fase que continúa. Aunque no es extraño que una fase inicie sus actividades sin haber aprobado previamente los entregables de la fase anterior cuando los riesgos involucrados son aceptables.

Cada organización puede establecer sólo un ciclo de vida para todos los proyectos o puede dar la libertad al administrador de proyecto que elija el ciclo de vida.

En los ciclos de vida de proyectos generalmente se define:

- Las actividades técnicas que se llevarán a cabo en cada una de las fases.
- La fecha de entrega de los productos de cada fase y la manera en la que serán revisados, verificados y validados.
- Los involucrados en cada fase.
- El control y mejora de cada fase.

A un nivel muy detallado del ciclo de vida se incluyen los formatos, políticas, listas de verificación, etc.

Algunas características que se pueden encontrar en los ciclos de vida de proyectos son:

- Las fases comúnmente son secuenciales y son definidas por la transferencia de información o por algún componente.
- El costo y el personal involucrado son bajos al inicio, en las fases intermedias aumentan para disminuir al final.
- El nivel de incertidumbre se encuentra en la fase inicial; va desapareciendo conforme se avanza en el proyecto.
- La influencia de los clientes en las características y costos del proyecto se presentan al principio para disminuir durante el desarrollo. Mientras que los costos asociados a los cambios y las correcciones aumentan conforme se avanza en el proyecto.

La realización y aprobación de productos es lo que identifica a una fase. Un producto es algo que puede ser medido y verificado. Los productos pueden ser tanto propios del mismo proceso de administración del proyecto como un elemento del producto final. Los productos y las fases son parte de un proceso secuencial diseñado para asegurar el control del proyecto para logar el producto o servicio.

Una fase generalmente finaliza cuando el trabajo y los productos son revisados para determinar su aceptación. Completar una fase no incluye la autorización a la siguiente fase. Existe una dependencia entre cada fase que indica los resultados que son permitidos y esperados de uno a otro. Con una revisión final de la fase se obtiene la autorización para cerrar una fase y abrir la siguiente.

3.3 Oficina de proyectos

También se conoce como PMO (*Project Management Office*), es una unidad organizacional que centraliza y organiza la administración de proyectos bajo su dominio. Supervisa la administración de proyectos, programas o la combinación

de ambos. Algunos PMO coordinan y administran proyectos relacionados. En muchas organizaciones esos proyectos se encuentran realmente agrupados u organizados de manera que el PMO pueda administrar y coordinar esos proyectos.

El PMO se enfoca en la planificación coordinada, priorización y ejecución de proyectos y subproyectos que se encuentran ligados a los objetivos de negocio de la organización o clientes.

Algunas de las actividades que realiza el PMO son:

- Coordinar todos los recursos de los proyectos que tiene a su cargo.
- Identificar y desarrollar la metodología para la administración de proyectos, y con ello todas las técnicas, estándares, políticas, etc.
- Centralizar la administración de la configuración de todos los proyectos.
- Centralizar el repositorio y la administración de todos los riesgos, tanto específicos como compartidos entre los proyectos.
- Centralizar la administración y operación de las herramientas utilizadas para los proyectos.
- Coordinar de manera centralizada la comunicación entre los proyectos.
- Coordinar todos los estándares de calidad entre los estándares internos y los organismos de estandarización.
- Alinear los objetivos de los proyectos a los de la organización.

3.4 PSP

PSP (*Personal Software Process*)

La calidad del software comienza por los individuos. El propósito es mejorar al ingeniero de software. Los elementos se presentan de manera que se pueda hacer uso de ellos según sean requeridos, es por ellos que se plantean las técnicas que posteriormente pueden ser aplicadas a métodos y herramientas. Los recursos son para quien diseña, desarrolla, documenta o realiza

mantenimiento de software. Es una sugerencia sobre lo que se tiene que mejorar y cómo hacerlo.

Este proceso incluye una administración efectiva de defectos y métodos de planificación, seguimiento y análisis. Todo esto para ser aplicado a desarrollos de programas pequeños y grandes. Los datos históricos permiten hacer desarrollos más predecibles y eficientes.

En los desarrollos de gran tamaño se requiere un manejo exitoso de los equipos de trabajo. El proceso de software es un conjunto de pasos para realizar el desarrollo y mantenimiento de software, estableciendo lineamientos que le indican al ingeniero de software la manera en la que se debe trabajar. De esta manera se puede coordinar los trabajos de cada uno de los integrantes y dar seguimiento a su desarrollo.

El establecimiento de un proceso no es solamente la definición de técnicas y prácticas, sino que debe de estar fundamentado en la necesidad de cambio. Los desarrollos a gran escala se vuelven complejos, es por ello que el marco de madurez de procesos de software fue desarrollado. De esta forma se organiza para determinar las capacidades del proceso actual y establecer prioridades de mejora. A través de cinco niveles se representa progresivamente la madurez del proceso.

1. Inicial
2. Repetible
3. Definido
4. Administrado
5. Optimizado

El progreso en PSP cuenta con las siguientes fases de procesos de mejora:

- PSP0: proceso base

- PSP1: proceso de planificación personal
- PSP2: proceso de administración personal de calidad
- PSP3: proceso personal cíclico

PSP es diseñado para ser un CMM[12] (*Capability Maturity Model*) a nivel individual.

PSP enseña a los ingenieros a:

- Administrar la calidad de sus proyectos.
- Hacer compromisos que se puedan cumplir.
- Mejorar estimaciones y planificaciones.
- Reducir los defectos en sus productos.

El personal constituye gran parte de los costos del desarrollo de software, las habilidades de los ingenieros determinan considerablemente los resultados del proceso de desarrollo de software. PSP puede ser usado por ingenieros como guía o como un disciplinado y estructurado acercamiento al desarrollo de software. PSP es un prerrequisito para una organización que pretende introducir TSP (*Team Software Process*).

PSP puede ser incluida en muchas partes del proceso de desarrollo de software, incluyendo:

- Desarrollo de pequeños programas.
- Definición de requerimientos.
- Elaboración de documentos.
- Pruebas de sistemas.
- Mantenimiento de sistemas.
- Mejora de grandes sistemas de software.

[12] Este modelo se aborda en el tema diez.

Aunque las prácticas de PSP eran posibles y daban resultados, era casi imposible mantener la disciplina de las prácticas si el ambiente circundante no los animaba y exigía. Por esto Humphrey presentó TSP para unidades pequeñas de desarrollo.

Bibliografía del tema 3

Humphrey, Watts S. *A discipline for software engineering*. Mexico City, Addison-Wesley, 1995.

PROJECT MANAGEMENT INSTITUTE. *Project Management Body of Knowledge*. Pennsylvania, PMI, 2004.

Actividades de aprendizaje

A.3.1. Realizar una investigación que identifique las causas por las cuales fracasan los proyectos de desarrollo de software y realizar un informe.

A.3.2. Realizar un esquema en el que se identifiquen las fases más comunes de un proyecto y las actividades asociadas a cada una.

A.3.3. Realizar un reporte con tres ejemplos de herramientas de software que se puedan emplear en alguna parte del ciclo de vida de proyectos.

A.3.4. Elaborar un cuadro comparativo de los problemas más comunes entre desarrollos pequeños y los de gran escala.

A.3.5. Realizar una propuesta con los elementos expuestos en este tema para un área de desarrollo con cuatro personas.

Cuestionario de autoevaluación

1. ¿Cuál sería la función principal de una oficina de proyectos?
2. ¿Qué significa PSP?
3. ¿Qué es el ciclo de vida de proyectos?
4. ¿Qué es un proyecto?
5. ¿A qué nivel puede ser empleado el PSP dentro del proceso de desarrollo?
6. ¿Qué elementos puede definir un ciclo de vida de proyectos?
7. ¿Quién define las fases que deberán de llevarse a cabo del inicio al final de un proyecto?
8. ¿En qué difiere un proyecto al trabajo operativo?
9. ¿Cuál es la finalidad de dividir el proyecto en fases?
10. ¿A qué tipo de proyectos se puede emplear el PSP?

Examen de autoevaluación

Instrucciones. Escribe en la columna de la derecha la letra V (verdadero) o F (falso) según corresponda al enunciado

No.	Enunciado	V o F
1.	Un proyecto es un esfuerzo temporal emprendido para crear un producto, servicio o resultado único.	
2.	Los procesos de administración de proyectos son los mismos que las fases de un proyecto.	
3.	La oficina de proyectos define los productos a entregar en cada fase.	
4.	La oficina de proyectos se encarga de alinear los proyectos a los objetivos del cliente o de la organización.	
5.	El ciclo de vida de proyectos coordina la comunicación entre proyectos.	
6.	Una de las funciones de la oficina de proyectos es establecer la metodología de la administración de proyectos.	
7.	Una disposición legal es una fuente para generar un proyecto.	
8.	PSP sólo es para desarrollo a gran escala.	
9.	PSP es una propuesta para grupos de desarrollo.	
10.	PSP sólo puede utilizarse en una parte dentro del proceso de desarrollo de software.	

TEMA 4. ADMINISTRACIÓN DE REQUERIMIENTOS

Objetivo particular

Al finalizar este tema el alumno conocerá la importancia de los requerimientos para un proyecto de software, la forma en la que se clasifican y las técnicas para recolectarlos dentro del proceso de requerimientos.

Temario detallado

4.1 ¿Qué es un requerimiento?
4.2 Clasificación de requerimientos
4.3 FURPS y FURPS+
4.4 Buenos y malos requerimientos
4.5 Técnicas de recopilación de requerimientos
4.6 Modelo general
4.7 El modelo de RUP

Introducción

La definición de requerimientos es la primera actividad realizada dentro del proceso de desarrollo y los proyectos se encuentran vulnerables cuando estas actividades están poco desarrolladas. Su relación con las demás fases del desarrollo de software es muy cercana porque con base en los requerimientos se definirán las actividades a desarrollar.

4.1 ¿Qué es un requerimiento?

Un requerimiento describe una condición o capacidad que un sistema debe de cumplir, derivados de las necesidades del usuario, por contrato, estándares, especificaciones o cualquier otro documento formalmente establecido.

Un requerimiento es definido como una propiedad que debe de ser exhibida con la finalidad de resolver algún problema del mundo real.

Los requerimientos de software expresan las necesidades y restricciones colocadas a un producto de software que contribuye a la solución de un problema del mundo real. Una propiedad esencial de un requerimiento es que se pueda verificar.

Por lo general los requerimientos para un software en particular es una compleja combinación de requerimientos de diferentes personas con diferentes niveles dentro de una organización y con un ambiente diferente en el cual se operará.

Existen otros requerimientos que representan propiedades emergentes, las cuales no pueden ser consideradas como sólo un elemento sino que depende del buen funcionamiento entre los otros elementos.

Requerimiento: Condición o capacidad que el sistema debe cumplir.

- Necesidad que se debe cubrir.
- Constituyen la definición del sistema que se va a construir o mejorar

Para que algo sea un requerimiento:

- Debe ser solicitado formalmente
- Debe ser documentado
- Debe ser analizado formalmente para verificar el impacto en el proyecto
- Debe ser aprobado

La palabra “requerimientos” significa diferentes cosas para diferentes personas

- Para un ejecutivo puede ser un concepto de producto de alto nivel desde el punto de vista del negocio.
- Para un cliente puede ser una lista de ideas o soluciones propuestas.
- Para un desarrollador puede ser una interfaz gráfica.

Los requerimientos son descripciones de cómo debería comportarse el sistema o una propiedad o atributo de un sistema. También deben delimitar el proceso de desarrollo y al sistema.

La redacción de los requerimientos se debe de realizar de tal manera de que sea fácil de entender para los lectores y sobre todo cuando se trata de personas que no tienen relación con los sistemas informáticos. También se tiene que tener en cuenta el ambiente en el cual operará y su adecuación a los estándares y de más lineamientos establecidos, lo que significa que existen otras fuentes para los requerimientos.

4.2 Clasificación de requerimientos

De manera tradicional, existe la clasificación que se hace de requerimientos de dos tipos:

Requerimientos funcionales los cuales especifican funciones que el sistema debe ser capaz de realizar, sin tomar restricciones físicas a considerar.

Requerimientos no funcionales los cuales describen atributos del sistema o atributos del ambiente del sistema. También son conocidos como de delimitación o de calidad.

QFD (*Quality Function Deployment*) es una técnica de administración de calidad que traslada las necesidades del cliente en requerimientos técnicos de software. Desarrollada y utilizada por primera vez en Japón por Mitsubishi Heavy Industries a principio de la de la década de 1970. QFD enfatiza lo que es

realmente valioso para el cliente y lo pone en el proceso de ingeniería. Se identifican tres tipos de requerimientos:

- *Requerimientos normales*. Las metas y objetivos son establecidos durante las reuniones con el cliente. Si los requerimientos se presentan entonces el cliente se encuentra satisfecho.
- *Requerimientos esperados*. Estos requerimientos son implícitos del producto y es posible que el cliente no los establezca. Su ausencia puede ser la causa de insatisfacción en el cliente.
- *Requerimientos extra*. Son características que van más allá de las expectativas del cliente provocando una mayor satisfacción cuando se presentan.

4.3 FURPS y FURPS+

Son factores de calidad de software generado por Hewlett-Packard bajo el acrónimo formado de las palabras en inglés (*Functionality, Usability, Reliability, Performance, Supportability*). Sólo describen que pueden ser utilizados para establecer métricas de calidad dentro del proceso de ingeniería del software.

Funcionalidad. Evalúa el conjunto de características y capacidades del programa, la generalidad de las funciones que han sido entregadas y la seguridad de todo el sistema.
Facilidad de uso. Considera factores humanos, estéticos, consistencia y documentación.

Confiabilidad. Mide la frecuencia y severidad de las fallas, la exactitud de los resultados, la capacidad de recuperación de fallas y pronóstico del programa.

Desempeño. Mide la velocidad de procesamiento, tiempo de respuesta, consumo de recursos, rendimiento y eficiencia.

Soporte. Combina la capacidad del programa en expansión, adaptación, mantenimiento, servicio, pruebas, compatibilidad, configuración, facilidad de instalación y la facilidad para localizar los problemas.

El modelo FURPS+ es una extensión a la que se le adicionan los siguientes elementos:

Diseño. Especifica o restringe el diseño de un sistema.

Implementación. Especifica la construcción o codificación de un sistema con estándares, lenguajes, políticas de integración de base de datos, límites de recursos y ambientes de operación.

Interfaz. Elementos de interacción con el sistema, restricciones de tiempo y formato.

Físicos. Establece la característica material, forma, peso y tamaño.

4.4 Buenos y malos requerimientos

Una manera de realizar una buena obtención de requerimientos implica la utilización de técnicas repetibles y sistemáticas para asegurar que los requerimientos del sistema sean completos, consistentes, relevantes, etc. También implica cubrir las actividades relacionadas con el descubrimiento, documentación y mantenimiento de los requerimientos.

Los atributos individuales a considerar para un buen requerimiento serían:

- Correcto. Cualidad que sólo puede ser asegurado por el cliente o usuario.

- Posible (factible). Cualidad que requiere conocimiento de parte del área de desarrollo, herramientas disponibles, técnicas, gente y presupuesto que posibilite la satisfacción final del requerimiento.
- Necesario. Cualidad que puede ser determinada entre el cliente y el desarrollador en la obtención de requerimientos, con la finalidad de establecer los requerimientos a entregarse para esa versión.
- Priorizado. Cualidad determinada entre usuario y desarrollador para establecer en cada requerimiento un nivel de prioridad como: muy importante, absolutamente necesario, importante para la siguiente liberación y opcional.
- Claro. Cualidad que se refiere a que sólo puede tener un significado; fácil para leer y entender.
- Conciso. Que contenga la información necesaria para continuar con el desarrollo. Asociando historia, costos, programación, tareas, productos, etc.
- Verificable (probado, medido). Cualidad que significa que una persona o máquina puede revisar si el software cubrió los requerimientos.

Como conjunto, la especificación de requerimientos de software debería de contar con las siguientes características:

- Completo. Cualidad que puede ser asegurada por el cliente revisando que todos los requerimientos significativos se encuentran incluidos (funcionales, desempeño, externos, etc).
- Consistente. Cualidad que el desarrollador asegura entre los documentos internos.
- Modificable. El cambio en los requerimientos es algo normal.
- Rastreable. Cualidad compartida entre cliente y desarrollador en la que el requerimiento establecido puede seguirse desde su establecimiento hasta su entregable como software.

Los problemas que se presentan al momento de obtener requerimientos:

- Afectados que saben lo que quieren pero no pueden expresarlo.
- Afectados que pueden no saber lo que quieren.
- Afectados que saben lo que quieren hasta que usted les da lo que ellos decían que querían.
- Analistas que piensan que entienden los problemas de los usuarios mejor que los usuarios.

Un requerimiento no es:

- Una solicitud informal de alguien en una reunión, un pasillo o un elevador o una llamada telefónica
- Solicitudes de clientes por medio de encuestas, correos electrónicos o buzones de sugerencias
- Observaciones o comentarios durante reuniones de ventas o mercadeo

Un requerimiento documentado nos permite

- Llegar a un acuerdo con el cliente de lo que el sistema debe hacer.
- Identificar quién interactuará con el sistema.
- Identificar la interfaz que el sistema debe tener.
- Ayudar a verificar que no faltan requerimientos.
- Verificar que los desarrolladores entienden los requerimientos.
- Facilitar el acuerdo con el cliente de los requerimientos del sistema.
- Utilizar terminología que el usuario entiende.
- Verificar el entendimiento del desarrollador del sistema.
- Identificar el rol de los usuarios del sistema.
- Identificar la interfaz del sistema.
- Ayudar a verificar que todos los requerimientos sean capturados.

4.5 Técnicas de recopilación de requerimientos

Existen diferentes formas de obtener los requerimientos. Cada una de las técnicas nos permite obtener los requerimientos desde un punto de vista en particular. Es posible que las personas involucradas les sea más fácil expresar de manera más clara sus requerimientos con alguna técnica más que con otra. Es responsabilidad del analista de requerimientos establecer la técnica que utilizará. No hay nada que pueda restringir el uso de más de una técnica, todo eso quedará a juicio del analista de requerimientos.

Entrevistas. Es la técnica que comúnmente se emplea para obtener los requerimientos. El ingeniero de requerimientos o el analista discuten el sistema con los clientes y construyen los requerimientos. Existen básicamente dos tipos de entrevistas:

- Las entrevistas cerradas en donde el ingeniero de requerimientos busca las respuestas de conjunto de preguntas predefinidas.
- Las entrevistas abiertas en estas no existe una agenda predefinida y el ingeniero de requerimientos discute de manera abierta con los involucrados acerca del sistema.

No existe realmente una limitante para emplear alguna de las dos entrevistas e incuso se pueden llegar a combinar en un momento determinado. Esta técnica puede facilitar a los involucrados a que expresen sus problemas y dificultades en su trabajo. Pero también puede generar en el usuario una expectativa poco realista sobre el sistema. Por otra parte puede quedar inconclusa la identificación del dominio del sistema y las cuestiones organizacionales que afectan los requerimientos.

Hay dos puntos esenciales para una entrevista efectiva:

- El entrevistador debe de ser de mente abierta y de buena disposición para escuchar a los involucrados.

- Los involucrados deben de proporcionar algún punto de partida para la discusión.

El conocimiento de las aplicaciones es difícil de obtener durante las entrevistas porque:

- Muchas aplicaciones cuentan con su propia terminología y a muchos de los involucrados les resulta difícil discutir sobre ella si no utilizan esa terminología.
- Existen conocimientos sobre la aplicación en los cuales al involucrado le resulta difícil explicar o le resulta familiar que nunca se llegó a pensar en explicarlo.

Métodos ligeros de sistemas. Son modelos flexibles enfocados a producir modelos menos formales del conjunto técnico-social de un sistema. Consideran la automatización del sistema, la gente y la organización.

Varios de esos métodos han sido incluidos en SSM (*Soft Systems Methodology*), ETHICS y *Eason's User-Centred Design*.

SSM y otros métodos no son técnicas como tal para obtener requerimientos de manera detallada. Su aporte radica principalmente en hacer mucho más entendible el problema, la situación organizacional en donde existe el problema y las restricciones para la solución del problema.

Observación y análisis social. El trabajo es una actividad social que involucra grupos de gente que cooperan realizando diferentes tareas. La naturaleza de la cooperación es compleja y variada dependiendo de la gente involucrada, el ambiente y la organización en la que se realiza el trabajo. A los individuos y equipos les resulta difícil explicar la forma en la que trabajan y cooperan. El trabajo y sus mejoras muchas veces se realizan de manera intuitiva.

En ocasiones la observación es la mejor forma de entender una tarea que el cuestionamiento directo. Científicos sociales y antropólogos han utilizado la técnica de observación pasiva por largos periodos de tiempo para desarrollar un completo y detallado entendimiento de varias culturas.

Aplicando esta técnica a la obtención de requerimientos se presentan las siguientes guías:

- Asumir que la gente que se está estudiando es buena realizando su trabajo.
- Invertir tiempo conociendo a la gente involucrada y estableciendo una relación.
- Realizar notas detalladas del trabajo durante las observaciones.
- Entrevistas abiertas en donde la gente pueda hablar de su trabajo.
- Mantener la utilización de registro de video y audio al mínimo.
- Sesiones cortas para hablar del trabajo con la gente fuera del proceso puede identificar elementos críticos en el trabajo.

Esta técnica deberá ser utilizada como apoyo a la obtención de requerimientos.

Escenarios. A los usuarios finales les resulta más fácil realizar ejemplos en la vida real que realizar descripciones abstractas de un sistema. Por eso, que resulta conveniente utilizar escenarios de interacción entre el usuario y el sistema. De esta manera al momento de que el usuario simula la interacción es posible describir las tareas o funciones que desea.

Incluso el desarrollo de escenarios sin la interacción del usuario proporciona información para los requerimientos. Descubrir los escenarios expone las posibles interacciones que puede tener el sistema. En algunos métodos de análisis orientado a objetos los escenarios son una parte básica. Los escenarios pueden ser pensados como un relato de la manera en que un sistema es utilizado. Proporcionan información adicional a los requerimientos.

Se pueden elaborar escenarios de diferente forma pero como mínimo deberán contener la siguiente información:

- Una descripción del estado del sistema antes de iniciar el escenario
- Un flujo normal de eventos en el escenario
- Excepciones al flujo normal de eventos
- Información de otras actividades que pueden realizarse en paralelo
- Una descripción del estado del sistema al finalizar el escenario

La aplicación de escenarios involucra el trabajo en conjunto del ingeniero de requerimientos y el usuario final. De esta interacción surgen comentarios, problemas, sugerencias, maneras de realizar las actividades, involucrados, alternativas ante acciones inesperadas, etc.

Reutilización de requerimientos. Es una buena práctica para sistemas e ingeniería de software la reutilización del conocimiento para un nuevo sistema.

A pesar de que existen diferentes requerimientos para cada sistema, existen situaciones en las cuales es posible realizar la reutilización de los requerimientos.

- Requerimientos relacionados que proporcionan información acerca de la aplicación: son requerimientos que más que ver con la funcionalidad, tiene que ver con las restricciones u operaciones que se derivan de la aplicación.
- Requerimientos relacionados con la presentación de la información: es posible tener una interfaz consistente para todos los sistemas para las organizaciones.
- Requerimientos relacionados con las políticas de una organización.

FAST (*Facilitated Application Specification Techniques*). Su intención es conformar un grupo de trabajo con clientes y desarrolladores que trabajan juntos para identificar el problema, proponer elementos de solución, negociar diferentes acercamientos y especificar un conjunto preliminar de requerimientos.

También podemos encontrar otro tipo de técnicas para la obtención de requerimientos.

- Cuestionarios
- JAD (*Joint Application Development*)
- Mapas mentales
- Casos de uso
- Lluvia de ideas
- Prototipos
- Talleres de requerimientos
- Bosquejos
- Actuación de roles

4.6 Modelo general

Para comprender el proceso de requerimientos se debe tomar en cuenta que:

- No es una actividad discreta en el ciclo de vida del software. Comienza junto con el proyecto y continúa refinándose durante el ciclo de vida.
- Identifica los requerimientos de software como elementos de configuración y se administran con las mismas prácticas de administración de la configuración al igual que otros productos de los procesos del ciclo de vida.
- Necesita adaptarse a la organización y al contexto del proyecto.

Una manera de llevar a cabo un manejo adecuado de los requerimientos es a través de un proceso de requerimientos. Un proceso de este tipo debe de incluir las siguientes actividades:

- Obtención de requerimientos. Los requerimientos son descubiertos a través de los usuarios, clientes, documentos del sistema, experiencia y estudios de mercado.
- Negociación y análisis de requerimientos. Los requerimientos son analizados en detalle a través de un proceso formal de negociación en el que participen diferentes involucrados para decidir la aceptación de los requerimientos.
- Validación de requerimientos. Se debe realizar una revisión cuidadosa de los requerimientos para que se encuentren completos y consistentes.

4.7 El modelo de RUP

IBM RUP® (*Rational Unified Process*®), es un proceso de desarrollo de software que provee las mejores prácticas que pueden aplicarse dentro de esta área. Es uno de los muchos procesos dentro del *Rational Process Library*. Su objetivo es asegurase de producir software de alta calidad.

El UP (*Unified Process*) es el resultado de varias décadas de desarrollo. De ella se pueden encontrar los siguientes puntos importantes en su historia:

- Sus inicios datan de 1967 cuando Ericsson modeló un sistema completo con un conjunto interconectado de bloques. Ensamblando los niveles más bajos dentro de los más altos para tener una mejor administración del sistema. En esta fase se identificaron los casos de uso y una primera representación de lo que serían los diagramas de colaboración y de transición de estados.
- En 1987 Ivar Jacobson deja Ericsson y se establece en Objectory AB en Estocolmo. Trabajó junto con sus asociados en el proceso Objectory y produjo las versiones 1.0 en 1988 a la 3.8 en 1995. Este producto fue

visto como un sistema, permitiendo hacer nuevas versiones que permitían ajustarse a las necesidades particulares.

- Rational Software Corporation adquirió Objectory AB en 1995. Surgió la necesidad de unificar los principios de los distintos procesos de desarrollo y Rational había desarrollado varias prácticas de desarrollo de software. En donde las contribuciones más notables fueron el énfasis en la arquitectura y el desarrollo iterativo. UML (*Unified Modeling Language*) fue desarrollado y utilizado como lenguaje de modelado del ROP 4.1 (*Rational Objectory Process*).
- Rational adquirió más compañías de software y con ello más experiencia en los procesos de software para ser aplicados en el ROP. Se ingresó un flujo de trabajo para el modelado de negocios. Se expandió el diseño de las interfaces del usuario empleando casos de uso. Para 1998 el ROP podía ser aplicado para todo el ciclo de vida de desarrollo de software. Con las diversas contribuciones realizadas Rational liberó el RUP 5.0 (*Rational Unified Process*). Su nombre refleja la unificación en el desarrollo, el uso de UML y la unificación del trabajo de muchas metodologías.
- En el 2002 se anuncia la adquisición de Rational por IBM.

UP se encuentra basado en componentes, lo que significa que el software comienza a ser construido a partir de componentes de software que son interconectados a través de interfaces bien definidas.
UP utiliza UML para realizar los planos del sistema. Constituye una parte integral y ambos fueron desarrollados a la par.

Las partes esenciales que distinguen al RUP son:

- Manejo de casos de uso. Para construir un software exitoso se tiene que conocer las perspectivas del usuario de lo que quiere y necesita. Los usuarios no abarcan sólo a personas sino también a otros sistemas. Un caso de uso es una pieza de funcionalidad en un sistema que

proporciona al usuario un resultado. Los casos de uso capturan los requerimientos. El conjunto de casos de uso forma un modelo de casos de uso que describe la funcionalidad del sistema. Existe una influencia recíproca entre los casos de uso y la arquitectura de manera que van madurando durante el ciclo de vida.

- Centrado en la arquitectura. Involucra aspectos estáticos y dinámicos del sistema. La arquitectura va creciendo con las necesidades de la organización, de los usuarios y clientes. También influyen otros factores como la plataforma de software, los componentes reutilizables, consideraciones de desarrollo, sistemas heredados y requerimientos no funcionales. La arquitectura es vista como un diseño completo de las características más importantes. Una función corresponde a un caso de uso y forma la arquitectura, estos dos últimos deben elaborarse en paralelo. El arquitecto trabaja con un conjunto de casos de uso. Cada caso de uso es especificado y realizado en subsistemas, clases y componentes. Los casos de uso son especificados y madurados con los descubrimientos de la arquitectura.
- Iterativo e incremental. En desarrollos largos resulta práctico realizar divisiones en el trabajo, en mini proyectos. Cada mini proyecto es una iteración que resulta en un incremento. Las iteraciones hacen referencia a los flujos de trabajo (*workflow*) y los incrementos al crecimiento del producto. Para ser más efectivos, las iteraciones deben ser controladas. El desarrollador basa su selección de lo que se implementará en una iteración con los casos de uso asignados y los riesgos implicados en cada iteración. Cada mini proyecto incluye su trabajo de desarrollo: análisis, diseño, implementación y pruebas. En cada iteración los desarrolladores identifican y especifican los casos de uso relevantes, crean un diseño utilizando la arquitectura seleccionada como guía, implementan el diseño en componentes y verifican que los casos de uso satisfagan los casos de uso. Si la iteración alcanza las metas entonces se continúa a la siguiente iteración.

UP repite una serie de ciclos que se realizan en la vida de un sistema. Cada ciclo entrega un producto liberado al cliente. Cada ciclo consiste en cuatro fases: principio, elaboración, construcción y transición. Cada fase se encuentra dividida en iteraciones.

Cada ciclo produce una nueva liberación del sistema y cada liberación es un producto listo para ser entregado. Consiste en un conjunto de código fuente que puede ser compilado y ejecutado, más los manuales y entregables asociados. El producto final va más allá del código ejecutable, pues incluye los requerimientos, casos de uso, requerimientos no funcionales y casos de prueba. Esto facilitará el aumento o modificación de funcionalidades del sistema ya que a sus elementos se les puede dar un seguimiento.

Cada ciclo es dividido en cuatro fases. Dentro de cada fase se divide el trabajo en iteraciones y resultado de incrementos. Cada fase termina en un hito. Se define cada hito por la disponibilidad de un conjunto de artefactos, los cuales pueden ser modelos o documentos.

Existen cinco flujos de trabajo principales: requerimientos, análisis, diseño, implementación y pruebas. Una iteración típica cruza los cinco flujos de trabajo que se encuentran presentes en cada fase.

En la fase de principio se desarrolla una visión del producto final y el caso del negocio (*business case*) es presentado. Un modelo de caso de uso simplificado contiene los casos de uso más importantes. En esta parte la arquitectura es tentativa. Los riesgos más importantes son identificados y priorizados, se planifica la fase de elaboración a detalle y el proyecto es estimado tentativamente.

En la fase de elaboración los casos de uso son especificados a detalle y la arquitectura es diseñada. Los casos de uso más críticos que fueron encontrados en esta fase son desarrollados. El resultado de esta fase es la

arquitectura base. Se planifican las actividades y se estiman los recursos requeridos para completar el proyecto.

En la fase de construcción el producto es construido y colocado en la infraestructura. Los recursos solicitados son utilizados. Al final de esta fase el producto contiene todos los casos de uso que fueron agregados para la liberación.

Finalmente la fase de transición es el periodo de movimiento del producto a una liberación beta. En esta liberación los usuarios interactúan con el producto y se reportan defectos y deficiencia. También se involucran en esta fase las actividades de manufactura, entrenamiento al personal, soporte, corrección de defectos posteriores a la liberación. El mantenimiento se realiza dividiendo los defectos en aquellos que su afectación justifica inmediatamente una nueva liberación y aquellos que pueden esperar y ser incluidos en la próxima liberación ordinaria.

Bibliografía del tema 4

Jacobson, Ivar. *The Unified Software Development Process*. Mexico City, Addison-Wesley, 1999.

Kontoya, Gerald. *Requirements Engineering: Processes and Techniques*. Chichester, Wiley, 1997.

Sommerville, Ian. *Requirements Engineering: A good practice guide*. New York, Wiley, 1998.

Páginas de referencia

http://www.qfdi.org

Actividades de aprendizaje

A.4.1. Elaborar un cuadro con las clasificaciones de los requerimientos y llenar cada una de las clasificaciones con ejemplos.

A.4.2. Elabore una lista de verificación con los elementos que se tomarían en cuenta para revisar un requerimiento.

A.4.3. Elaborar la agenda de una entrevista cerrada en la que se incluyan las técnicas que se emplearán, los temas a tratar y el desarrollo de la entrevista.

A.4.4. Elaborar un cuestionario con preguntas que se aplicarían al usuario en la primera entrevista cerrada.

A.4.5. Investigar y realizar un reporte de dos herramientas de software que se puedan emplear para la obtención de requerimientos.

Cuestionario de autoevaluación

1. ¿Qué es un requerimiento?
2. ¿Cuál es la clasificación tradicional de los requerimientos?
3. ¿Qué significa RUP?
4. ¿Qué significa FURPS?
5. ¿Qué actividades debería de realizar el proceso de requerimientos?
6. Describe las técnicas que existen para obtener requerimientos.
7. ¿A qué otros elementos hace referencia el modelo FURPS+?
8. ¿Cuál es la relación que tienen los requerimientos con las demás fases dentro del desarrollo de software?
9. ¿Cuáles son los problemas más comunes que se relacionan con la obtención de requerimientos?
10. ¿Cuáles son los elementos principales del RUP?

Examen de autoevaluación

Instrucciones. Seleccione una de las opciones y escriba la letra correspondiente en el paréntesis de la derecha de cada pregunta.

1. ¿RUP es? ( )

a) Una técnica para obtener requerimientos
b) Una clasificación de requerimientos
c) Un proceso de requerimientos
d) Un proceso de desarrollo de software

2. ¿Los cuestionarios son? ( )

a) Una herramienta de análisis
b) Una técnica para la obtención de requerimientos
c) Una metodología de requerimientos
d) Ninguna de las anteriores

3. ¿De qué modelo son parte los requerimientos físicos? ( )

a) ETHICS
b) FURPS+
c) UML
d) FURPS

4. Es uno de los tipos de entrevistas ( )

a) Entrevistas de campo
b) Entrevistas de definición
c) Entrevistas abiertas
d) Todas las anteriores

5. Es uno de los elementos del modelo FURPS ( )

a) Facilidad
b) Utilidad
c) Productividad
d) Ninguno de los anteriores

6. Son las actividades sugeridas para un proceso de requerimientos ( )

a) Obtención de requerimientos
b) Negociación y análisis de requerimientos
c) Validación de requerimientos
d) Todas las anteriores

7. Forma parte de los métodos ligeros de sistemas ( )

a) SSM
b) FAST
c) QFD
d) UML

8. Son requerimientos que dependen del buen funcionamiento de otros elementos. ( )

a) Requerimientos físicos
b) Requerimientos de propiedades emergentes
c) Requerimientos operativos
d) Ninguna de las anteriores

9. Es una de las características que hace único al RUP ( )

a) Desarrollo iterativo e incremental
b) Funcionalidad
c) Fácil de usar
d) Rapidez

10. Es uno de los elementos que es necesario para que algo sea un requerimiento ( )

a) Solicitud formal
b) Estar documentado
c) Requiere aprobación
d) Todas las anteriores

TEMA 5. ANÁLISIS DE SOFTWARE

Objetivo particular

Al finalizar este tema el alumno será capaz de identificar la importancia del análisis de software y las actividades para llevarlo acabo dentro del ciclo de desarrollo de software.

Temario detallado

5.1 Estudios de factibilidad

5.2 Estructurado

5.3 Orientado a objetos

5.4 UML

Introducción

El análisis de software es un paso intermedio entre la obtención de requerimientos y el diseño. En esta fase se aplican los métodos y técnicas a partir de los requerimientos y objetivos establecidos para obtener una mejor comprensión del proyecto de software que se pretende realizar.

El análisis de requerimientos genera una representación de la información, funciones y comportamiento que puede ser trasladado a datos, arquitectura, interfaces y diseños a nivel componentes.

5.1 Estudios de factibilidad

Este tipo de estudios proporcionan los elementos para decidir la conveniencia de realizar un proyecto considerando elementos internos y externos. De esta forma se pueden incluir a los estudios de factibilidad: factibilidad técnica, factibilidad económica y análisis de sensibilidad.

Es importante destacar que para realizar estos pasos es necesario que ya se tengan establecidos los objetivos del proyecto.

Factibilidad técnica. Se refiere a los estudios de concreción técnica del proyecto, teniendo en cuenta los objetivos del proyecto y los recursos que serían necesarios. Para ello se evalúa:

- Las probabilidades reales de realizar el proyecto
- Tipo de riesgos técnicos implicados
- Medidas para prevenir y disminuir los riesgos

Una herramienta de análisis que se puede utilizar es la generación de escenarios, en él se puede representar un proyecto tanto en las mejores condiciones como en el peor de los casos. De esta forma es posible identificar los riesgos, entradas y salidas.

Al momento de realizar la programación de las actividades es recomendable que se verifique la disponibilidad de los recursos para el momento en el cual se van a utilizar así como la cantidad, todo esto con la finalidad de obtener cierta seguridad. Otro aspecto a considerar por parte de los recursos que se emplearán es el priorización por importancia o costo de cada uno.

Analizar la estructura de la organización permitirá identificar las condiciones con las que cuenta para llevar a cabo el proyecto. De esta manera se realizará la comparación entre la propuesta teórica y la real para encontrar la manera más adecuada de aplicar el proyecto dentro de la organización.

En el marco legar se analizan todas las leyes, códigos y normas que pueden afectar a un proyecto al momento de su realización. Los reglamentos técnicos son de carácter obligatorio. Mientas que las normas son de carácter voluntario. Son determinaciones convencionales que son dadas por un organismo que se encarga de proporcionarlas y certificar su cumplimiento.

En cuanto al impacto ambiental se realiza una evaluación del impacto que puede llegar a tener el proyecto en la flora, fauna, salud humana, etc. La importancia radica en reducir al mínimo las alteraciones que pudieran presentarse.

Factibilidad económica. Es el estudio de las posibilidades de concreción económica del proyecto.

A partir de los objetivos establecidos del proyecto y con el análisis previo de los recursos que se necesitan, es posible evaluar el tipo de riesgos financieros, los ingresos y egresos del proyecto, alternativas de financiamiento, puntos débiles de la economía del proyecto y las medidas para contrarrestarlos.

El análisis de mercado es otro de los aspectos que se deben de considerar para determinar la viabilidad de un proyecto. Se realizan evaluaciones del mercado.

En la oferta se observa a los competidores dentro del mercado y cada una de sus características que la hacen diferenciarse de los demás.

En la demanda se evalúa a los clientes o consumidores. Con la formación de segmentos en el mercado se determinar su estructura de la demanda.

A partir de la oferta y del precio que existe en el mercado, se establece una base sobre la cual puede trabajar el proyecto.

El aspecto comercial tiene que ver con las alternativas de comercialización existente y aquellas que pueden ser desarrolladas.

En el análisis impositivo se toman en cuenta todos los elementos impositivos que afectan al proyecto como: precios, carga social, impuestos, etc.

El análisis económico financiero permite la comprensión de los efectos positivos y negativos, la magnitud del esfuerzo realizado y la conveniencia de la realización. Para lo cual es necesario identificar la situación anterior a la realización del proyecto. Este estudio muestra el manejo de ingresos y egresos en el tiempo.

Factibilidad operativa. Es el impacto del proyecto sobre la organización. Con este análisis se obtiene el dimensionamiento de los cambios que se tendrán en la organización. Se tomará en cuenta las funciones y estructura que tiene la organización, se elaborarán nuevas funciones, roles y flujos de trabajo. Se identifican los posibles riesgos que conllevan las modificaciones. Se establece un plan de capacitación para las partes involucradas y se genera el análisis costo-beneficio operativo.

Factibilidad legal. Es este paso se analiza las posibles infracciones, violaciones o responsabilidades legales que puede llevar el desarrollo de un sistema de software. Esto incluye licencias de hardware, software, bandas de frecuencia, leyes ambientales, etc.

Análisis de sensibilidad. Permite calcular la importancia de una hipótesis en particular. Esto genera un enfoque sobre la precisión de los factores críticos.
En este punto se toma en cuenta el estudio de los factores internos y externos que pueden influir en el resultado del proyecto. A cada proyecto en particular se puede ver afectado por diferentes factores, al inicio se deben de establecer datos base para cada una de las variables. Para observar el comportamiento de cada variable se puede hacer uso de métodos estadísticos que permitan entender los ciclos y tendencias. La afectación de una o varias variables puede ser tanto a los costos como a los beneficios del proyecto.

El comportamiento de las variables más significativas pueden hacerse sobre tres escenarios principales: el más probable, el optimista y el pesimista. Para estos escenarios es importante realizar la identificación de las variables más

confiables y de aquellas que son poco confiables en su comportamiento y a partir de ellas generar cada uno de los escenarios.

5.2 Estructurado

Con el nombre de estructurado se hace referencia a los recursos disponibles para la perspectiva de programación estructurada y diferenciarla de este modo de los recursos disponibles para la perspectiva orientada a objetos. El análisis de requerimientos para la programación estructurada se divide en cinco áreas:

- Reconocimiento del problema
- Evaluación y síntesis
- Modelado
- Especificaciones
- Revisiones

Los métodos de análisis se encuentran basados en un conjunto de principios:

- La información de un problema debe de ser representado y entendido
- Las funciones del software deberán de ser definidas
- El comportamiento del software debe de ser representado
- Los modelos que representan información, funciones y comportamiento deberán de ser divididos de manera de que sean abarcados a detalle.
- El proceso de análisis deberá de mover la información esencial hacia una implementación detallada

Con la finalidad de realizar una mejor representación del sistema de software se presentan algunos modelos.

Existe un conjunto de modelos (datos, función y comportamiento) que pueden utilizarse con la finalidad de hacer una representación gráfica que ayude a la comprensión y análisis de los requerimientos.

ERD (*Entity Relationship Diagrams*). Estos modelos permiten identificar los objetos de datos y sus relaciones a través de una notación gráfica. Se definen los datos que ingresan, se almacenan, transforman y genera la aplicación. En la representación los datos se organizan en grupos y se muestra la relación entre grupos.

Los atributos de cada entidad dentro del ERD son descritos en el diccionario de datos. Los atributos para un objeto de datos son determinados con el entendimiento del contexto del problema. Los atributos se asocian a identificadores y llaves de las bases de datos.

Un diagrama ERD se compone de cajas que representan a las entidades y líneas que indican las relaciones entre ellos. Una entidad es algo en el mundo del que necesitamos almacenar datos. Las entidades se identifican dentro de un sistema con nombres de sustantivos (archivos, repositorio de datos, elementos que son manufacturados, roles funcionales, etc.). Los eventos y actividades también pueden considerarse como entidades. Una entidad es sólo una representación de datos.

Un atributo es una característica única de una entidad. Las relaciones son las conexiones entre las entidades. Cada una de las relaciones representa una asociación entre dos entidades. La cardinalidad se refiere al número de relaciones entre instancias de dos entidades (uno a uno, uno a muchos, muchos a muchos). La normalización reestructura datos para reducir redundancia y dependencias inconsistentes y para incrementar la flexibilidad del modelo. La redundancia aumenta el tamaño en disco y hace más difícil el mantenimiento. Existen cinco niveles de normalización llamados formas normales.

- Primera forma normal.
- Segunda forma normal.
- Tercera forma norma.

- Cuarta forma normal.
- Quinta forma normal.

DCD (*Data Context Diagram*). Es un diagrama de alto nivel que captura al sistema y sus interfaces externas a diferentes entidades o sistemas. Se identifica el límite entre el sistema y su ambiente con sus entradas y salidas. Con un solo proceso se muestra la transformación realizadas por el sistema. La notación de DCD incluye cajas que representan a las entidades externas al sistema, el flujo de datos se representa con flechas y círculos que representan los procesos del sistema que transforman los datos.

DFD (*Data Flow Diagrams*). Es una representación del sistema como una red de procesos funcionales conectados entre ellos por flujos de datos y repositorio de datos. Los procesos transforman los datos. Para abarcar todo el sistema se divide en componentes. Empleando la descomposición funcional es posible ir de una representación de alto nivel a uno de menor nivel o más detallado.

DFD utiliza círculos para representar a los procesos, flechas para los flujos de datos y dos líneas paralelas para los repositorios de datos. En este tipo de diagrama sólo representa los repositorios externos pero no las entidades.

PSPEC (*Process Specifications*). Es una descripción de un proceso o función a un nivel de descomposición detallado. Captura los pasos que son necesarios para realizar cada proceso, en los que se manejan las entradas y las salidas. Para ello se puede realizar seudocódigo, tablas de decisión, ecuaciones u otras gráficas.

CCD (*Control Context Diagram*). Cuenta con sólo un proceso, muestra las entradas y salidas que se realizan con las entidades externas, detallándose hasta finalizar en especificaciones. La diferencia con respecto a otros diagramas es el cambio de estado del sistema que activa y desactiva las transformaciones. Los símbolos que emplea el CCD son los mismos a un DCD,

con la excepción del control de datos que se representa de flechas de línea no continua para distinguirla de otros datos.

CFD (*Control Flow Diagrams*). Es un diagrama semejante al DFD, con la variación de que se modela el flujo de control de datos y no flujo de datos. Comparte las mismas propiedades del DFD, así como también los procesos y repositorio de datos. La diferencia radica en la presentación del flujo de control a través de líneas no continuas y el símbolo de una barra que indica la especificación de control (CSPEC) que puede cambiar el estado del sistema. CFD no muestra las entidades externas.

CSPEC (*Control Specifications*). Contiene la información adicional relacionada a los aspectos de control. El propósito primario es modificar la respuesta del procesamiento de datos de acuerdo a las condiciones pasadas y presentes tanto dentro y fuera del sistema. Es importante identificar todos los posibles estados del sistema para representarlos por modelos finitos de estados que describen el comportamiento del sistema. STD (*State Transition Diagram*) representa el comportamiento de un sistema para que sean observables los eventos y los estados que causan el cambio de estado de un sistema. STD representa a los CSPEC en donde las condiciones son representadas por entradas de control de flujo y las acciones son representadas por flujos de control de salida.

5.3 Orientado a objetos

Los modelos orientados a objetos se encuentran enfocados como soporte a la programación orientada a objetos. Llevan a cabo la representación del sistema de software con sus elementos y sus relaciones.

Un modelo de análisis orientado a objetos se basa en los siguientes principios:

- La información dominio es modelada

- Las funciones son descritas
- El comportamiento es representado
- Los modelos de datos, funciones y comportamiento son divididos para presentar los detalles
- Los modelos anteriores representan la esencia del problema y los modelos posteriores proporcionan el detalle de la implementación

En el análisis se definen todas las clases que son relevantes para solucionar el problema junto con las operaciones y atributos asociados, sus relaciones y el comportamiento. Para lograr lo anterior es importante realizar las siguientes tareas:

1. Los requerimientos básicos del usuario deben de ser comunicados entre el cliente y el ingeniero de software.
2. Las clases deben de ser identificadas (atributos y métodos)
3. Una jerarquía de clase debe de ser especificada
4. Las relaciones entre objetos deberán de ser representadas
5. El comportamiento del objeto deberá de ser modelado
6. Las tareas del 1 al 5 se aplican de manera iterativa hasta que el modelo es completado

Grady Booch, James Rumbaugh e Ivar Jacobson colaboraron para generar un método unificado para el análisis y diseño orientado a objetos (UML). Este conjunto de diagramas permiten ver a un sistema desde diferentes perspectivas.

El análisis orientado a objetos en un nivel de abstracción de aplicación el modelo de objetos se enfoca en los requerimientos específicos del cliente. El proceso comienza con definir la manera en la que se representa el sistema orientado a objetos. Se documentan las clases con sus atributos y operaciones. Se proporciona una vista inicial de la colaboración entre objetos. Posteriormente se clasifican los objetos y se genera una jerarquía de clase. Se

pueden emplear subsistemas o paquetes para encapsular objetos relacionados. El modelo de relación de objetos muestra la conexión que existe entre objetos. El modelo de comportamiento de objetos proporciona el comportamiento de un objeto en particular y el de todos los objetos. Para realizar las representaciones mencionadas anteriormente el analista puede hacer uso de varios de los diagramas de UML (*Unified Modeling Language*).

5.4 UML

UML (*Unified Modeling Language*) es un lenguaje gráfico para visualizar, especificar, construir y documentar los artefactos de un sistema de software. UML proporciona una forma estandarizada de representar los planos de un sistema, incluyendo los procesos del negocio y funciones del sistema, así como cuestiones concretas del lenguaje de programación, estructura de base de datos y componentes reutilizables de software.

UML no se encuentra sujeto a una metodología en específico, por lo que es posible adoptarla de acuerdo a las especificaciones de la metodología que se vaya a emplear.

UML 2 define trece diagramas, los cuales se encuentran agrupados de la siguiente manera:

Diagramas de estructura

- Diagrama de clase. Clases, interfases, tipos y otras relaciones entre ellos.
- Diagrama de objetos. Instancias de objetos de las clases definidas en los diagramas de clase en configuraciones importantes del sistema.
- Diagrama de componentes. Componentes importantes del sistema y las interfases que utilizan para comunicarse con los otros.

- Diagrama de estructura compuesta. Los elementos de una clase o componentes y la descripción de relaciones de clases dentro de un contexto.
- Diagrama de paquetes. Organización jerárquica de grupos de clases y componentes.
- Diagrama de despliegue. Como se despliega el sistema finalmente en una situación real.

Diagramas de comportamiento

- Diagrama de casos de uso. Interacción entre el sistema y el usuario u otros sistemas externos.
- Diagrama de actividades. Actividades secuenciales o paralelas dentro del sistema.
- Diagrama de estados. Estado de un objeto durante su tiempo de vida y los eventos que pueden cambiar su estado.

Diagramas de interacción

- Diagrama de secuencias. Interacción entre objetos en donde el orden de las interacciones es importante.
- Diagrama de comunicación. Las formas en las cuales los objetos interactúan y las conexiones necesarias para esas interacciones.
- Diagrama de tiempos. Interacciones entre objetos en donde el tiempo es importante.
- Diagrama de descripción de interacción. Se emplea para conjuntar los diagramas de secuencia, comunicación y tiempo para capturar una importante interacción dentro del sistema.

Bibliografía del tema 5

Futrell, Robert T., *Quality software project management*. Nueva Jersey, Prentice Hall, c2002.

Miles, Russell, *Learning UML 2.0*. California, O'Reilly, 2006.

Pressman, Roger, *Software Engineering: A practitioner's approach*. Nueva York, McGraw-Hill, 2001.

Salvarredy, Julián Raúl. *Gestión económica y financiera de proyectos*. Buenos Aires, Omicron System, c2003.

Páginas electrónicas de referencia

http://www.uml.org/

Actividades de aprendizaje

A.5.1. Elaborar una lista con cinco riesgos técnicos comunes en un proyecto de desarrollo de software con las actividades para contrarrestar cada uno de ellos.

A.5.2. Investigar dos modelos de análisis adicionales que pueden utilizarse en el esquema estructurado.

A.5.3. Investigar dos modelos de análisis adicionales que puede utilizarse en el esquema orientado a objetos.

A.5.4. Investigar dos herramientas de software que pueden utilizarse para el análisis de software.

A.5.5. Elaborar un cuadro de referencia con la simbología de cada uno de los modelos presentados en el tema.

Cuestionario de autoevaluación

1. ¿Entre qué fases del desarrollo de software se encuentra el análisis de software?
2. ¿Qué es un estudio de factibilidad?
3. ¿Cuáles son los elementos a considerar en un estudio de factibilidad?
4. ¿En qué consiste el análisis de software?
5. ¿Cuál es la importancia del análisis de software?
6. ¿Qué actividades son importantes de realizar en un análisis de software?
7. ¿Qué tan compatibles son las técnicas de análisis entre la perspectiva estructurada y la orientada a objetos?
8. ¿De qué forma se puede aplicar UML al análisis de software?
9. ¿Cómo se encuentran divididos los diagramas de UML?
10. Viendo al análisis de software como un proceso ¿Cuáles serían las entradas y las salidas?

Examen de autoevaluación

Instrucciones. Relacione las dos columnas colocando la letra de la columna derecha dentro de los paréntesis de la columna izquierda

1.	( ) Interacción entre el sistema y el usuario u otros sistemas externos.	a) ERD
2.	( ) Lenguaje gráfico para visualizar, especificar, construir y documentar los artefactos de software.	b) Diagramas de clases
3.	( ) Actividades secuenciales o paralelas dentro del sistema.	c) Escenarios
4.	( ) Proporcionan los elementos para decidir la conveniencia de realizar un proyecto considerando elementos internos y externos.	d) DFD
5.	( ) Este modelo permite identificar los objetos de datos y sus relaciones a través de una notación gráfica.	e) Casos de uso
6.	( ) Es un diagrama semejante al DFD, con la variación de que se modela el flujo de control de datos y no flujo de datos.	f) CFD
7.	( ) Es un diagrama de alto nivel que captura al sistema y sus interfaces externas a diferentes entidades o sistemas.	g) UML
8.	( ) Clases, interfaces, tipos y otras relaciones entre ellos.	h) Factibilidad
9.	( ) Es una representación del sistema como una red de procesos funcionales conectados entre ellos por flujos de datos y repositorio de datos.	i) DCD
10.	( ) Herramienta de análisis que puede presentar las mejores condiciones como en el peor de los casos.	j) Diagrama de actividades

TEMA 6. DISEÑO DE SOFTWARE

Objetivo particular

Con este tema el estudiante reconocerá las diferentes técnicas en el diseño de software.

Temario detallado

6.1 Estudios de factibilidad

6.2 Estructurado

6.3 Orientado a objetos

6.4 UML

Introducción

En el diseño de software se genera una representación del software que se va a desarrollar. Las diferentes técnicas de diseño responden a la necesidad de personificar al software en cada una de sus perspectivas. Dicha representación ayuda a clarificar los elementos del software para diferentes niveles y roles que intervienen.

6.1 Estudios de factibilidad

Este punto ya fue desarrollado anteriormente en el subtema 5.1.

6.2 Estructurado

El diseño se enfoca principalmente en las siguientes áreas: datos, arquitectura, interfaces y componentes. Estos niveles de diseño parten del modelo de requerimientos. Debido a la complejidad que implica desarrollar software es

importante contar con el diseño que permita construir adecuadamente los componentes necesarios.
El modelo de análisis proporciona los elementos necesarios para crear los cuatro modelos necesarios para realizar una especificación de diseño.

Diseño de datos. Transforma el modelo de información en estructuras de datos que serán requeridas en para implementar el software. Parte del diseño de datos se realiza en paralelo con el diseño de arquitectura de software. El diseño más a detalle de los datos se realiza a nivel componente.

Diseño de arquitectura. Define las relaciones entre elementos estructurales mayores del software.

Diseño de interfaces. Describe la manera en la que el software se comunica consigo mismo, con otros sistemas y con los humanos. Esto implica flujo de información y los tipos de comportamiento.

Diseño de componentes. Transforma los elementos estructurales en descripción de procedimientos de componentes de software.

En el proceso de diseño se realiza la transformación de los requerimientos en un plano de construcción del software. Es una secuencia de pasos que posibilitan al diseñador describir todos los aspectos del software que se pretende construir.

El modelo de diseño es una representación de alto nivel que se va depurando gradualmente hasta proveer una guía de construcción detallada.

Existen principios de diseño que pueden emplearse durante el proceso:

- El proceso de diseño no debe de tener una visión cerrada
- El diseño debe de ser rastreable para el modelo de análisis

- El diseño no debe de ser una reinvención de las cosas
- El diseño debe de minimizar la distancia intelectual entre el software y el problema
- El diseño debe de mostrar uniformidad e integración
- El diseño debe de estar estructurado para ajustarse a los cambios
- El diseño debe de estar estructurado para adecuarse a situaciones inusuales
- Diseño no es generar código, generar código no es diseñar
- El diseño debe de ser evaluado en calidad cuando se está generando y no al final
- El diseño debe de ser revisado para minimizar los errores de significado

A continuación se presentan algunos diagramas que permiten la asimilación de los requerimientos a un nivel de construcción de software. Muchos de ellos utilizan diagramas que previamente fueron elaborados en el análisis de requerimientos.

ACD (*Architecture Context Diagrams*). Establece la información límite entre el sistema y su entorno. Muestra la asignación física de los requerimientos del modelo de datos (DFD) y control de procesamiento (CFD). Establece procesos del modelo de requerimientos a módulos físicos del sistema y establece la relación entre ellos. Es un diagrama de representación de alto nivel. Los elementos que intervienen generan un módulo de arquitectura.

AFD (*Architecture Flow Diagram*). Los procesos del DFD son reorganizados de acuerdo a propiedades físicas. Es una representación en red de la configuración física de un sistema. Muestra la asignación de los requerimientos del modelo de datos y de procesamiento de control, así como del flujo entre entidades físicas. Que realizarán actividades asignadas. Esto significa una división física del sistema en piezas y flujos entre ellas. La definición de módulos físicos agrega más perspectivas a las perspectivas funcionales y de control de procesamiento.

Structure Charts. Organiza los sistemas divididos en cajas negras jerarquizadas. Las cajas negras: indican que el usuario conoce las funciones pero no las operaciones internas, reducen la complejidad porque ocultan todo aquello que no se necesita saber, reciben entradas y proporcionan salidas. Esto facilita su construcción, prueba, corrección, entendimiento y modificación. La distribución es una secuencia vertical temporal. Se encuentra constituido por módulos, cada módulo es un conjunto de problemas establecidos con entradas, salidas, funciones y datos internos. Los módulos superiores son los que se ejecutan antes que aquellos que se encuentran más abajo.

Chapin Charts. Es un modelo de diseño a nivel detallado. También conocido como diagramas Nassi-Schneiderman, describe los procedimientos para recibir, procesar y transmitir información. Proveen la lógica necesaria para que los programadores inmediatamente escriban programas estructurados. Se utilizan cinco estructuras básicas (secuencia, if-then-else, do-while, do-until y do-case) que permiten su traducción en código de computadora. Su función es igual a la de los demás diagramas. Su lectura se realiza como al igual que una página impresa. Tiene un flujo jerárquico de instrucciones y opciones que mantiene la lógica en los procedimientos.

Architecture Interconnect Diagrams. Representa los canales de comunicación que existen entre los módulos de arquitectura. Muestra el medio físico por el cual los módulos se comunican. Captura las características de los canales por los cuales fluirá la información entre los módulos. Esta parte debe de ir fuertemente ligada con el modelo de análisis.

6.3 Orientado a objetos

El diseño orientado a objetos requiere de una definición multicapa de la arquitectura del software, especificación de subsistemas, una descripción de objetos y la descripción de los mecanismos de comunicación que permiten el flujo de datos entre capas, subsistemas y objetos.

El diseño orientado a objetos se rige bajo cuatro importantes principios: abstracción, ocultamiento, independencia funcional y modularidad.

Existen cuatro capas en el diseño:

- Capa de subsistema. Se encarga de cada uno de los subsistemas que permiten alcanzar los requerimientos establecidos e implementar la infraestructura técnica que los soporta.
- Capa de objetos y clases. Contiene la jerarquía de clases que son utilizadas empleando generalizaciones y estableciendo objetivos específicos. Se genera la representación de cada objeto.
- Capa de mensajes. Tiene los detalles que hacen que cada objeto se comunique. Aquí se establecen las interfaces internas y externas del sistema.
- Capa de responsabilidades. Contiene la estructura de datos y diseño de algoritmos para todos los atributos y operaciones de cada objeto.

Independientemente del modelo de diseño que se utilice se puede observar que se encuentran presentes los siguientes pasos:

1. Describe cada subsistema para localizar procesos y tareas
2. Selecciona una estrategia de diseño para administrar la implementación de datos, soporte de interfaces y administración de tareas
3. Diseña un mecanismo de control apropiado para el sistema
4. Realiza el diseño de objetos para crear una representación de procedimientos de cada operación y estructura de datos para los atributos de las clases
5. Desarrolla el diseño de mensajes empleando colaboraciones entre objetos y relaciones de objetos
6. Crea el modelo de mensajes
7. Revisar el modelo de diseño

Se puede hacer uso de algunos diagramas para realizar la representación del diseño de software, algunos de ellos se pueden encontrar en la propuesta que hace UML. A continuación se presentan algunas opciones.

Interaction Diagrams—Collaboration Diagrams. Gráficas con énfasis en objetos y ligas de objetos. Muestra como un caso de uso es realizado en términos de cooperación de objetos, definido por clases dentro de la entidad, puede ser especificado con una colaboración. El caso de uso de una entidad puede ser refinado a un conjunto de casos de uso de elementos contenidos en la entidad. La interacción de esos casos de uso subordinados puede ser expresada en una colaboración.

Interaction Diagrams—Sequence Diagrams. La gráfica muestra una secuencia temporal, con especial interés en el orden de los mensajes.

State Transition Diagrams. Representa los estados de una clase y los estados que pasa la instancia de una clase.

6.4 UML

Este punto ya fue desarrollado en el subtema 5.4.

Bibliografía del tema 6

Futrell, Robert T., *Quality software project management*. Nueva Jersey, Prentice Hall, c2002.

Pressman, Roger. *Software Engineering: A practitioner's approach*. Nueva York, McGraw-Hill, 2001.

Actividades de aprendizaje

A.6.1. Realizar un cuadro de correspondencia entre los modelos vistos de análisis y diseño de software.

A.6.2. Investigar dos modelos adicionales que puedan utilizarse en el esquema estructurado.

A.6.3. Investigar dos modelos adicionales que puedan utilizarse en el esquema estructurado.

A.6.4. Investigar dos opciones de herramientas de software que puedan emplearse en el diseño.

A.6.5. Elaborar cuadro de referencia con la simbología de cada uno de los modelos presentados en el tema.

Cuestionario de autoevaluación

1. ¿Qué es el diseño de software?
2. ¿Cuál es la importancia de realiza el diseño de software?
3. ¿De qué forma se mejora la comprensión del software con el diseño?
4. ¿Por qué existen diferentes técnicas de diagramación?
5. ¿Cuáles son los principios de la programación orientada a objetos?
6. ¿Es necesario aplicar todas las técnicas de diseño para asegurar la calidad del software?
7. ¿Qué diagramas de UML pueden aplicarse al diseño de software?
8. ¿Cuáles son los principios de diseño de la perspectiva estructurada?
9. ¿Cuáles son cuatro capas del diseño de la perspectiva orientada a objetos?
10. ¿Bajo qué condiciones sería recomendable realizar el diseño del software?

Examen de autoevaluación

Instrucciones. Relacione las dos columnas colocando la letra de la columna derecha dentro de los paréntesis de la columna izquierda

1.	( ) Es un modelo de diseño a nivel detallado. También conocido como diagramas Nassi-Schneiderman.	a) AFD
2.	( ) Organiza los sistemas que han sido divididos en cajas negras jerarquizadas.	b) Chapin Charts
3.	( ) Tienen un énfasis en objetos y ligas de objetos.	c) Architecture Interconnect Diagrams
4.	( ) Nivel de diseño estructurado	d) Interaction Diagrams
5.	( ) Establece la información límite entre el sistema y su entorno.	e) State Transition Diagrams
6.	( ) Nivel de diseño orientado a objetos.	f) Structure Charts
7.	( ) Muestra una secuencia temporal, con especial interés en el orden de los mensajes.	g) Datos
8.	( ) Representa los estados de una clase y los estados que pasa la instancia de una clase.	h) Subsistema
9.	( ) Representa los canales de comunicación que existen entre los módulos de arquitectura.	i) Collaboration Diagrams
10.	( ) Los procesos del DFD son reorganizados de acuerdo a propiedades físicas	j) ACD

TEMA 7. VERIFICACIÓN Y VALIDACIÓN

Objetivo particular

En este tema el alumno conocerá las actividades que le permitirán asegurar la calidad del software durante el proceso de desarrollo.

Temario detallado

7.1 Pruebas

7.2 Puntos por función

Introducción

Durante el proceso de desarrollo de software es necesario llevar a cabo actividades que permitan asegurarnos que los productos que se están generando se estén realizando bien y correctamente. Cada desarrollo deberá de identificar los elementos que considere más importantes para que sean sometidos por estas actividades.

7.1 Pruebas

Probar es una operación o acción técnica que consiste en la determinación de una o más características de un producto, proceso o servicio, de acuerdo a un conjunto de especificaciones dadas.

El ciclo de vida de las pruebas de software involucra pruebas durante todo el proceso de desarrollo. Para poder emplear este ciclo de vida es necesario que se cuente con un proceso de desarrollo definido, pues éste definirá las fases y los entregables que serán sometidos a prueba.

De manera general se presentan algunas de las actividades que se realizan en cada fase del ciclo de vida.

Fase del ciclo de vida	Actividades
Requerimientos	• Determinar la verificación • Determinar la adecuación de los requerimientos • Generar pruebas funcionales de datos • Determinar la consistencia del diseño con los requerimientos
Diseño	• Determinar la adecuación del diseño • Generar pruebas funcionales y estructurales de datos • Determinar la consistencia con el diseño
Desarrollo/Construcción	• Determinar la adecuación de la implementación • Generar pruebas estructurales y funcionales de datos para los programas
Pruebas	• Pruebas de sistema
Instalación	• Colocar las pruebas de sistema en producción
Mantenimiento	• Modificar y volver a probar

A continuación se presentan las definiciones más comunes que se emplean dentro del área de pruebas de software. Durante el ciclo de vida se aplican varios tipos de pruebas.

Categorías de las pruebas:

- Unitarias. Pruebas aplicadas a un módulo aislado o unidad de código.
- Integración. Pruebas desarrolladas a un grupo de módulos para asegurar que los datos y el control sean intercambiados adecuadamente entre módulos.
- Sistema. Una combinación predeterminada de pruebas para asegurarse de que el sistema alcanza los requerimientos.
- Aceptación. Prueba que asegura que el sistema alcanza las necesidades de la organización y los del usuario o cliente final.
- Regresión. Son pruebas realizadas después de que se han realizados las modificaciones para asegurar de que no se hayan ingresado cambios indeseados.

Error. La gente comete errores o equivocaciones. Los errores tienden a propagarse; un error en la etapa de requerimientos puede acrecentarse durante el diseño y aumentar aún más durante la codificación.

Defecto. El defecto es el resultado de un error. De forma más precisa podría decirse que un defecto es la representación de un error, representación que queda expresado en forma de texto, diagrama de flujo, código fuente, etc. Los sinónimos utilizados son bug y falta. Los defectos pueden ser difíciles de detectar. Cuando se tiene una omisión el defecto consiste en que algo que debería estar presente no lo está. Por lo que podría decirse que existen dos tipos de defectos: por omisión y por comisión. El defecto por comisión tiene que ver con una representación incorrecta de lo solicitado. El defecto por comisión ocurre al momento de ingresar información. De estas dos el defecto por omisión es más difícil de detectar y resolver.

Falla. La falla aparece al momento en que se ejecuta un defecto. Este por lo tanto sólo se da para una representación ejecutable y en defectos por comisión. Entonces, ¿qué hacer con los defectos que nunca se ejecutan o no

se ejecutan por mucho tiempo? Una opción aceptable es realizar buenas revisiones, que pueden prevenir fallas y encontrar defectos por omisión.

Casos de prueba. Estructura que permita probar todos los procesos posibles del sistema para encontrar sus inadecuaciones con los requerimientos y normas establecidas, con el menor esfuerzo y tiempo posibles. Es un conjunto de entradas, condiciones de ejecución y resultados esperados desarrollados para un objetivo particular como, por ejemplo, ejercitar un camino concreto de un programa o verificar el cumplimiento de un determinado requisito.

Verificación. Son todas las actividades a través del ciclo de vida que aseguran que los productos internos en el proceso cumplan con sus especificaciones.

Validación. Asegura que el producto final cumple con las especificaciones establecidas.

Pruebas estáticas. Son las realizadas sin la ejecución del código.

Pruebas dinámicas. Son realizadas con la ejecución del código.

Pruebas funcionales. Pruebas a partir de los requerimientos (qué es lo que se supone que hace). Se asegura de que los requerimientos sean cumplidos adecuadamente por el software.

Pruebas estructurales. Pruebas sobre la estructura del sistema (cómo el sistema fue implementado). Se asegura de que el producto diseñado es estructuralmente lógico.

Pruebas manuales. Son pruebas realizadas por personas.

Pruebas automatizadas. Son pruebas realizadas por la computadora.

Pruebas de caja negra. Pruebas basadas en especificaciones externas sin el conocimiento previo de cómo se encuentra construido.

Pruebas de caja blanca. Pruebas basadas en el conocimiento de la estructura interna y lógica.

Dentro de las pruebas existen técnicas que hacen posible reducir los riesgos inherentes a los sistemas. Una técnica de pruebas es un proceso para asegurar que algún aspecto del software o unidad funcione adecuadamente. A partir de las técnicas se definen las herramientas, el cual es el vehículo para llevar a cabo el proceso de pruebas. Algunas técnicas de pruebas son:

- Pruebas de estrés
- Pruebas de ejecución
- Pruebas de restauración
- Pruebas de operaciones
- Pruebas de conformidad
- Pruebas de seguridad
- Pruebas de requerimientos
- Pruebas de regresión
- Pruebas de manejo de errores
- Pruebas de soporte manual
- Pruebas entre sistemas
- Pruebas de control
- Pruebas en paralelo
- Pruebas unitarias

A continuación se presentan un listado de herramientas que pueden emplearse en conjunto con alguna de las técnicas de pruebas mencionadas anteriormente.

1. Criterios de aceptación. Consiste en proporcionar estándares que deben de ser alcanzados por el sistema para que sean aceptables para el usuario.
2. Análisis de valor límite. Divide al sistema de arriba hacia abajo en segmentos lógicos y con ello limita las pruebas con límites en cada segmento.
3. Gráfica causa efecto. Intenta mostrar los efectos de cada evento procesado con el fin de categorizar eventos por el efecto que ocurrirá como resultado del procesamiento.
4. *Checklist*. Una serie de preguntas diseñadas para ser utilizadas en la revisión en una determinada área o función.
5. Comparación de código. Identifica la diferencia entre dos versiones del mismo programa.
6. Análisis basado en compilador. Detecta errores de un programa durante el proceso de compilación.
7. Métrica basada en complejidad. Utiliza relaciones para identificar el grado de complejidad de procesamiento del programa.
8. Confirmación / Examinación. Verifica que las condiciones han o no han ocurrido.
9. Análisis de control de flujo. Identifica inconsistencias de procesamiento para identificar problemas lógicos dentro de los programas.
10. *Correctness Proof*. Involucra un conjunto de sentencias o hipótesis que definen lo correcto del procesamiento. Esas hipótesis son probadas para determinar si las aplicaciones desempeñan el procesamiento de acuerdo con las sentencias.
11. Métricas basadas en cobertura. Utiliza relaciones matemáticas para mostrar qué porcentaje de las aplicaciones han sido cubiertas por el proceso de pruebas. La métrica resultante puede ser utilizada para predecir la efectividad del proceso de pruebas.
12. Diccionario de datos. Genera datos de prueba para verificar la validación de los programas de validación de programas basados en los datos contenidos en el diccionario.

13. Análisis de flujo de datos. Un método para asegurar que los datos usados por el programa han sido definidos adecuadamente y los datos definidos sean utilizados adecuadamente.
14. Pruebas funcionales basadas en diseño. Reconoce qué funciones dentro de una aplicación son necesarias para los requerimientos. Este proceso identifica las funciones basadas en diseño para propósitos de pruebas.
15. Revisiones del diseño. Revisiones conducidas durante el proceso de desarrollo normalmente en concordancia con la metodología de desarrollo. El objetivo primario es asegurar la conformidad con la metodología de desarrollo.
16. Revisiones de escritorio. Proporcionan una evaluación realizada por el programador o analista al elemento de programa lógico antes de ser codificado o diseñado.
17. Prueba de desastre. Son simulaciones de fallas para determinar si el sistema puede ser correctamente recuperado después de la falla.
18. Conjeturas de error. Utiliza la experiencia o el juicio de la gente para determinar a través de conjeturas los errores que muy probablemente puedan ocurrir y a partir de ellos probar para asegurarnos si el sistema puede manejar esas condiciones.
19. Especificaciones ejecutables. Requiere de una interpretación del sistema de alto nivel para escribir las especificaciones. Las especificaciones compiladas tienen menos detalles y precisión a comparación de los programas finales implementados, pero son suficientes para evaluar que las especificaciones se encuentran completas y con el adecuado funcionamiento.
20. Pruebas exhaustivas. Se intenta crear suficientes pruebas para evaluar cada camino y condición de la aplicación.
21. Indagación de hechos. Desempeña los pasos necesarios para obtener hechos de apoyo al proceso de pruebas.
22. Diagrama de flujo. Representación gráfica de la lógica y el flujo de datos de una aplicación.

23. Inspecciones. Revisión paso por paso del producto en el que cada paso es revisado contra una lista con criterios predeterminados.
24. Instrumentación. Mide la función de la estructura de un sistema utilizando contadores u otro instrumento de monitoreo.
25. Facilidad integrada. Permite la introducción de datos de prueba en el ambiente de producción, de forma que las aplicaciones pueden ser probadas al mismo tiempo que se encuentran ejecutándose en producción.
26. Mapeo. Identifica qué partes de un programa son ejecutadas durante una prueba y su frecuencia.
27. Modelado. Método de simulación de las funciones de la aplicación y/o ambiente para determinar si las especificaciones de diseño alcanzan los objetivos del sistema.
28. Operación en paralelo. Se corre la vieja y la nueva versión a la vez para identificar las diferencias entre los dos procesos.
29. Simulación en paralelo. Aproxima los resultados esperados de procesamiento para simular el proceso y determinar si los resultados son razonables.
30. Revisión en pares. Provee una evaluación en pares de la eficiencia, estilo, seguimiento de los estándares. Del producto diseñado para mejorar la calidad del producto.
31. Matriz de riesgos. Se produce una matriz que muestra las relaciones entre los riesgos del sistema, el segmento en el sistema en donde ocurre el riesgo y la ausencia o ausencia de controles para reducir ese riesgo.
32. SCARF (por sus siglas en inglés) Sistema de control de auditoría y revisión de archivos. Construye un historial de los errores potenciales con la finalidad de comparar problemas en una unidad en un periodo de tiempo y/o compararlo contra otras unidades.
33. Marcador. Identifica áreas en la aplicación que requieren pruebas, a través de un criterio para relacionarlas a problemas.
34. Foto. Muestra el contenido almacenado de una computadora en puntos específicos durante el proceso.

35. Ejecución simbólica. Identifica caminos de procesamiento para probar programas con símbolos y no con datos de prueba.
36. Registros del sistema. Proporciona un rastro y monitorea eventos en un ambiente controlado por un software.
37. Datos de prueba. Crea transacciones para que se utilicen en determinadas funciones de un sistema de computadora.
38. Generados de datos de prueba. Proporciona pruebas de transacciones basadas en parámetros que necesitan probarse.
39. Seguimiento. Sigue y enlista el flujo de procesamiento y datos basados en búsquedas.
40. Programa utilitario. Analiza e imprime el resultado de una prueba a través del uso de un programa de propósito general.
41. Prueba de volumen. Identifica las restricciones del sistema y crea grandes volúmenes de transacciones diseñados para exceder esos límites.
42. *Walk-throughs*. Lleva al equipo de pruebas a realizar una simulación manual del producto utilizando pruebas de transacción. Se cuestiona al autor del elemento de software y se le solicita que explique su funcionamiento.

7.2 Puntos por función

En 1979 fue publicada la métrica *Function Point* por A. J. Albrecht de IBM. Son el resultado de la utilización de la relación empírica basada en las mediciones contables directas del software y la evaluación de la de complejidad del software.

Los puntos por función son computados para completar una tabla que contiene cinco características que son determinadas y contadas para posteriormente colocarlas en su lugar correspondiente dentro de la tabla. Los valores de cada característica se definidos de la siguiente forma:

- Número de entradas del usuario. Se cuenta cada pantalla o ventana que provee una entrada al sistema. Si una ventana tiene varios campos, cada campo es calculada como una entrada. Si la ventana provee varias fuentes que no son enviadas al mismo tiempo, cada fuente es calculada como una entrada.
- Número de salidas del usuario. Se cuenta cada salida que provee información al usuario. Se incluyen reportes, mensajes de error, ventanas, etc. Los campos no son calculados de manera individual.
- Número de consultas del usuario. Una consulta de usuario es una entrada directa de un usuario que obtiene una respuesta directa.
- Número de archivos. Un grupo lógico de datos permanentes es considerado un archivo. Se incluyen tablas de bases de datos, archivos de configuración, archivos de ayuda, etc.
- Número de interfaces externas. Se incluyen todas las interfaces a otros sistemas. Las interfaces deben de ser máquina a máquina. Si interviene el factor humano entonces se considera una entrada o consulta de usuario.

Una vez que se han contabilizado todas características entonces se agrega un valor de complejidad a cada uno. Cada organización determina un criterio para determinar si es simple, promedio o complejo.

Para computar los puntos por función se emplea la siguiente relación:

$$FP = \text{Conteo total} \times [0.65 + 0.01 \times \sum (F_i)]$$

En donde:

- FP es la suma total de todas las características después de aplicarse el estimado de dificultad.
- F_i (i = 1 a 14) es la suma de los valores de ajuste de complejidad. Se basa en catorce preguntas. Cada pregunta tiene una escala del 0 al 5.

- 0.65 y 0.01 son números basados en evidencia empírica.

Las preguntas que se aplican para generar el ajuste de complejidad son:

1. ¿El sistema requiere una restauración y respaldo confiable?
2. ¿Las comunicaciones de datos son requeridos?
3. ¿Existen funciones de procesamiento distribuido?
4. ¿El desempeño es crítico?
5. ¿El sistema correrá dentro de un ambiente existente y de procesamiento pesado?
6. ¿El sistema requiere de entrada de datos en línea?
7. ¿Los datos ingresados en línea requieren la transacción de entradas para realizar múltiples pantallas u operaciones?
8. ¿Los archivos maestros se actualizan en línea?
9. ¿Las entradas, salidas, archivos, o consultas son complejos?
10. ¿El procesamiento interno es complejo?
11. ¿El diseño de código es reutilizable?
12. ¿Las conversiones y la instalación son incluidas en el diseño?
13. ¿El sistema es diseñado para múltiples instalaciones en diferentes organizaciones?
14. ¿La aplicación es diseñada para facilitar el cambio y facilidad de uso para el usuario?

La escala establecida para responder cada una de las preguntas anteriores es:

- 0: sin influencia
- 1: secundario
- 2: moderado
- 3: promedio
- 4: significante
- 5: esencial

Una vez que se ha realizado los cálculos se pueden emplear de manera que se puede medir la productividad, calidad y otros atributos como:

- Errores por FP
- Defectos por FP
- Costo por FP
- FP por persona al mes
- Páginas de documentación por FP

Parámetro de medición	**Conteo**		**Simple**	**Promedio**	**Complejo**		**Resultado**
Entradas de usuario		X	3	4	6	=	
Salidas de usuario		X	4	5	7	=	
Consultas de usuario		X	3	4	6	=	
Archivos		X	7	10	15	=	
Interfaces externas		X	5	7	10	=	
Conteo total							

Bibliografía del tema 7

Pressman, Roger. *Software Engineering: A practitioner's approach*. Nueva York, McGraw Hill, 2001.

Perry, William E. *Effective methods for software testing*. Nueva York, Wiley, 1995.

Jorgensen, Paul C. *Software Testing: A Craftsman's Approach*. Boca Raton, CRC, 2002.

Beizer, Boris. *Software Testing Techniques*. Nueva York, Van Nostrand Reinhold, 1983.

Páginas de referencia

http://www.spr.com/

http://www.ifpug.org/

Actividades de aprendizaje

A.7.1. Investigar una herramienta de software que pueda utilizarse para realizar pruebas funcionales.

A.7.2. Conseguir el estándar IEEE 829 y realizar un diseño de formatos basados en el estándar.

A.7.3. Elaborar un checklist que contenga los elementos a revisar en una prueba de conformidad.

A.7.4. Elaborar una matriz con las pruebas de software en cada una de las fases de desarrollo de software.

A.7.5. Elaborar los formatos necesarios para aplicar los puntos por función.

Cuestionario de autoevaluación

1. ¿Cuál es la diferencia entre verificar y validar?
2. ¿Cuál es la importancia de verificar y validar dentro del desarrollo de software?
3. ¿Es posible probar todo en el software?
4. ¿Cuáles serían las limitantes para utilizar los puntos por función?
5. ¿Con qué finalidad realizamos las pruebas de software?

6. ¿En qué momento se deben de realizar las pruebas de software?
7. ¿Los puntos por función se entran en la categoría de verificación o de validación?
8. ¿Qué otras actividades se pueden realizar para verificar o validar?
9. ¿Cuándo es recomendable dejar de realizar las pruebas de software?
10. ¿Cuál es la relación entre los puntos por función y las pruebas de software?

Examen de autoevaluación

I. Escribe en el paréntesis A si la descripción corresponde a una herramienta o B si corresponde a una técnica de pruebas de software

Descripción		Tipo de recurso
1.	( ) Checklist	a. Herramienta b. Técnica
2.	( ) Revisión en pares	
3.	( ) Unitarias	
4.	( ) Valor límite	
5.	( ) Stress	

II. Seleccione una de las opciones y escriba la letra correspondiente en el paréntesis de la derecha de cada pregunta

6. Son las pruebas basadas en especificaciones externas sin el conocimiento previo de cómo se encuentra construido ( )

a) Pruebas de convivencia
b) Pruebas de caja blanca
c) Pruebas de caja negra
d) Pruebas de compatibilidad

7. ¿Cuál es la definición de Defecto? ( )

a) Es una omisión de algo que debería de estar y no está o la representación incorrecta de lo que se ha solicitado.
b) Aparece sólo al momento en que se ejecuta.
c) Es el originado por una actividad humana.
d) Es una inconsistencia lógica.

8. ¿Cuál es la definición de Verificación? ( )

a) Asegura que el producto final cumple con las especificaciones establecidas.
b) Se asegura de que el producto diseñado es estructuralmente lógico.
c) Son todas las actividades a través del ciclo de vida que aseguran que los productos internos en el proceso cumplan con sus especificaciones.
d) Es la actividad realizada con la ejecución del programa.

9. ¿Cuántas preguntas comprende el ajuste de complejidad en los puntos por función? ( )

a) 5
b) 6
c) 16
d) 14

10. ¿Es un uso que se le puede dar a los resultados de los puntos por función? ( )

a) Costo por FP
b) Defectos por FP
c) Errores por FP
d) Todas las anteriores

TEMA 8. MANTENIMIENTO DE SOFTWARE

Objetivo particular

Al finalizar el tema el alumno habrá identificado la importancia del mantenimiento de software, así como las actividades involucradas alrededor de esta fase del ciclo de vida de software.

Temario detallado

8.1 De corrección

8.2 De prevención

8.3 De adaptación

8.4 De mejora

Introducción

El mantenimiento de software es una parte dentro del ciclo de vida del software. En esta fase se realizan actualizaciones, mejoras o correcciones que no fueron detectadas en las fases anteriores. Se realizan las actividades necesarias para proporcionar un soporte rentable al software. Según la norma ISO/IEC 14764 son cuatro los tipos de mantenimiento: correctivo, adaptación, mejora y preventivo.

8.1 De corrección

Mantenimiento correctivo. Son las modificaciones necesarias que se realizan al software después de su liberación para corregir problemas encontrados. También se emplea cuando el software no satisface los requerimientos establecidos.

8.2 De prevención

Mantenimiento preventivo. Modificaciones realizadas al software después de su liberación para detectar y corregir defectos potenciales en el software antes de que se conviertan en fallas.

8.3 De adaptación

Mantenimiento de adaptación. Son modificaciones realizadas al software después de su liberación para mantener la utilidad del software ante los cambios del ambiente. Estas modificaciones no se encuentran en el diseño de especificaciones. Se incluyen los nuevos requerimientos que incluyen modificaciones para incluir una nueva interface, sistema o hardware.

8.4 De mejora

Mantenimiento de mejora. Modificaciones realizadas al software después de su liberación para perfeccionar su desempeño y mantenimiento. En él se puede incluir nuevas funcionalidades para el usuario o una ingeniería inversa para crear la documentación faltante o modificar la existente.

Proceso de mantenimiento

El proceso de mantenimiento contiene actividades y tareas específicas que permiten realizar modificaciones al software sin afectar su integridad. El proceso abarca la migración a un nuevo ambiente hasta el retiro del software.

Al proceso se le relacionan seis actividades:

- Proceso de implementación. Se establece el plan y procedimientos que serán utilizados en el proceso de mantenimiento. El plan de mantenimiento debe de ser elaborado al mismo tiempo que se realiza el plan de desarrollo.

- Análisis de problemas y modificaciones. Esta actividad se inicia después de la transición del software y es llamado cada vez que se levantan una necesidad de modificación.
- Implementación de modificaciones. El encargado del mantenimiento desarrolla y prueba las modificaciones realizadas al software.
- Revisión/aceptación. Se asegura que las modificaciones sean correctas conforme a los estándares establecidos y con la metodología correcta.
- Migración. El encargado deberá de establecer las acciones necesarias para llevar a cabo la migración hacia un nuevo ambiente y posteriormente desarrollar y documentar los pasos requeridos para la migración.
- Retiro. Se emplea cuando el software ha alcanzado el final de su vida útil. Se realiza un análisis que permita tomar la decisión de retirar el software. El análisis puede estar basado en una perspectiva económica y puede ser incluida en el plan de retiro. El encargado deberá de determinar las acciones necesarias para realizar el retiro.

Cada una de las actividades cuenta con entradas definidas que son transformadas por las tareas para producir salidas. También se encuentran controles y procesos de soporte identificados.

Es importante establecer el proceso de mantenimiento ya que es muy probable que durante la operación del sistema se encuentren errores que deberán de corregiste y se presentarán nuevas necesidades que requerirán la modificación del software. El objetivo del proceso de mantenimiento es modificar un software ya existente preservando su integridad. Realizando las actividades mencionadas se podrá reducir de manera significativa los riesgos que conlleva las modificaciones al software.

Una estrategia de mantenimiento debería de contener los siguientes elementos:

- Concepto de mantenimiento. Alcance, vista general del proceso, costos, responsables.
- Plan de mantenimiento. Necesidad, roles, recursos, controles, capacitación, actividades y tareas.
- Análisis de recursos. Personal, ambiente y financiero.

Bibliografía del tema 8

ISO/IEEE. “Software Engineering - Software Life Cycle Processes - Maintenance”. IEEE std 14764-2006, 2006.

Actividades de aprendizaje

A.8.1. Diagramar una de las actividades del proceso de mantenimiento de software utilizando UML.

A.8.2. Investigar una herramienta de software que pueda utilizarse en el mantenimiento de software.

A.8.3. Elaborar el formato para el plan de mantenimiento.

A.8.4. Generar un listado con los recursos necesarios en un proceso de mantenimiento de software.

A,8.5. Conseguir el estándar ISO/IEC 9126 y extraer las métricas relacionadas con el mantenimiento de software.

Cuestionario de autoevaluación

1. ¿Por qué el mantenimiento es considerada una actividad que forma parte del desarrollo de software?
2. ¿Las modificaciones producidas durante el desarrollo de software son consideradas como parte del mantenimiento de software?
3. ¿Cuál es la importancia del mantenimiento de software?
4. ¿Es posible realizar actividades de mantenimiento sin los antecedentes previos del proyecto de software?
5. ¿Cuál es la diferencia entre el mantenimiento de software y un nuevo ciclo de desarrollo de software?
6. ¿En qué momento se establece el proceso de mantenimiento de software?
7. ¿Qué eventos pueden dar inicio a las actividades de mantenimiento de software?
8. ¿Quién es el encargado de llevar el registro de los cambios realizados por las actividades de mantenimiento?
9. ¿En todos los proyectos de desarrollo de software se encuentran presentes las actividades de mantenimiento?
10. ¿En qué momento se finaliza el proceso de mantenimiento de software?

Examen de autoevaluación

I. Seleccione una de las opciones y escriba la letra correspondiente en el paréntesis de la derecha de cada pregunta

1. Es uno de los cuatro tipos de mantenimiento ( )

a) Restauración
b) Preventivo
c) Desastres
d) Integral

2. Es la definición del mantenimiento de mejora ( )

a) Son las modificaciones realizadas sobre las funciones del software.
b) Son las modificaciones realizadas en la apariencia del software.
c) Modificaciones realizadas al software después de su liberación para perfeccionar su desempeño y mantenimiento.
d) Son modificaciones realizadas al software después de su liberación para mantener la utilidad del software ante los cambios del ambiente.

3. ¿Cuál es el estándar que contiene el proceso de mantenimiento de software? ( )

a) IEEE14764
b) IEEE 12207
c) ISO/IEC 9126
d) IEEE 610

4. Es una de las actividades del proceso de implementación ( )

a) Elaborar plan de retiro
b) Capacitación
c) Realizar pruebas a las modificaciones del software
d) Documentar los pasos para la migración

5. Es un elemento de la estrategia de mantenimiento ( )

a) Capacitación
b) Análisis de recursos
c) Compra de equipo
d) Acondicionamiento de área de trabajo

II. Instrucciones. Escribir en la columna de la derecha la letra V (verdadero) o F (falso) según corresponda al enunciado.

No.	Enunciado	V o F
6.	La planificación del proceso de mantenimiento comienza una vez que se ha liberado el software.	
7.	El proceso de mantenimiento de software finaliza con el retiro del software.	
8.	Un error del software detectado en el ambiente de producción es motivo para iniciar las actividades de mantenimiento.	
9.	El mantenimiento de software realiza las actividades necesarias para proporcionar un soporte rentable al software.	
10.	El mantenimiento de software lleva a cabo las actividades de registro de todas las modificaciones que se van realizando a cada una de las liberaciones de software.	

TEMA 9. ADMINISTRACIÓN DE LA CONFIGURACIÓN

Objetivo particular

Al finalizar el tema el alumno podrá identificar el papel que realizar la administración dentro del desarrollo de software. Conocerá las actividades que podrá llevar a cabo con la finalidad de controlar los cambios que se realizan sobre todos los elementos de un proyecto de software.

Temario detallado

9.1. Proceso de administración de la configuración

Introducción

El cambio es algo normal y en el desarrollo de software esto no es una excepción. Pero cada cambio puede conllevar una serie de resultados indeseables, además de genera una confusión dentro de las personas involucradas si no se realiza de manera adecuada, lo que puede derivar en un caos dentro un proyecto.

9.1. Proceso de administración de la configuración

La administración de la configuración es una disciplina que aplica las direcciones técnicas y administrativas y vigilancia para: identificar y documentar las características físicas y funcionales de un elemento de configuración, controlar los cambios de esas características, registrar y procesar el proceso de cambios y el estatus de implementación y verificar el cumplimiento con los requerimientos especificados.[13]

[13] IEEE 610.12

La configuración de un sistema son las características físicas y/o funcionales del software o hardware o su combinación, como un conjunto en documentación técnica y enfocada al producto. Es una colección de versiones específicas de elementos de hardware o software combinado conforma a específicos procedimientos de construcción para propósitos particulares.

La administración de la configuración es la disciplina que identifica la configuración de un sistema en distintos puntos en el tiempo con el propósito de controlar sistemáticamente los cambios de configuración, manteniendo de esta forma la integridad y el rastreo de la configuración durante el ciclo de vida.

Administración de la configuración de software es un proceso de soporte del ciclo de vida que beneficia la administración de proyectos, las actividades de desarrollo y mantenimiento, las actividades de aseguramiento y a los productos finales de los usuarios y clientes.

Los elementos de la administración de la configuración con todos aquellos que se generan durante el proceso de software como: código fuente, ejecutables, documentación y datos.

Un indicador que puede ayudar a reconocer que las cosas van bien es cuando cada elemento es registrado para su seguimiento y control, cada cambio puede ser rastreado y analizado y cuando aquellos que necesitan ser informados de un cambio se les ha informado.

La administración de la configuración se relaciona con las demás áreas desde el momento en que el proceso registra cada artefacto que se produce y se utiliza a través del proceso de software.

El estándar ISO/IEC 12207 indica que el proceso de administración de la configuración de software se encarga de establecer y mantener la integridad de

los elementos de software de un proceso o un proyecto y hacerlos disponibles. Para realizarlo se apoya de las siguientes actividades:

- Implementación del proceso. Se desarrolla el plan que describe las actividades, procedimientos y la programación para realizar las actividades, se establecen los responsables y la relación del proceso con otras organizaciones. El plan se documenta e implementa.
- Identificación de la configuración. Se establecen los elementos que serán controlados. A cada versión y su elemento correspondiente se le coloca un identificador y se establece una línea base.
- Control de la configuración. Se encarga de llevar a cabo el seguimiento al identificar y registrar las solicitudes de cambio, evaluar los cambios, aprobación o desaprobación de los requerimientos, implementaciones, verificaciones y liberaciones. Seguimiento a las modificaciones, razones de modificación y su autorización.
- Estatus de la configuración. Es la administración de reportes de registro y estados que muestran un histórico controlado de los elementos de software.
- Evaluación de la configuración. Se asegura de que exista una relación entre el elemento funcional del software contra los requerimientos y los elementos físicos.
- Administración de liberación y entrega. La liberación y la entrega de los productos de software deben de encontrarse controlados. Mantener las copias maestras de código y documentación. Los elementos críticos son manejados, almacenados, empacados y entregados conforme a las políticas establecidas por la organización.

En SWEBOK se pueden encontrar actividades asociadas a la administración de la configuración. De las cuales se mencionan las siguientes:

- Administración del proceso de administración de la configuración. En ella se ven involucradas la planificación y administración del proceso. Lo que

implica una comprensión del contexto de la organización, la identificación de restricciones y el diseño e implementación del proceso.

- Identificación de la configuración de software. En ella se identifican todos los elementos que deberán de ser controlados para establecer una estructura de identificación por elemento y versión. Se establecerá las técnicas y herramientas que se utilizarán para administrar los elementos.
- Control de la configuración de software. Se establece la administración de los cambios durante el ciclo de vida del software. Se incluye los tipos de cambio, autoridades para realizar los cambios, soporte a la implementación de cambios.
- Registro del estado de la configuración de software. Es el registro y reporte de información necesaria para una efectiva administración de la configuración de software.
- Auditoría de la configuración de software. Es una evaluación independiente de la conformidad de los productos de software y el proceso a regulaciones, estándares, lineamientos, planes y procedimientos. La auditoría determina el grado en el que los elementos satisfacen los requerimientos funcionales y características físicas.
- Administración de liberación y entrega. Liberar se refiere a la distribución de un elemento de configuración de software fuera de las actividades de desarrollo. Esto incluye las distribuciones internas y hacia los clientes. Cuando las diferentes versiones de un mismo elemento se encuentran disponibles para ser entregadas es necesario generar paquetes de esa versión con el material correcto. La librería de software es un elemento importante que acompaña a las tareas de entrega y liberación.

Mientras para el estándar IEEE 1042 se pueden mencionar las siguientes actividades a manera de resumen:

- Identificación de la configuración. Identifica la estructura del producto. Los niveles de identificación es la organización sobre la que se pueden identificar los elementos de la configuración. Posteriormente se

establece el identificador o etiqueta con el que se dará seguimiento a los elementos. Se establece una línea base de que se tomará como referencia a todos los elementos.

- Control de la configuración. Se identifica los procedimientos para los cambios a partir de una línea base. Se generan niveles de autoridad para controlar cambios y se asignan las responsabilidades.
- Registro del estado de la configuración. Identifica la información necesaria, su obtención y reporte.
- Auditorias y revisiones. Involucra los procedimientos empleados para verificar que los productos de software contra la descripción de los elementos de configuración que se encuentra en las especificaciones y documentos. Las auditorias se orientan hacia las funciones de cambio, operación de las librerías y otras actividades relacionadas a la administración de la configuración de software.
- Liberación. Las liberaciones de software deben de estar descritas de manera de quien lo reciba entienda lo que le ha sido entregado. Esta actividad se encarga de verificar que el paquete de liberación se encuentre completo y listo para el usuario.

Bibliografía del tema 9

IEEE. “IEEE Standard Glossary of Software Engineering Terminology”. IEEE Std 610.12-1990, 1990.

ANSI/IEEE. “IEEE Guide to Software Configuration Management”. Std 1042-1987, 1987.

ISO/IEC. “System and Software Engineering – Software Life Cycle Processes”. Std 12207-2008, 2008.

IEEE. “Guide to the Software Engineering Body of Knowledge”. IEEE Computer Society SWEBOK, 2004.

Actividades de aprendizaje

A.9.1. Conseguir el estándar IEEE 829 y basado en él elaborar el formato de plan de administración de la configuración de software.

A.9.2. Elaborar resumen de los tipos de revisiones descritos por el estándar IEEE 1028.

A.9.3. Elaborara catálogo de identificadores para elementos de la configuración de software.

A.9.4. Diagramar una de las actividades de la administración de la configuración utilizando UML.

A.9.5. Investigar una herramienta de software que pueda utilizarse en la administración de la configuración de software.

Cuestionario de autoevaluación

1. ¿Qué es la administración de la configuración?
2. ¿En qué momento se recomienda utilizar la configuración de software?
3. ¿A la administración de la configuración sólo le concierne llevar el código de software?
4. ¿Qué elementos debe de contener una línea base?
5. ¿Cuándo se establece el proceso de configuración de software?
6. ¿Qué tipo de modificaciones que requieren llevar un registro?
7. ¿Cuál es la importancia de llevar un control de configuración durante el proceso de desarrollo de software?
8. ¿De qué manera la configuración de software garantiza la calidad del desarrollo de software?
9. ¿Qué tan riguroso debe de ser el registro de los cambios a la configuración de software?
10. ¿En qué momento finaliza la configuración de software?

Examen de autoevaluación

Instrucciones. Seleccione una de las opciones y escriba la letra correspondiente en el paréntesis de la derecha de cada pregunta

1. ¿Qué es la administración de la configuración? ( )

a) Es la disciplina que identifica la configuración de un sistema en distintos puntos en el tiempo.
b) Es la disciplina que se encarga de aplicar las configuraciones tanto a equipos clientes como a servidores.
c) Se encarga de elaborar manuales y capacitar al personal que se encargará de instalar una nueva aplicación.
d) Se encarga de supervisar que cualquier actualización no modifique las configuraciones del software instalado.

2. Es una de las actividades establecidas por el estándar ISO/IEC 12207 para la administración de la configuración ( )

a) Identificación de la configuración
b) Establecer métricas
c) Generar modelo de negocio
d) Realizar respaldo

3. Es una de las actividades establecidas en SWEBOK para la administración de la configuración ( )

a) Establecer proceso de recuperación
b) Auditoria de la configuración de software
c) Inventario de equipos
d) Actualizaciones de software

4. Es una de las actividades establecidas por el estándar IEEE 1042 para la administración de la configuración ( )

a) Documentación
b) Mantenimiento
c) Capacitación
d) Liberación

5. ¿En qué actividad del proceso de configuración se establece la línea base? ( )

a) Liberación
b) Implantación del proceso
c) Identificación de la configuración
d) Control de la configuración

6. Es la actividad encargada de administrar los reportes de configuración y de llevar un histórico. ( )

a) Auditoria
b) Estatus de la configuración
c) Control de la configuración
d) Implantación del proceso

7. ¿De qué elementos se encarga la administración de la configuración de software? ( )

a) Código fuente
b) Documentación
c) Datos
d) Todas las anteriores

8. ¿En qué actividad del proceso de configuración de software se establecen los roles y actividades? ( )

a) Implementación del proceso
b) Auditoria y revisiones
c) Liberación
d) Control de la configuración

9. ¿En qué actividad se lleva a cabo el seguimiento a las modificaciones en el software? ( )

a) Revisión y auditoria
b) Control de la configuración
c) Implantación del proceso
d) Liberación

10. Es un producto del proceso de configuración de software ( )

a) Librerías
b) Material de capacitación
c) Plan de mantenimiento
d) Plan de pruebas

TEMA 10 ADMINISTRACIÓN DE LA CALIDAD

Objetivo particular

Al concluir este tema el alumno entenderá el papel de la calidad en la ingeniería del software, así como los modelos más importantes que los que se puede apoyarse.

Temario detallado

10.1 ¿Qué es la calidad?

10.2 CMM

10.3 ISO 9000

Introducción

En la actualidad es posible encontrar modelos de calidad generalmente aceptados por la industria del software que pueden ser de utilidad. La calidad puede verse desde diferentes ángulos y cada modelo adoptará alguna de ellas para desarrollarla.

10.1 ¿Qué es la calidad?

Cada persona puede identificar de alguna forma lo que es la calidad. La calidad se convierte en un reto al momento de definirla de manera general debido a que puede ser vista desde muchos puntos de vista. Una vez establecida la perspectiva de calidad se presenta la tarea de concretarla, lo cual tampoco resulta fácil, las limitantes y la resistencia al cambio pueden hacerla difícil de establecerla.

En las siguientes definiciones es posible observar algunos puntos de vista que se tienen de la calidad:

- La calidad es cumplir especificaciones. Cero defectos (P. Crosby).
- La calidad es el cumplimiento de los propósitos. Adecuación para el uso satisfaciendo las necesidades del cliente (J. M. Juran).
- La calidad es un grado predecible de uniformidad y fiabilidad a bajo costo, adecuado a las necesidades del mercado (E. W. Deming).
- La calidad es el costo que un producto impone a la sociedad desde el momento de su concepción (G. Tagushi).

Por otra parte existen otros cinco tipos de definiciones de calidad:

- Definición trascendental. La calidad es absoluta y universalmente reconocible sin que pueda definirse por completo.
- Definición basada en el usuario. Se basa en las necesidades de cada usuario y un producto o servicio es de mayor calidad entre más cumpla con las necesidades. Es un punto de vista subjetivo.
- Definición basada en el producto. Es una variable precisa y medible. La calidad es inherente a las características del objeto, lo que lo hace medible.
- Definición basada en la manufactura. La calidad es la conformidad con las especificaciones. Si el objeto cumple con lo establecido es cuando se cuenta con un nivel aceptable de calidad. La calidad se adquiere gradualmente conforme las características son completadas.
- Definición basada en el valor. La calidad está en relación con los costos. Un objeto de calidad ofrece buena ejecución a un precio aceptable.

Un sistema de aseguramiento de la calidad es una estructura organizacional con responsabilidades, procedimientos, procesos y recursos para implementar la administración de la calidad. Estos sistemas posibilitan a las organizaciones para que sus productos o servicios satisfagan las expectativas de los clientes mediante el logro de sus especificaciones.

10.2 CMM

Modelo de capacidad de madurez CMM (*Capability Maturity Model*). Es un modelo para mejorar el proceso de desarrollo de software.

En 1984, el departamento de defensa de los Estados Unidos de América establece el SEI (*Software Engineering Institute*) en la Universidad de Carnegie Mellon, con la finalidad de encontrar una valoración de sus contratistas. CMM significó para el departamento de defensa una mejora en cuanto a los productos y servicios que empleaba.

Mark Paulk y otros en el SEI crearon el primer modelo de madurez de capacidad, diseñado para organizaciones de desarrollo software. SW CMM había sido el principal producto del SEI, liberado en 1991. Este modelo permitió a las organizaciones a mejorar la eficiencia en el desarrollo de la calidad de los productos de software.

El modelo de madurez de las capacidades es un modelo de referencia de prácticas maduras en una disciplina específica, utilizada para evaluar la capacidad de los grupos para desempeñar esa disciplina.

La capacidad del proceso de software describe el rango de resultados esperados que se obtienen siguiendo un proceso de software.

La madurez del proceso de software es cuando un proceso en específico es definido explícitamente, administrado, medido, controlado y es efectivo.

El objetivo de un proceso de software maduro es producir productos de calidad que cumplan con las especificaciones.

CMM dirige su enfoque a la mejora de procesos en una organización, estudia los procesos de desarrollo y produce una evaluación de la madurez de la organización según una escala de cinco niveles:

1. Inicial. El proceso de software es un proceso improvisado y caótico.
2. Repetible. Se establecen procedimientos de administración del proceso que son básicos para determinar costos, calendarios y funcionalidad.
3. Definido. El proceso de software para las actividades administrativas y técnicas se encuentra documentado, estandarizado e integrado dentro de la organización.
4. Administrado. Se recolectan medidas detalladas del proceso de software y de la calidad del producto. Son cuantitativamente entendidos y controlados.
5. Optimizado. El mejoramiento continuo del proceso es garantizado por la retroalimentación cuantitativa desde el proceso y las pruebas de técnicas y herramientas innovadoras.

Los modelos contienen los elementos esenciales de procesos efectivos para una o más disciplinas y describen el camino para evolucionar y mejorar desde procesos inmaduros a procesos disciplinados, maduros con calidad y eficiencia mejorada y probada.

Debido al éxito de SW CMM y a la demanda de modelos en otras áreas, el SEI desarrolló otros modelos de CMM, conformados de la siguiente manera:

- *Systems engineering* SE – CMM
- *Integrated product development* IPD – CMM
- *Software Acquisition* SA – CMM
- *Human resources* People – CMM
- *Software* SW - CMM

Aunque los modelos habían sido útiles para muchas organizaciones, el uso de múltiples modelos se había vuelto problemático. Muchas organizaciones querían enfocar sus esfuerzos a través de diferentes disciplinas. Pero la diferencia entre cada uno de los modelos, arquitectura, contenido y acercamiento, limitaban la habilidad de las organizaciones para enfocar sus

mejoras de manera exitosa. Aplicar múltiples modelos que no se encontraban integrados se volvía costoso en términos de capacitación, valoración y actividades de mejora.

CMMI fue elaborado para solucionar el problema de múltiples CMMs. El objetivo consistía en combinar tres modelos:

- SW – CMM
- *System engineering capability model* (EIA 731)
- IPD - CMM

Estos tres modelos fueron seleccionados por su amplia aceptación y los diferentes accesos para mejorar procesos en una organización. CMMI fue liberado por el SEI en el 2002.

10.3 ISO 9000

ISO (*Internacional Standard Organization*) en 1987 a través de su TC 176 crea ISO 9000. Describe los elementos de aseguramiento de la calidad de manera genérica.

ISO 9000 es un conjunto de normas utilizadas como marco para diseñar, implantar y certificar sistemas de gestión de calidad. Trata a una organización como a una red de procesos interconectados. Los procesos deberán de encontrarse dirigidas a áreas identificadas él y deberán de estar documentadas y practicadas como se menciona.

Se describen de manera general los elementos que conforman al sistema de aseguramiento de la calidad como: estructura organizacional, procesos, procedimientos y recursos necesarios. Los modelos muestran el “qué” pero no el “cómo” de cada uno de los elementos.

Conjunto de estándares que establecen los requerimientos para la gestión de los sistemas de calidad. Se realizan revisiones periódicas cada cinco años.
La ISO es una organización mundial no gubernamental, compuesta por representantes de los organismos de normalización nacional, con sede en Ginebra. Dicha organización se articula en comités técnicos que se encargan de la elaboración de las normas internacionales, y dichos comités están integrados por miembros de los organismos federados interesados en el objeto de trabajo de la comisión. Una vez elaborado el proyecto de norma, éste es enviado a los organismos miembros para su aprobación, la cual requiere el voto favorable de al menos dos terceras partes de los organismos miembros del comité. Tras su aprobación las normas son difundidas internacionalmente a través de los organismos nacionales federados.

Los estándares se aplican para demostrar que se cuenta con un nivel de experiencia en el diseño y construcción de un producto. Son usadas para regular la calidad interna y asegurar la calidad de los proveedores.

La familia de normas ISO 9000 fue publicado por primera vez en 1987, revisado en 1994 y actualizado nuevamente en el año 2000 (con un compromiso de ser revisado cada 5 años).

Dentro de la familia de estándares del ISO 9000 podemos encontrar:

- ISO 9000, Fundamentos y vocabulario.
- ISO 9001, Requisitos para aseguramiento de la calidad.
- ISO 9004, Directrices para la mejora del rendimiento.
- ISO 9011, Directrices para la auditoría de los sistemas de gestión de la calidad y/o ambiental.

De los estándares del ISO 9000, el ISO 9001 también puede ser aplicado a las actividades relacionadas con el software. La presentación del ISO 9001:2000 es general, pero existe una interpretación se encuentra en el ISO 9000-3, que

es una guía para las organizaciones en la aplicación del ISO 9001:2000 para adquisición, abastecimiento, desarrollo, operación y mantenimiento de software y servicios de soporte relacionados.

La norma ISO 9001:2000 puede utilizarse para cualquier tipo de industria por su carácter “genérico”. Es un enfoque de gestión de calidad basada en procesos. Este enfoque facilita la ejecución y gestión de actividades, así como ciclos de mejora continua siguiendo el principio planificar-ejecutar-verificar-actuar (PDCA por sus siglas en inglés). El flujo de la información generada por las mediciones son proporcionadas a los responsables de tomar decisiones y corregir planes. ISO 9001 se centra en la eficacia del sistema de gestión de la calidad para dar cumplimiento a los requisitos del cliente.

ISO 9001: 2000 “Sistema de gestión de calidad – requisitos” está estructurado en ocho secciones:

1. Objetivo y campo de aplicación
    1.1 Generalidades
    1.2 Aplicación
2. Referencias normativas
3. Términos y definiciones
4. Sistema de gestión de calidad
    4.1 Requisitos generales
    4.2 Requisitos de documentación
        4.2.1 Generalidades
        4.2.2 Manual de calidad
        4.2.3 Control de registros de la calidad
5. Responsabilidad de la dirección
    5.1 Compromiso de la dirección
    5.2 Enfoque al cliente
    5.3 Política de calidad
    5.4 Planificación

7.4.1 Proceso de compras

7.4.2 Información de las compras

7.4.3 Verificación de los productos comprados

7.5 Producción y presentación del servicio

7.5.1 Control de la producción y presentación del servicio

7.5.2 Validación de los procesos

7.5.3 Identificación y trazabilidad

7.5.4 Propiedad del cliente

7.5.5 Preservación del producto

7.6 Control de los dispositivos de seguimiento y medición

8. Medición, análisis y mejora

8.1 Generalidades

8.2 Seguimiento y medición

8.2.1 Satisfacción del cliente

8.2.2 Auditoría interna

8.2.3 Seguimiento y medición de los procesos

8.2.4 Seguimiento y medición del producto

8.3 Control del producto no conforme

8.4 Análisis de datos

8.5 Mejora

8.5.1 Mejora continua

8.5.2 Acción correctiva

8.5.3 Acción preventiva

Bibliografía del tema 10

Pressman, Roger. *Software Engineering: A practitioner's approach*. New York, McGraw-Hill, 2001.

Chrissis, Mary Beth. *CMM Guidelines for process integration and product improvement*. México, Addison-Wesley, 2003.

Actividades de aprendizaje

A.10.1. Generar un listado con diez problemas relacionados a la calidad.

A.10.2. Elaborar un resumen de las características de calidad presentadas en el estándar ISO/IEC 9126.

A.10.3. Investigar las diferencias entre el modelo CMM y CMMi.

A.10.4. Investigar y hacer un resumen de un modelo de calidad distinto a los mencionados por el tema.

A.10.5. Investigar una herramienta de software que se pueda utilizar en la administración de la calidad.

Cuestionario de autoevaluación

1. ¿Qué es la calidad?
2. ¿Cuáles son las dificultades para definir a la calidad?
3. ¿Qué es el software de calidad?
4. ¿Cuáles son los principales obstáculos para construir software de calidad?
5. ¿Quién establece los criterios de calidad?
6. ¿Es posible abarcar todos los criterios de calidad?
7. ¿Cuáles son los medios para asegurar la calidad en el software?
8. ¿Por qué es importante construir software de calidad?
9. ¿Cuál es el primer paso para adoptar una perspectiva de calidad de software?
10. ¿Cuál es la relación de la calidad y la ingeniería del software?

Examen de autoevaluación

Instrucciones. Seleccione una de las opciones y escriba la letra correspondiente en el paréntesis de la derecha de cada pregunta

1. ¿Quién menciona que son cinco los tipos de definición que existen para la calidad? ( )

a) J. M. Juran
b) Van Genuchten
c) G. Tagushi
d) Roger Pressman

2. ¿De quién es la definición “La calidad es cumplir especificaciones”? ( )

a) P. Crosby
b) E. W. Deming
c) Van Genuchten
d) J. M. Juran

3. Es la definición de calidad basada en el valor ( )

a) Se basa en las necesidades de cada usuario y un producto o servicio es de mayor calidad entre más cumpla con las necesidades
b) La calidad es absoluta y universalmente reconocible sin que pueda definirse por completo
c) La calidad está en relación con los costos.
d) La calidad es inherente a las características del objeto, lo que lo hace medible

4. ¿Qué es la capacidad del proceso de software? ( )

a) Un proceso en específico es definido explícitamente, administrado, medido, controlado y es efectivo.
b) Que un proceso se encuentre automatizado
c) Describe el rango de resultados esperados que se obtienen siguiendo un proceso de software
d) Que un proceso se encuentre alineado a los objetivos de una organización

5. ¿Qué es madurez del proceso de software? ( )

a) Un proceso en específico es definido explícitamente, administrado, medido, controlado y es efectivo.
b) Que un proceso se encuentre automatizado
c) Describe el rango de resultados esperados que se obtienen siguiendo un proceso de software
d) Que un proceso se encuentre alineado a los objetivos de una organización

6. ¿De los modelos de CMM cuál es el que aplica para el desarrollo de software? ( )

a) SW - CMM
b) SE - CMM
c) SA – CMM
d) IPD – CMM

7. CMM es un modelo para… ( )

a) Mejorar el desempeño del software
b) Mejorar el proceso de desarrollo de software
c) Dimensionar el tamaño de proyectos de software
d) Desarrollar software

8. ¿Es una guía de interpretación del ISO 9001 para organizaciones con actividades relacionadas al software? ( )

a) ISO 9004
b) ISO 9000-3
c) ISO 9011
d) ISO 9000

9. ¿Cuántos son los niveles de madurez según CMM? ( )

a) 5
b) 10
c) 8
d) 4

10. ¿Cuántas secciones tiene el estándar ISO 9001:2000? ( )

a) 5
b) 10
c) 8
d) 4

Respuestas de los exámenes de autoevaluación

INGENIERÍA DEL SOFTWARE

Tema 1	Tema 2	Tema 3	Tema 4	Tema 5	Tema 6	Tema 7	Tema 8	Tema 9	Tema 10
I. 1. a	1. d	1. v	1. d	1. e	1. b	I. 1. a	I. 1. b	1. a	1. b
2. a	2. e	2. f	2. b	2. g	2. f	2. a	2. c	2. a	2. a
3. c	3. a	3. f	3. b	3. j	3. i	3. b	3. a	3. b	3. c
4. c	4. f	4. v	4. c	4. h	4. g	4. a	4. c	4. d	4. c
5. a	5. g	5. f	5. d	5. a	5. j	5. b	5. b	5. c	5. a
II. 6. c	6. b	6. v	6. d	6. f	6. h	II. 6. c	II. 6. f	6. b	6. a
7. d	7. c	7. v	7. a	7. i	7. d	7. a	7. v	7. d	7. b
8. b	8. h	8. f	8. b	8. b	8. e	8. c	8. v	8. a	8. b
9. e	9. j	9. f	9. a	9. d	9. c	9. d	9. v	9. b	9. a
10. a	10. i	10. f	10. d	10. c	10. a	10. d	10. f	10. a	10. c

.