Técnico en
DESARROLLO DE SOFTWARE

Fundamentos de Construcción de Software

Fundamentos de Construcción de Software

Unidad II

Etapas de la construcción del software

1. Análisis y recolección de requerimientos

Para poder iniciar un proyecto de desarrollo de software, debemos recabar la información necesaria comunicándonos con el cliente. Esta es una tarea clave, donde el cliente nos explica cuál es el problema que quiere resolver utilizando una solución basada en computación. Este paso es muy importante porque debemos recolectar la mayor cantidad de información para poder construir la solución adecuada. Para que la comunicación sea efectiva debemos aplicar algunos principios, estos principios deben utilizarse no solo con el cliente sino también con el equipo de trabajo.

- **Escuchar:** Escuche atentamente lo que indica el cliente, si algo no está claro haga las preguntas necesarias pero evite las interrupciones constantes. Si algo no le parece, evite las palabras bruscas o los gestos groseros, indique amablemente alguna alternativa o el motivo por el cual no se puede realizar la solicitud.
- **Prepararse:** Dedique algún tiempo a entender el problema antes de reunirse con otras personas. Si es necesario, haga algunas investigaciones para entender el

vocabulario propio del negocio. Si tiene la responsabilidad de conducir la reunión, prepare una agenda antes de que ésta tenga lugar.

- **Tomar notas y documentar las decisiones:** Es importante escribir de forma resumida todos los temas tratados y las decisiones tomadas en la reunión. Esto sirve como recordatorio de los responsables de las tareas.
- **Perseguir la colaboración:** La colaboración y el consenso ocurren cuando el conocimiento colectivo de los miembros del equipo se utiliza para describir funciones o características del producto o sistema. Cada pequeña colaboración sirve para generar confianza entre los miembros del equipo y crea un objetivo común para el grupo.
- **Permanecer centrado en el tema:** Entre más personas participen en cualquier comunicación, más probable es que la conversación salte de un tema a otro. Cuando el cliente cambie de tema se debe evitar perder tiempo en ello y regresar a los temas del proyecto.
- **Si algo no está claro, hacer un dibujo:** La comunicación verbal tiene sus límites. Con frecuencia, un esquema o dibujo arroja claridad cuando las palabras no bastan para hacer el trabajo.
- **Avanzar:** La comunicación requiere tiempo. En vez de hacer iteraciones sin fin, las personas que participan deben reconocer que hay muchos temas que requieren análisis y que “avanzar” es a veces la mejor forma de tener agilidad en la comunicación.
- **La negociación no es un concurso:** Hay muchas circunstancias en las que usted y otros participantes deben negociar funciones y características, prioridades

y fechas de entrega. Si el equipo ha colaborado bien, todas las partes tendrán un objetivo común. Aun así, la negociación demandará el compromiso de todas las partes.

2. Recolección de requerimientos

Tanto el desarrollador como el cliente participan activamente en la recolección de los requisitos. El cliente intenta replantear un sistema confuso, a nivel de descripción de datos, funciones y comportamiento, en detalles concretos. El desarrollador actúa como interrogador, como consultor, como persona que resuelve problemas y como negociador.

La recolección y análisis de los requisitos puede parecer un proceso sencillo pero no lo es. En la comunicación entre el desarrollador y el cliente pueden darse malas interpretaciones, falta de información o ambigüedad. A continuación se listan algunas herramientas para la recopilación de requerimientos:

Entrevistas / reuniones uno-a-uno

La técnica más común usada para iniciar el proceso de comunicación es una entrevista. Las reuniones se utilizan ampliamente para reunir los requisitos. Las entrevistas uno a uno requieren un poco de planificación y preparación antes de la entrevista y la documentación de los hallazgos posteriores. Hacer diferentes tipos de preguntas, por ejemplo, de seguimiento o sondeo, puede ayudar a obtener los requisitos del sistema para cada uno de los usuarios o las partes interesadas. De manera general el desarrollador debe encontrar las respuestas a las siguientes interrogantes:

- ¿Quién utilizará el sistema?
- ¿Cuáles son los objetivos del sistema?
- ¿Cuáles son los beneficios de esta solución?

El desarrollador debe centrarse en entender el problema:

¿Cuál es el entorno donde se va utilizar el software?

¿Cuáles son las restricciones o mejoras sobre la situación actual?

¿Hay más personas que darán información?

¿Existen dudas por parte del cliente?

¿Se debe preguntar más?

Entrevistas / reuniones grupales

Son entrevistas donde estén presentes más usuarios o representantes de las partes interesadas, pero por lo demás son similares a las anteriores. Las entrevistas grupales pueden ser más productivas si los entrevistados se encuentran en un nivel similar o en la misma sección dentro de un departamento.

Talleres

Talleres pueden comprender 6-10 o más usuarios / partes interesadas, trabajando juntos para identificar los requisitos. Los talleres tienden a tener una duración definida, en lugar de un resultado, y es posible que deban repetirse brevemente para aclarar u obtener más detalles.

Lluvia de ideas

La lluvia de ideas se puede hacer individualmente o en grupos. Las ideas recogidas pueden luego ser revisadas / analizadas y, cuando corresponda, incluirlas dentro de los requisitos del sistema. Las ideas pueden provenir de lo que los usuarios o partes interesadas, ideas que han visto en otros sistemas (por ejemplo, en exhibiciones de software) o lo que han experimentado en otro lugar (por ejemplo, antes de unirse a la organización actual).

Cuestionarios

Los cuestionarios pueden ser útiles para obtener detalles de requisitos de sistema que estén limitados de usuarios o partes interesadas, que tienen un aporte menor o son geográficamente remotos. El diseño del cuestionario (ya sea fuera de línea o basado en la web) y los tipos de preguntas son importantes y pueden influir en las respuestas, por lo que se necesita cuidado al plantearlo.

Observación / seguimiento

Observar a las personas que utilizaran el software, imitar a los usuarios o incluso hacer parte de su trabajo, pueden proporcionar información sobre procesos, entradas y productos existentes.

Prototipos

El prototipado consiste en reunir requisitos y usarlos para construir un prototipo. Permite a los usuarios ver una posible solución y tener una mejor idea de lo que necesitan. Luego,

los usuarios agregan o modifican sus requisitos, y el prototipo se vuelve a trabajar de forma iterativa, hasta que se cumplan los requisitos.

Historias de usuario

Las historias de usuario son uno de los principales artefactos de desarrollo para los equipos de proyecto de Scrum y Extreme Programming (XP). Una historia de usuario es una definición de alto nivel de un requerimiento, que contiene la información suficiente para que los desarrolladores puedan producir una estimación razonable del esfuerzo para implementarla.

Un concepto importante es que las partes interesadas del proyecto escriben las historias de los usuarios, no los desarrolladores del proyecto. Las historias de los usuarios son lo suficientemente simples como para que las personas puedan aprender a escribirlas en unos minutos, por lo que tiene sentido que los expertos en el dominio (las partes interesadas) las escriban.

Las historias deben ser breves, deben poder escribirse en una ficha de papel. Toda historia se debe conversar con las partes interesadas para que el equipo tenga suficiente información, se hagan ajustes y se pueda confirmar la realización de la historia.

Mike Cohn sugiere un enfoque más formal para escribir las historias de usuario, el formato es el siguiente:

Como <tipo de usuario>, puedo o necesito <algún objetivo> para que <algún motivo>.

Por ejemplo:

Como administrador, puedo ver el listado de productos con baja existencia para hacer más pedidos.

Como usuario, necesito validar el ticket para poder salir del parqueo.

3. Análisis de requisitos

Los diferentes métodos de análisis de los requisitos que se han establecidos se basan en una serie de principios fundamentales que permiten dar un enfoque sistémico al problema a resolver. Roger S. Pressman identifica los siguientes principios:

- Se debe comprender el ámbito de información del problema.
- Se deben desarrollar los modelos que representan la información, función y el comportamiento del sistema.
- Se deben subdividir los modelos (y el problema) de forma que se descubran los detalles de una manera progresiva (o jerárquica).
- El proceso de análisis debe ir de la información esencial hacia el detalle de la implementación.

El ámbito de la información determina el flujo el contenido y la estructura de la información. Los modelos se construyen para darle mejor entendimiento al sistema que se va a desarrollar. El modelo facilita el entendimiento de la información, la función y el comportamiento del sistema y sirve como base del diseño. La subdivisión de los problemas grandes en los problemas más pequeños facilita su comprensión como un todo. El planteamiento esencial de los requisitos del software presenta las funciones que deben realizarse independientemente de los detalles de implementación. El

planteamiento de implementación de los requisitos presenta las funciones de procesamiento y de las estructuras del mundo real.

Para poder analizar de mejor forma los requisitos se pueden utilizar varias técnicas, entre ellas están los casos de uso y dividir historias de usuario.

Clasificación de requisitos

Los requisitos se pueden clasificar para poder organizarlos de mejor forma, de la siguiente manera:

- **Requisitos funcionales**: Son las funcionalidades que el software debe proporcionar, cómo debe reaccionar el software ante entradas particulares y cómo debe comportarse el software en situaciones particulares (y a veces también lo que no debería hacer el software).
- **Requisitos no funcionales**: Restricciones en los servicios o funciones que ofrece el software, tales como restricciones de tiempo, restricciones en el proceso de desarrollo, estándares, seguridad, etc. Se aplican al sistema como un todo.
- **Requisitos de dominio**: Requisitos que provienen del dominio de la aplicación del sistema y que reflejan las características de ese ámbito, como las reglas del negocio.

Casos de Uso

Un caso de uso caracteriza una forma de usar un sistema, representa un diálogo entre un usuario y el sistema desde el punto de vista del usuario. Se utiliza para capturar requisitos funcionales. Los beneficios de hacer los casos de uso son establecer un entendimiento entre los requisitos del cliente y los desarrolladores del sistema,

comprender las posibles situaciones problemáticas y capturar un nivel de funcionalidad para poder planificar el proyecto.

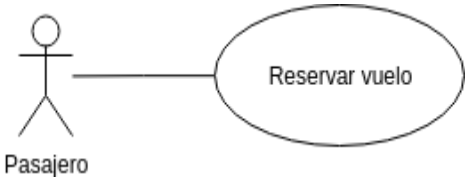
En los casos de uso las personas que interactúan con el sistema se denominan actores y a la persona que inicia la acción se le denomina el actor principal. La meta es el resultado deseado al ejecutar la acción por el actor principal.

Estilos de casos de uso

Diagramas de casos de uso (UML)

La lista de los casos de uso de un sistema se pueden dibujar como diagramas de alto nivel con:

- Actores como iconos, con sus nombres (sustantivos).
- Casos como elipses con sus nombres (verbos).
- Asociaciones de línea, conectando a un actor con un caso de uso en el que ese actor participa.

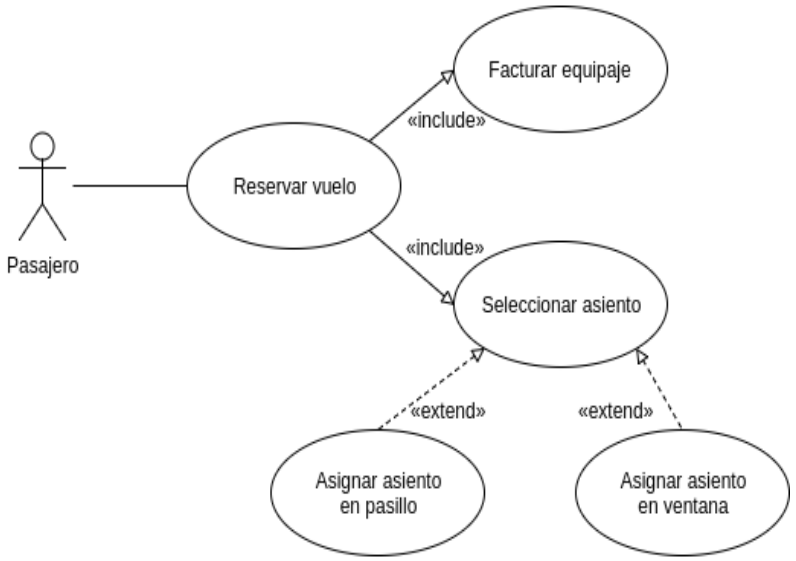


Los casos de uso se pueden conectar a otros casos, por ejemplo cuando se desea incluir un caso existente se utiliza la palabra «include». Se utiliza para simplificar los diagramas de casos de uso.

Los casos de uso también se pueden extender con casos de uso opcionales bajo ciertas condiciones, para ello se utiliza la etiqueta «extend». Esto es útil para manejar casos

especiales, por ejemplo en un caso de uso de pagos el cliente puede hacer uso de la extensión de pago en efectivo o hacer uso de la extensión de pago con tarjeta de crédito.

Include y extend se utiliza generalmente para detallar un caso de uso, por ejemplo:

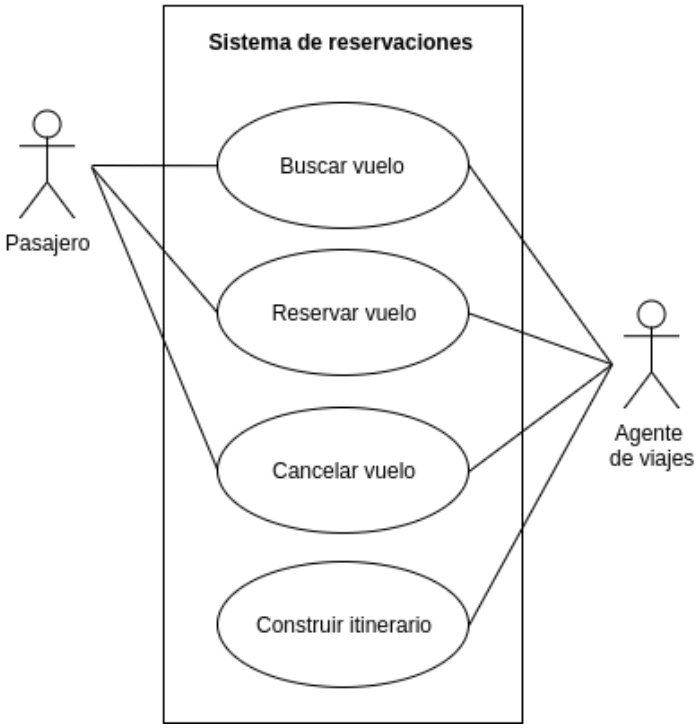


Antes de crear los diagramas es útil crear una lista o tabla de los posibles actores de la aplicación solicitada y las posibles acciones. Por ejemplo:

Actor	Acción
Pasajero	Buscar vuelo
	Reservar vuelo
	Cancelar vuelo
Agente de viajes	Buscar vuelo
	Reservar vuelo
	Cancelar vuelo

	Construir itinerario

Luego los datos de la tabla se deben trasladar al diagrama. En un diagrama de casos de uso pueden aparecer varios actores, por ejemplo:



Casos de uso informales

El caso de uso informal se escribe como un párrafo que describe el escenario/interacción.

Por ejemplo:

Pasajero pierde boleto de vuelo

El pasajero reporta en el mostrador de la aerolínea que perdió su boleto. El empleado de la aerolínea le pregunta al pasajero los datos de su vuelo y le solicita su pasaporte. El

empleado ingresa al sistema y verifica que los datos concuerden. En el sistema el empleado debe marcar que el boleto actual fue extraviado, para que nadie más lo pueda utilizar y procede a generar uno nuevo. Luego imprime el boleto y se lo entrega al pasajero.

Casos de uso formales

Para los casos de uso formales se debe detallar a profundidad cada caso, por ejemplo:

Objetivo	Un pasajero podrá reservar boletos aéreos por internet
Actor	Pasajero
Ámbito	Sistema de reservaciones en línea
Nivel	Usuario
Pre-condición	Pasajero debe iniciar sesión en el sistema
Condición de éxito	Pasajero reserva el vuelo
Condición de fallo	Pasajero no puede reservar vuelo
Desencadenante	Pasajero da click en botón reservar vuelo
Escenario exitoso	Pasajero ingresa usuario y password El sistema verifica los datos e inicia sesión El sistema le presenta la página principal El pasajero da click en reservar vuelo

	El pasajero ingresa los datos de origen, destino y fechas de salida y regreso El sistema le muestra los vuelos disponibles El pasajero selecciona el vuelo más adecuado y hace la reservación El sistema le solicita el pago El pasajero paga el boleto El sistema le proporciona al pasajero el archivo con el boleto
Posibles fallas	2.a. Password es incorrecto 2.a.1 El sistema le muestra que el password es incorrecto 2.a.2 El pasajero ingresa de nuevo el password ...
Variaciones (Escenarios alternos)	5. El pasajero reserva solo boleto de ida ...

Pasos para crear un caso de uso completo

- *Identificar actores y objetivos*: Los actores son los que van a utilizar el software y los objetivos son las tareas que se esperan realizar con el software.
- *Escribir el escenario exitoso:* Escribir la secuencia de pasos que debe seguir el actor para alcanzar el objetivo deseado. Debe ser fácil de leer y comprender. El escenario principal es la ruta preferida por el usuario. Captura el intento y la responsabilidad de cada actor, desde el desencadenante hasta la entrega del objetivo.
- *Enumerar las posibles fallas:* Usualmente cada paso puede fallar, se deben considerar las posibles condiciones de falla de cada paso del escenario exitoso.

- *Listar las variaciones*: Muchos pasos pueden tener comportamientos o escenarios alternativos, también se deben listar para tomarlos en cuenta.

Después de crear los casos de usuario, revisarlos para:

- Eliminar todos los requisitos que no son absolutamente necesarios
- Simplificar aquellos que son más complicados que necesarios
- Sustituir opciones más baratas cuando se pueda
- Mover elementos no esenciales a futuras versiones

Dividir historias de usuario

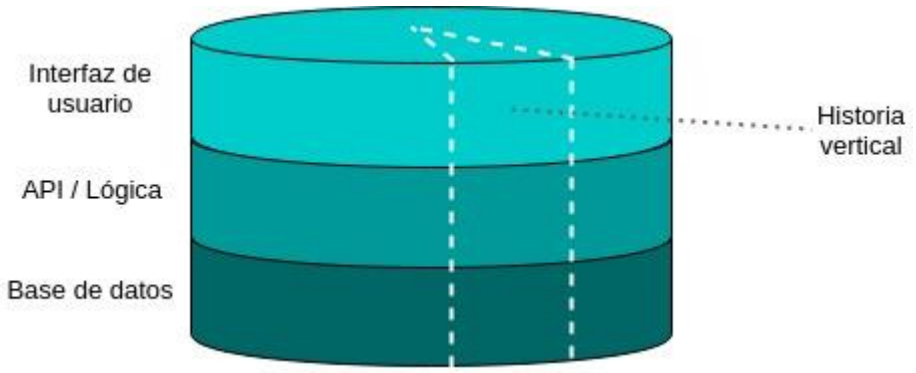
Aprender a dividir historias de usuario largas en historias más pequeñas es una de las mejores cosas que un equipo puede realizar para mejorar su flujo de trabajo de Scrum.

Es posible dividir las historias de usuario de manera horizontal y vertical.

Las historias horizontales son aquellas que pertenecen a una capa estructural del sistema, como la base de datos, la interfaz de usuario, etc. Son las historias que son fáciles de entender solo por los especialistas de esa capa en particular. Inclusive si reciben el nombre de historias, y sean escritas como historias, en realidad son un requisito, ya que rara vez las historias horizontales llegarán al consumidor sin interacción con otras capas del sistema.

Las historias verticales son todas aquellas historias que cruzan todos los límites de la arquitectura del sistema, pero solo implementan una funcionalidad a la vez. La ventaja de dividir una historia de manera vertical, es que cada historia resultante dará un valor significativo para el usuario. Esto la hará más completa y más sencilla de probar, lo cual

facilitará al consumidor poder dar su opinión en base a la experiencia que haya tenido con ella.



Enfoques para dividir historias de usuario

Hay una serie de aspectos a partir de los cuales puede surgir la complejidad de la historia de un usuario, cada uno de los cuales podría prestarse para tomar una porción vertical y dividir la historia en historias más pequeñas y manejables:

Usuarios	¿La característica está destinada a varios tipos de usuarios? ¿Puede cada tipo de usuario tener su propia historia, comenzando por la más valiosa?
Datos	¿La característica maneja varios tipos de datos? ¿Pueden restringirse los datos para generar una historia inicial menos compleja con historias adicionales para incorporar los tipos de datos adicionales?
Procesos	¿La historia describe un proceso o flujo de trabajo? ¿Se pueden dejar los pasos iniciales y finales como una historia y agregar los pasos intermedios como historias adicionales? ¿Se puede dejar la ruta simple y agregar las excepciones y casos extremos más adelante?

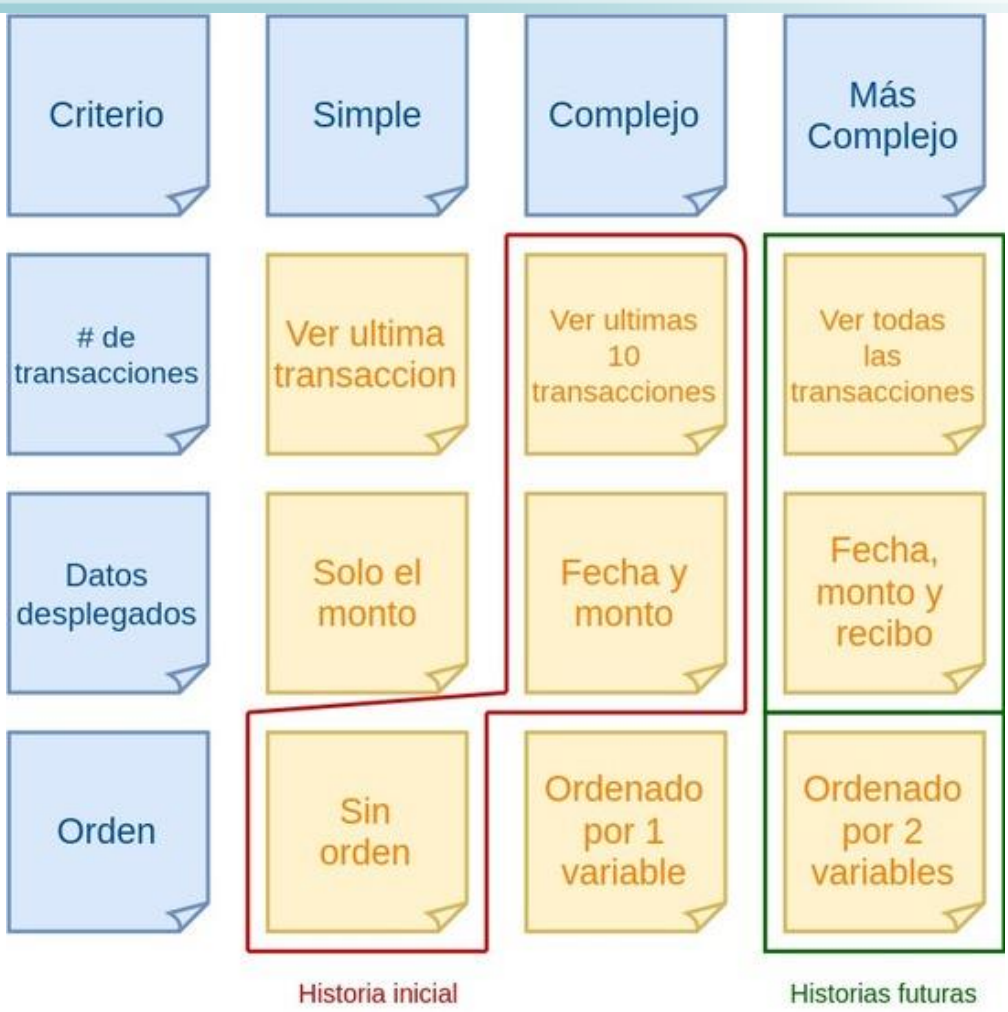
Criterios de aceptación	¿Se pueden quitar algunos de los criterios de aceptación y ponerlos en una historia separada?
Desempeño	¿La historia gana complejidad a partir de requisitos no funcionales? ¿Pueden los requisitos no funcionales ser retirados y el rendimiento diferido para historias posteriores?
Interfaz	¿La historia involucra una interfaz compleja? ¿Se puede crear uno más simple primero? O bien, si la historia maneja datos de varias interfaces, ¿Pueden abordarse primero las más valiosas y las otras agregarse después?

Ejemplo práctico

El cliente nos ha dado la siguiente historia: "Como usuario autenticado, quiero ver una lista de transacciones" y los siguientes criterios de aceptación:

- Se deben poder ver todas las transacciones
- Se debe poder ver la fecha, cantidad, destinatario de cada transacción
- Se debe poder ordenar transacciones por fecha y/o cantidad

Para poder dividir la historia, una opción es tomar cada criterio de aceptación y proporcionar una cantidad de opciones alternativas desde la más simple a la más compleja. Para crear las sub-historias se puede tomar una opción de cada criterio y agruparlas.



4. Diseño de software

El diseño del software es muy importante en el desarrollo ya que facilita la entrega de software de alta calidad. La meta del diseño de software es crear un modelo que pueda ser utilizado como guía para implementar la funcionalidad y al mismo tiempo buscando incluir una estética agradable al producto final. El diseño de software es la etapa en la cual se une los requerimientos, necesidades e ingenio del desarrollador para crear un producto funcional.

El diseño de software utiliza modelos para de esta manera representar la arquitectura del sistema y las interfaces que utilizarán los usuarios finales. Es muy útil que los modelos se aproximen al resultado final lo más posible ya que de esta manera pueda ser evaluado por el equipo de desarrollo. Esta revisión permite evaluar si existen errores, funcionalidades faltantes, realizar optimizaciones, etc. Una estrategia que facilita el diseño del software es empezar con un análisis de la información que el sistema gestionará.

El proceso de diseño de software también debe incluir la definición de los estándares de calidad que el software debe cumplir para poder ser lanzado. Deben definirse pruebas y definir las entradas y salidas esperadas del programa. Esto sirve como guía para los ingenieros que general el código y las personas que lo prueban.

El proceso de abstracción es muy útil para el diseño del software. Al realizar la abstracción de los distintos componentes del sistema proveen soluciones modulares en el software. Como se expuso con anterioridad, existen requerimientos funcionales y no funcionales. Estos requerimientos moldean la arquitectura del software en propiedades estructurales y extrafuncionales. En las propiedades estructurales encontramos los componentes del sistema como módulos, objetos, filtros, etc. y la lógica que las agrupa e interconecta. En las propiedades extrafuncionales se definen los requerimientos específicos como índices de rendimiento, capacidades, confiabilidad, seguridad, etc.

4.1. Diseño modular

Un diseño de software modular permite la división de problemas grandes en varios problemas pequeños por resolver. El software es dividido en varios componentes con

distintos nombres, cada uno conocido como un módulo y pueden trabajarse de manera independiente. A pesar que es ideal dividir el software en múltiples módulos independientes, es necesario evaluar el alcance del proyecto y el esfuerzo que tomaría separar los componentes ya que si el costo es muy alto, entonces es mejor utilizar menos cantidad de módulos. El costo involucrado es el costo de integración de los componentes. En conclusión, debe evitarse hacer pocos módulos y también debe evitarse hacer muchos módulos.

Cada módulo debe de poder funcionar independientemente a otros módulos. A este concepto se le conoce como independencia funcional. Por ejemplo, una calculadora puede separarse en diferentes módulos que conforman las distintas operaciones definidas. Cada operación espera un conjunto de entradas y devuelve una salida, pero lo importante es que cada operación puede operar independientemente de otra. Es decir, la suma, es funcionalmente independiente a la resta y a la multiplicación, etc. Esta capacidad de funcionar independientemente permite la reutilización de los módulos en diferentes proyectos y esto ayuda a que los desarrolladores no pierdan tiempo reinventando la rueda. A la vez, también facilitan el mantenimiento y aumento de funcionalidad de los mismos sin afectar al resto del programa.

Un aspecto que los métodos de desarrollo ágiles contemplan es el rediseño. El rediseño es pensado con el fin de simplificar el código, sin cambiar su función o comportamiento. Este proceso busca eliminar redundancias, corregir estructuras de datos mal construidas o inapropiadas. El resultado es un software más fácil de integrar, probar y mantener.

4.2. Arquitectura de software

Además de tener el diseño del software como tal dividido en módulos, es necesario realizar el diseño de la arquitectura de cada uno de esos componentes. A partir de los requerimientos de información, se crean las estructuras de datos y la del programa. Este trabajo puede ser realizado por los desarrolladores de software aunque en sistemas grandes, esta tarea se delega a los diseñadores de bases de datos o data warehouse. El rol del arquitecto de software es muy importante ya que define los planos que los desarrolladores seguirán. Debido a que una buena arquitectura minimizará los errores que se puedan encontrar en un sistema, es necesario realizar revisiones periódicas en cada etapa para que estos estén claros, sean correctos, estén completos y alineados con los requerimientos.

La arquitectura de software cuenta con cinco componentes principales que son:

Estructura funcional: Representan instancias de una función o proceso. Se definen las propiedades de los componentes y la organización de las interfaces.

Estructura de implementación: Son los paquetes, clases, objetos, procedimientos, funciones, etc. que se usan para ejecutar diversas acciones con distintos grados de abstracción.

Estructura de concurrencia: Son tareas que se pueden realizar de manera paralela. Usualmente incluyen algún proceso de sincronización. En esto se definen las prioridades y tiempo de ejecución para prevenir deadlocks y minimizar tiempos de espera.

Estructura física: Son los componentes de hardware físico que necesita el software para poder llevar a cabo las tareas. Por ejemplo, un lector de código de barras en un punto de venta.

Estructura de desarrollo: Define los componentes, productos de trabajo y fuentes de información que se requieren y utilizan durante el proceso de ingeniería de software. Cada estructura muestra un punto de vista distinto de la arquitectura del software y muestra información útil para el software a medida que éste es modelado y construido.

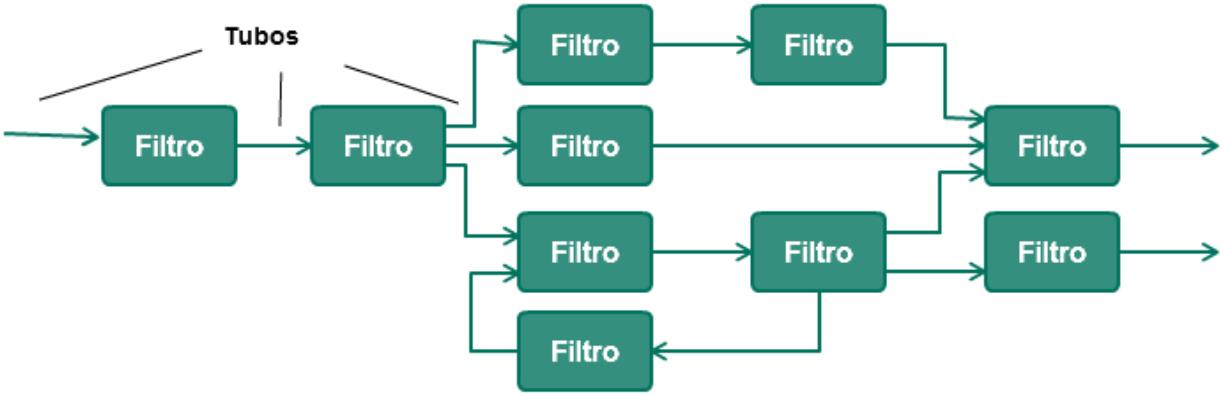
Las principales arquitecturas son las siguientes:

Arquitecturas centradas en datos

Esta arquitectura tiene la característica que en el centro de la misma tiene una estructura de almacenamiento de datos o una base de datos. Estos datos son solicitados frecuentemente y la información cambia con distintas acciones, como actualización, agregar información, eliminar o modificar datos.

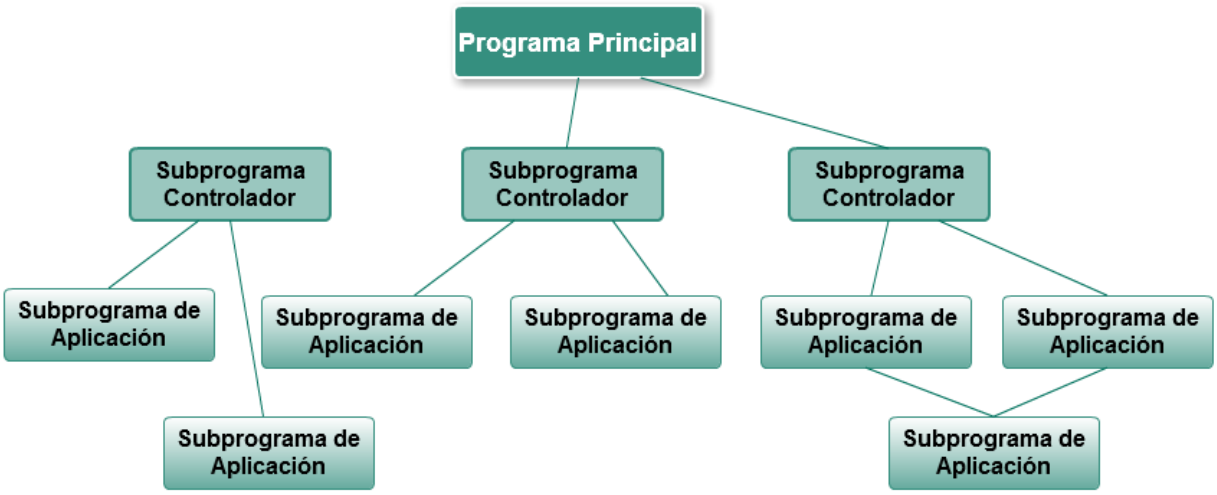
Arquitectura de flujo de datos

En esta arquitectura, se sigue un flujo definido en el que se ingresa información y ésta se transforma por medio de varios componentes que manipulan los datos hasta obtener el resultado final. Se puede hacer una analogía con un sistema de tuberías.



Arquitectura de llamar y regresar

Este estilo de arquitectura se comporta de manera jerárquica, es decir que un programa principal requiere e invoca a varios componentes y cada componente invoca a otros subcomponentes. Otra variante de esta arquitectura son las de llamada de procedimiento remoto. En esta variante, se ejecutan funciones por medio de integraciones en computadoras que están interconectadas a través de una red.

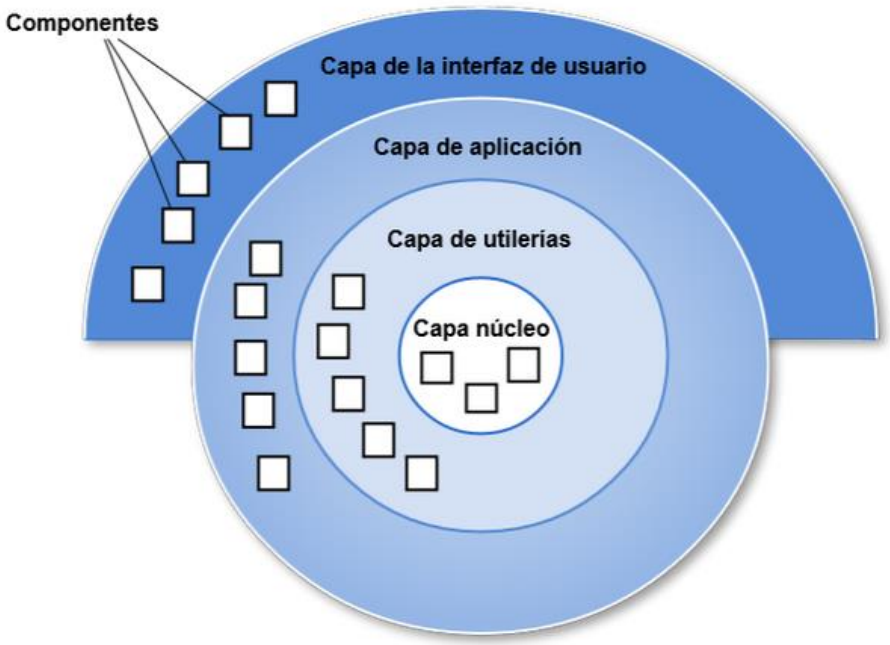


Arquitectura orientada a objetos

Los componentes del sistema se modelan en clases en la cual se definen atributos y métodos los cuales modifican la información. Estos componentes incluyen los datos y las operaciones disponibles para manipularlos. Es necesario coordinar y comunicar los componentes para transmitir datos.

Arquitectura en capas

Se definen capas con diferentes niveles de abstracción, desde la presentación de interfaces de usuario hasta operaciones básicas del computador.



4.3. El proceso de diseño

Durante el proceso de diseño del software se traducen los requisitos en una representación del software. El diseño se realiza en dos pasos:

- El diseño preliminar
- El diseño detallado.

El diseño preliminar realiza la transformación de los requisitos de datos y de arquitectura. En el diseño detallado se realiza el refinamiento de la estructura arquitectónica, se obtiene una estructura de datos detallada y la estructura algorítmica del software. Además del diseño de datos, del diseño arquitectónico y del diseño procedimental se realiza el diseño de la interfaz que describe cómo se comunica el software consigo mismo, con los sistemas que operan con él y con los operadores que lo emplean.

Para obtener un diseño de calidad se necesita tomar en consideración algunos criterios. Según el Roger S. Pressman [1], los criterios de calidad del diseño son los siguientes:

- Presentar relaciones jerárquicas de los componentes necesarios por la aplicación, por ejemplo la relación tipo cliente-servidor entre componentes.
- Ser modular, es decir, estar dividido en forma lógica en componentes que realicen funciones y subfunciones específicas.
- Contener abstracciones de datos y procedimentales
- Producir componentes que exhiban características funcionales independientes.
- Conducir a interfaces que reduzcan la complejidad de las conexiones entre los componentes y el entorno exterior.
- Obtenerse usando un método que pudiera repetirse según la información obtenida durante el análisis de requisitos del software. [1]

5. Codificación de software

La construcción del software se refiere a la creación detallada del software por medio de la codificación, verificación, pruebas y depuración. Los fundamentos de construcción del software comprenden los siguientes aspectos:

Minimizar la complejidad

La necesidad de reducir la complejidad de software se aplica a todo el aspecto de la construcción del software y tiene mucha importancia en el proceso de verificación y prueba. En el proceso de la construcción del software la reducción de la complejidad alcanza a través del uso de código simple y legible y no tanto inteligible. Se logra minimizar la complejidad por medio del uso de los estándares y de una serie de técnicas específicas.

Anticiparse a los cambios

La mayoría de los sistemas de software cambian a través de tiempo y es importante anticiparse a estos cambios.

Construir para verificar

El software debe construirse de tal manera que los desarrolladores deben ser capaces de detectar las fallas con facilidad al escribir el código.

Gestión de la construcción

El proceso de la planificación también define el orden en el que se crean los componentes según el método establecido. Se puede medir las numerosas actividades de la construcción incluyendo los códigos desarrollado, modificado, reutilizado, destruido, la complejidad de código, las estadísticas de verificación de código, las tasas de error entre otros. Estas mediciones pueden ser muy útiles durante el proceso de construcción para mejorar la calidad del software.

Consideraciones prácticas

La construcción del software es un proceso que tiene que ver con las restricciones del mundo real. Durante el proceso de construcción del software se realiza un diseño detallado y este diseño está determinado por las restricciones que impone el problema del mundo real que está siendo afrontado por el software.

Los lenguajes de construcción corresponden a los tipos de comunicación que se llevan a cabo durante el proceso. Entre ellos están: lenguaje de configuración (por ejemplo archivos de configuración de texto), lenguajes de herramientas (se utilizan para construir las aplicaciones a partir de las herramientas) y lenguajes de programación.

En el proceso de codificación se deben tomar en cuenta las siguientes consideraciones:

- El código fuente debe ser comprensible.
- Debe considerarse el tratamiento de errores tanto planeado como las excepciones.
- Deben prevenirse las brechas en la seguridad a nivel de código.
- El código fuente debe organizarse en rutinas, clases, paquetes y otras estructuras.
- El código debe documentarse.

Durante la construcción del software se realizan dos tipos de pruebas: Pruebas unitarias y Pruebas de integración. El propósito principal de estas pruebas es reducir el tiempo entre los momentos en que se introducen los fallos en el software y el tiempo en el que se detectan estos fallos. Las pruebas de construcción también sirven como una técnica para mejorar la calidad de software. Otras técnicas que se usan para mejorar la calidad durante la construcción de software son: el código paso por paso, utilización de aserciones, depuración, revisiones técnicas, etc.

Una de las actividades clave en el proceso de la construcción es la integración de los componentes de software construidos por separado. El proceso de la integración debe incluir la planificación de la secuencia en que se integran los componentes, de la estructura que soporta las versiones provisionales del software, determinar las pruebas que se realizarán a las versiones del software.

6. Aseguramiento de la calidad de Software

La calidad de software depende en su totalidad de la concordancia entre los requisitos planteados respecto a los obtenidos. Para medir la calidad de software se miden los atributos que lo conforman. El usuario final mide la calidad del software según lo que tenga o no, es en ese sentido de que la calidad del software depende de quien la juzgue. El hecho de que una empresa tenga certificación en calidad de software no garantiza que su software sea de calidad.

El aseguramiento de la calidad del software (en inglés Software Quality Assurance SQA) es un conjunto de actividades que se aplican a lo largo de todo el proceso de desarrollo del software y aportan la confianza en que el software satisfaga los requisitos de calidad. La SQA abarca los siguientes aspectos:

- Métodos y herramientas de análisis, diseño, codificación y prueba o Inspecciones técnicas formales o Una estrategia de prueba
- El control de documentación del software y de los cambios realizados.
- Procedimientos para ajustar el software a los estándares.
- Mecanismos de medida (métricas) y de información

La SQA comprende una serie de actividades:

1. **Aplicación de los métodos técnicos**

   La garantía de la calidad de software inicia con el conjunto de herramientas y métodos técnicos que apoyan al analista en obtener una especificación y un diseño de alta calidad.

2. **Realización de revisiones técnicas**

   La revisión técnica consiste básicamente en una reunión del personal técnico con el objetivo de descubrir los problemas de calidad.

3. **Prueba del software**

   La prueba del software abarca las estrategias de múltiples pasos y los métodos de diseño de casos para asegurar la detección efectiva de errores.

4. **Ajuste a los estándares**

   Si existen estándares formales, esta actividad garantiza que se siguen estos estándares.

5. **Control de cambios**

   El control de cambios contribuye directamente a la calidad de software ya que cualquier cambio realizado sobre el software puede introducir errores.

6. **Mediciones**

   El proceso de medición consiste en recolectar las métricas del software para asegurar la calidad.

7. **Registro y realización de informes**

   El registro y realización de informes brindan los procedimientos para la recolección y divulgación de la información de SQA.

Revisión del software

Las revisiones del software sirven para validar la calidad o el estado de un producto. El producto puede ser un documento, un módulo de software, un prototipo, etc. Las

revisiones se aplican en diferentes momentos del desarrollo del software y permiten detectar defectos que pueden ser eliminados. Existen diferentes tipos de revisiones, cada una especializada en un determinado producto. El beneficio más importante de las revisiones técnicas es la detección temprana de los defectos del software, lo que brinda la oportunidad de corregirlos antes de llegar a la siguiente etapa del diseño. El proceso de revisión reduce considerablemente los costos de los siguientes pasos de las fases del desarrollo.

Costo de la calidad

Sabemos que la calidad es importante, pero cuesta tiempo y dinero lograr el nivel de calidad en el software que en realidad queremos. No hay duda de que la calidad tiene un costo, pero la mala calidad también lo tiene - no sólo para los usuarios finales que deban vivir con el software defectuoso, sino también para la organización del software que lo elaboró y que debe darle mantenimiento -. La pregunta real es ésta: ¿por cuál costo debemos preocuparnos? Para responder a esta pregunta debe entenderse tanto el costo de tener calidad como el del software de mala calidad. El costo de la calidad incluye todos los costos en los que se incurre al buscar la calidad o al realizar actividades relacionadas con ella y los costos posteriores de la falta de calidad. Para entender estos costos, una organización debe contar con unidades de medición que proveen el fundamento del costo actual de la calidad, que identifiquen las oportunidades para reducir dichos costos y que den una base normalizada de comparación. El costo de la calidad puede dividirse en los costos que están asociados con la prevención, la evaluación y la falla.

Los costos de prevención incluyen lo siguiente:

- El costo de las actividades de administración requeridas para planear y coordinar todas las actividades de control y aseguramiento de la calidad.
- El costo de las actividades técnicas agregadas para desarrollar modelos completos de los requerimientos y del diseño.
- Los costos de planear las pruebas
- El costo de toda la capacitación asociada con estas actividades.

Los costos de evaluación incluyen las actividades de investigación de la condición del producto la "primera vez" que pasa por cada proceso. Algunos ejemplos de costos de evaluación incluyen los siguientes:

- El costo de efectuar revisiones técnicas de los productos del trabajo de la ingeniería de software.
- El costo de recabar datos y unidades de medida para la evaluación
- El costo de hacer las pruebas y depurar

Los costos de falla son aquellos que se eliminarían si no hubiera errores antes o después de enviar el producto a los consumidores. Los costos de falla se subdividen en internos y externos. Se incurre en costos internos de falla cuando se detecta un error en un producto antes del envío. Los costos externos de falla se asocian con defectos encontrados después de que el producto se envió a los consumidores. Algunos ejemplos de costos externos de falla son los de solución de quejas, devolución y sustitución del producto, ayuda en línea y trabajo asociado con la garantía. La mala reputación y la pérdida resultante de negocios es otro costo externo de falla que resulta difícil de

cuantificar y que, sin embargo, es real. Cuando se produce software de mala calidad, suceden cosas malas.

Como es de esperar, los costos relacionados con la detección y la corrección de errores o defectos se incrementan en forma abrupta cuando se pasa de la prevención a la detección, a la falla interna y a la externa.

7. Despliegue de software

El despliegue del software son todas las actividades que hacen que un sistema de software esté disponible para su uso.

Las estrategias de implementación pueden variar dependiendo del tipo de software: web, escritorio, móvil, etc.

Algunas recomendaciones para la implementación son las siguientes:

- **Usar una lista de verificación de implementación**

  Desplegar un nuevo software puede ser una tarea complicada, pero es una tarea simple una vez que se tiene un proceso. Nada prepara un equipo para el éxito como una lista de verificación. Se deben completar las tareas críticas antes y después del despliegue de software.

  Es recomendable tener indicadores del rendimiento del software para saber que con los nuevos cambios todo esté funcionando sin problemas.

- **Elegir las herramientas de implementación adecuadas**

Cada proyecto es diferente, por lo que no existe un conjunto perfecto de herramientas para cada equipo. En general, se desean herramientas que funcionen de forma nativa con la infraestructura de su aplicación y que se integren con sus otras herramientas.

- **Usar un servidor de integración continua**

  Una de las herramientas más importantes para una implementación exitosa es un servidor de integración continua (CI). Los servidores de CI extraen el código fuente de todos los desarrolladores y lo prueban juntos en tiempo real. Esto ayuda a los equipos a evitar el "infierno de integración", donde el código funciona en la estación de trabajo de un desarrollador, pero no en la rama principal.

  Los servidores de CI a veces se denominan "servidores de compilación" porque toman el código fuente y lo empaquetan en un artefacto de aplicación. La integración continua también incorpora principios de pruebas continuas, donde los equipos recopilan constantemente comentarios para detectar problemas lo antes posible.

- **Adoptar la entrega continua**

  La integración continua y la entrega continua (CD) a menudo se mencionan de una sola vez, pero son dos prácticas diferentes. Una vez que se integra el código y se genera la aplicación, el CD implica empaquetar y preparar el código para el despliegue. La aplicación se coloca en un entorno de preproducción o una réplica del servidor de producción real. Se realizan pruebas rigurosas durante la fase de

entrega para garantizar que la aplicación funcionará una vez implementados los cambios.

El objetivo de la entrega continua es tener aplicaciones que estén siempre listas para desplegarse. Esto no solo acelera la implementación del software, también se ha demostrado que produce software de mayor calidad.

- **Controlar sus KPI**

  Una vez que implemente una nueva versión de su software, observe sus métricas clave. Los KPI de implementación común incluyen la utilización del servidor, las tasas de excepción, el volumen de registro y el rendimiento de la base de datos. Si hay un problema de rendimiento, también es importante saber de dónde viene el problema.

- **Tener una estrategia de reversión**

  Desea poder implementar con confianza, y eso significa siempre tener una estrategia de reversión o rollback en caso de que algo salga mal. Cuando se produce un error al implementar una nueva versión, a veces la mejor solución es volver a la última versión de trabajo conocida del software. Puede conservar una copia de seguridad de su última versión hasta que sepa que la nueva funciona correctamente.

8. Mantenimiento

Los esfuerzos de desarrollo de software resultan en la entrega de un producto de software que satisface los requisitos del usuario. En consecuencia, el producto de

software debe cambiar o evolucionar. Una vez en funcionamiento, se descubren los defectos, los entornos operativos cambian y aparecen nuevos requisitos de los usuarios. La fase de mantenimiento del ciclo de vida comienza después de un período de garantía o entrega de soporte posterior a la implementación, pero las actividades de mantenimiento ocurren mucho antes. El mantenimiento del software es una parte integral de un ciclo de vida del software. Sin embargo, no ha recibido el mismo grado de atención que las otras fases. Históricamente, el desarrollo de software ha tenido un perfil mucho más alto que el mantenimiento de software en la mayoría de las organizaciones. Esto ahora está cambiando, ya que las organizaciones se esfuerzan por aprovechar al máximo su inversión en desarrollo de software manteniendo el software operando el mayor tiempo posible.

El mantenimiento es necesario para garantizar que el software continúe satisfaciendo los requisitos del usuario. El mantenimiento se aplica al software que se desarrolla utilizando cualquier modelo de ciclo de vida del software (por ejemplo, espiral o lineal). Los productos de software cambian debido a acciones de software correctivas y no correctivas. El mantenimiento debe realizarse para corregir fallas; mejorar el diseño; implementar mejoras; interfaz con otro software; adaptar los programas para que se puedan utilizar diferentes hardware, software, características del sistema e instalaciones de telecomunicaciones; migrar software heredado; y retire el software. Cinco características clave comprenden las actividades del mantenedor: mantener el control sobre las funciones cotidianas del software; mantener el control sobre la modificación del software; perfeccionando funciones existentes; identificando amenazas de seguridad y

arreglando vulnerabilidades de seguridad; y evitar que el rendimiento del software se degrade a niveles inaceptables.

Para lograr un adecuado mantenimiento al sistema, se hacen las siguientes sugerencias:

- Tener un seguimiento del sistema, a efecto de evaluar la alimentación de datos, ejecución de procesos, revisión de resultados y respaldo de la información.
- Evaluar el empleo de recursos y tiempo de respuesta que ocupa el sistema en su operación, para optimizarla.
- Analizar detalladamente las fallas detectadas, así como los nuevos requerimientos antes de iniciar cualquier alteración.
- Incorporar las experiencias y avances tecnológicos para la optimización del sistema.
- Respaldar el sistema antes de comenzar los cambios.
- Durante la modificación, respetar las normas, estándares y procedimientos que han respaldado la construcción del sistema.

9. Bibliografía

- Pressman, 2010. Ingeniería del Software: Un Enfoque Práctico. 7ª Edición. McGraw-Hill.
- “Advantages of the “As a user, I want” user story template”, Mike Cohn, 2008
- http://www.axia-consulting.co.uk/html/requirements_gathering.html
- https://www.mountaingoatsoftware.com/agile/user-stories
- http://www.pmoinformatica.com/2015/05/historias-de-usuario-ejemplos.html
- http://www.ambysoft.com/essays/deploymentTips.html

- http://swebokwiki.org/Chapter_5:_Software_Maintenance

Descargo de responsabilidad

La información contenida en este documento descargable en formato PDF o PPT es un reflejo del material virtual presentado en la versión online del curso. Por lo tanto, su contenido, gráficos, links de consulta, acotaciones y comentarios son responsabilidad exclusiva de su(s) respectivo(s) autor(es) por lo que su contenido no compromete al área de e-Learning del Departamento GES o al programa académico al que pertenece.

El área de e-Learning no asume ninguna responsabilidad por la actualidad, exactitud, obligaciones de derechos de autor, integridad o calidad de los contenidos proporcionados y se aclara que la utilización de este descargable se encuentra limitada de manera expresa para los propósitos educacionales del curso.