Ingeniería de software 1

Teórico

Alumno: Santiago Vietto

Docente: Alejandra Bosio

DNI: 42654882

Institución: UCC

Año: 2021

Los procesos de software

Introducción

Concepto básico de ingeniería de software

Ingeniería de software: es una parte de la ingeniería que aplica teorías, métodos y herramientas para el desarrollo de software profesional. Con profesional hacemos referencia a un desarrollo o nivel casi industrial, es decir, cuando tenemos que empezar a resolver problemas para otros y generar herramientas de solución para problemas importantes, hace falta encararlos de manera profesional y de una forma más formal. La ingeniería de software es importante porque:

- La economía de muchos países depende en alguna medida del software.
- Cada vez más sistemas son controlados por software.
- El gasto en software representa una fracción significativa del PBI.
- Los errores de software pueden ser muy caros (tanto económicos como en vidas humanas).

_ El software hoy por hoy está en todos lados, y no se nos ocurre actividad que se pueda realizar con un producto de software en el medio. entonces siendo así la situación, tiene que haber alguna forma de arreglar, definir y de establecer de qué manera se puede trabajar con eso.

Disciplina de Ingeniería: el uso de las teorías y los métodos adecuados para resolver los problemas teniendo en cuenta las limitaciones financieras y de organización. Cualquier cosa que sea una disciplina de la ingeniería implica que tiene ciertos métodos, reglas, teorías que avalan que las cosas se hagan de determinada manera pero teniendo en cuenta las limitaciones que hay para eso. En ingeniería, en cualquier rama, siempre hay que encontrar la mejor solución, en el menor tiempo y gastando lo menos posible.

Errores de software

_ Los errores en el software son un problema común y en muchos casos, no hay graves consecuencias, se soluciona con una nueva versión corregida en muchos otros casos puede haber consecuencias fatales, graves o muy caras. Debemos tomar conciencia de que los errores de software en general se pueden detectar y resolver sin consecuencias tan graves, pero hay un montón de situaciones donde las consecuencias son muy graves, es por eso que hace falta la ingeniería de software justamente como una rama de la ingeniería, es decir, una actividad en la cual podamos establecer reglas o buenas prácticas de trabajo, y que podamos tener métricas, o sea, que podamos medir lo que estamos haciendo, de tal forma que los resultados sean los mejores posibles, y que si hay errores graves, seamos capases de prevenirlos y resolverlos a tiempo.

Costos del software

_ El software tiene un ciclo de vida, y ese ciclo es muy largo. El ciclo empieza en la idea, en la necesidad que tiene una organización o alguien, de tener un sistema de software capaz de resolver determinada cosa. Luego este ciclo va pasando por distintas etapas, que es primero entender que es lo que necesitamos, despues poderlo diseñar, poder definir su arquitectura, poder implementarlo, luego poder testearlo o validarlo, ponerlo en operación. Por más que el software este en operación, este tiene una etapa de mantenimiento.

_ El costo del software suele ser mayor que el costo del hardware. El mantenimiento del software cuesta más que el desarrollo del mismo. Para sistemas que tienen una larga vida, los costos de mantenimiento superan ampliamente los costos de desarrollo. Si un desarrollo no está bien hecho y empiezan a saltar errores, esos errores hay que corregirlos, y ahí es cuando el mantenimiento es caro, ya que hay que hacer bien las cosas que antes se hicieron mal. Entonces uno de los trabajos de la ingeniería de software es que el desarrollo de software sea rentable, es decir, que se haga bien de forma que despues el mantenimiento implique fundamentalmente evolución y no corrección. Resumiendo decimos que la ingeniería de software tiene que ver con el desarrollo de software rentable.

Productos de software

_ Cuando hablamos de software a nivel profesional, este no solamente está formado por programas o código, sino que están formados por un conjunto de programas, y además de eso tiene un conjunto de documentación relacionada con esos programas. Entonces cuando hablamos de software hablamos de algo mucho más grande, que es el conjunto de programas que resuelve determinada situacion o que da un servicio, sumado a toda la documentación relacionada con eso. Por lo general los desarrolladores tienen el problema de resistirse a la documentación, y la forma en que las cosas se hagan bien es que estas estén documentadas. Con respecto a la documentación, vamos a ver todo lo que implica la documentación de los requerimientos, es decir, documentar que es lo que nuestro software debe hacer, si no lo tenemos escrito en ningún lado no va a ser posible, también documentamos que diseño se hizo, como se eligió la arquitectura, como se realizaron o se van a realizan las pruebas y que resultados se obtuvo, todo esto es parte de la documentación, inclusive un manual de usuario es parte del software. Entonces cuando hablamos de software ya a nivel profesional estamos hablando de un conjunto de programas que generan o dan valor de alguna manera, pero a eso sumado toda la documentación relacionada con ese conjunto de programas.

Especificaciones del producto

_ Dado el concepto de software que ahora tenemos, podemos ver las siguientes clasificaciones:

Productos genéricos: sistemas que se comercializan y venden a cualquier cliente, por ejemplo software para gráficos, herramientas de gestión de proyecto, software CAD, software para mercados específicos (sistema de turno para dentista). La especificación de lo que el software debe hacer es propiedad del desarrollador del software y las decisiones sobre los cambios en el software son hechas por el desarrollador. Por ejemplo se desarrolla un software que consideramos de que va a ser útil, donde luego hay un montón de gente que ve la utilidad de eso y lo comienza a utilizar, y es la persona a la que se le ocurre la idea del software quien especifica los requerimientos, que pueden evolucionar agregándole más funcionalidades, agregamos también que la gestión de esto tiene determinadas características.

Productos personalizados: software que encarga un cliente específico para satisfacer sus propias necesidades. Por ejemplo sistemas de control o monitoreo, software de control del tráfico aéreo, etc. La especificación de lo que el software debe hacer es propiedad del cliente del software y él es el que toma decisiones sobre los cambios de software necesarios. Este caso es más general, donde tenemos un cliente ya sea una persona, empresa u organización, que requiere un producto para resolver un problema que está teniendo, y en ese caso la especificación de los requerimientos implica que nosotros entendamos muy bien cuál es la necesidad de ese cliente.

Preguntas sobre la ingeniería de software

_ Cuando hablamos sobre ingeniería de software nos tenemos que contestar las siguientes preguntas:

¿Qué es software?	Programas de cómputo y documentación asociada. Los productos de software se desarrollan para un cliente en particular o para un mercado en general.
¿Cuáles son los atributos del buen software?	El buen software debe entregar al usuario la funcionalidad y el desempeño requeridos, y debe ser sustentable, confiable y utilizable.
¿Qué es ingeniería de software?	La ingeniería de software es una disciplina de la ingeniería que se interesa por todos los aspectos de la producción de software.
¿Cuáles son las actividades fundamentales de la ingeniería de software?	Especificación, desarrollo, validación y evolución del software.
¿Cuál es la diferencia entre ingeniería de software y ciencias de la computación?	Las ciencias de la computación se enfocan en teoría y fundamentos;

	mientras la ingeniería de software se enfoca en el sentido práctico del desarrollo y en la distribución de software.
¿Cuál es la diferencia entre ingeniería de software e ingeniería de sistemas?	La ingeniería de sistemas se interesa por todos los aspectos del desarrollo de sistemas basados en computadoras, incluidos hardware, software e ingeniería de procesos. La ingeniería de software es parte de este proceso más general.
¿Cuáles son los principales retos que enfrenta la ingeniería de software?	Tiempos de distribución limitados y desarrollo de software confiable.
¿Cuáles son los costos de la ingeniería de software?	Aproximadamente 60% de los costos del software son de desarrollo, y 40% de prueba. Para el software elaborado específicamente, los costos de evolución superan con frecuencia los costos de desarrollo.
¿Cuáles son los mejores métodos y técnicas de la ingeniería de software?	Aun cuando todos los proyectos de software deben gestionarse y desarrollarse de manera profesional, existen diferentes técnicas que son adecuadas para distintos tipos de sistema. Por ejemplo, los juegos siempre deben diseñarse usando una serie de prototipos, mientras que los sistemas críticos de control de seguridad requieren de una especificación completa y analizable para su desarrollo. Por lo tanto, no puede decirse que un método sea mejor que otro.
¿Qué diferencias ha marcado la Web a la ingeniería de software?	La Web ha llevado a la disponibilidad de servicios de software y a la posibilidad de desarrollar sistemas basados en servicios distribuidos ampliamente. El desarrollo de sistemas basados en Web ha conducido a importantes avances en lenguajes de programación y reutilización de software.

Atributos esenciales de un buen software

_ Esta bueno tener en mente que si la ingeniería de software tiene por objetivo construir software profesional y de buena calidad, y se establecen métricas para ver si ese proceso se realiza bien o no, algunas características del software son las que nos deberían mostrar las cosas fundamentales para ver si el software es bueno o no, y vemos que de las siguientes características, algunas son visibles desde afuera y otras no.

Características del producto	**Descripción**
Mantenimiento	El software debe escribirse de tal forma que pueda evolucionar para satisfacer las necesidades cambiantes de los clientes. Éste es un atributo crítico porque el cambio del software es un requerimiento inevitable de un entorno empresarial variable.
Confiabilidad y seguridad	La confiabilidad del software incluye una variedad de características incluyendo confiabilidad y seguridad. El software fiable no debe causar daño físico o económico en caso de fallo del sistema. Los usuarios malintencionados no deben poder acceder o dañar el sistema.
Eficiencia	El software debe optimizar el uso de los recursos del sistema, como la memoria y los ciclos del procesador. Por lo tanto, la eficiencia incluye la capacidad de respuesta, el tiempo de procesamiento, la utilización de la memoria, etc.
Aceptabilidad	El software debe ser aceptable para el tipo de usuario para el que está diseñado. Esto significa que debe ser comprensible, utilizable y compatible con otros sistemas que utilizan.

- Cuando hablamos de mantenimiento en general tiene que ver con el código ya a nivel de implementación. Es importante que el software sea de fácil mantenimiento porque es un producto que despues de su puesta en producción puede cambiar mucho, entonces la forma de que eso se pueda hacer es que el código sea mantenible y este codificado de acuerdo a buenas prácticas como por ejemplo hacer módulos pequeños, resolver problemas de manera separada, ver que un cambio no impacte al resto, no trabajar con una estructura monolítica sino más bien de microservicio, pero también otra cosa

que impacta mucho en la facilidad de mantenimiento es la cantidad de documentación que se tenga, porque si no la tenemos y realizamos un cambio que impacta a otros requerimientos, tendremos problemas. Por ende para lograr que el software tenga un fácil mantenimiento implica codificar bien siguiendo buenas prácticas de codificación, pero también implica tener documentación que avale que es lo que se hizo.

- Con confiable y seguro nos referimos a que si un usuario o cliente no ve que el sistema que le hemos dado tiene una cierta seguridad y que efectivamente no causa desastres, si no siente confianza o seguridad entonces no va a utilizar el software. El sistema debe garantizar fundamentalmente de que no haga daños tanto económicos como humanos.
- Eficiencia hace referencia a que se tienen que manejar bien los recursos, no sería bueno un sistema que no haga un buen uso de los recursos a nivel de memoria, que para desplegar una opción demore mucho tiempo, etc.
- Y con aceptabilidad decimos que el software debe ser aceptable, es decir, el usuario lo tiene que poder usar, porque lo peor que le puede pasar a un sistema es que nadie lo use por más bien que este hecho y por lo tanto es un fracaso.

Modelos de proceso de software

Introducción

Proceso de software: es una serie de actividades relacionadas que conduce a la elaboración de un producto de software. Estas actividades pueden incluir el desarrollo de software desde cero en un lenguaje de programación estándar como Java o C. Sin embargo, las aplicaciones de negocios no se desarrollan precisamente de esta forma. El nuevo software empresarial con frecuencia ahora se desarrolla extendiendo y modificando los sistemas existentes, o configurando e integrando el software comercial o componentes del sistema.

Actividades del proceso de software

_ Existen muchos diferentes procesos de software, pero todos deben incluir cuatro actividades que son fundamentales para la ingeniería de software:

Especificación del software: tienen que definirse tanto la funcionalidad del software como las restricciones de su operación. Es decir, esta es la parte donde quien necesita el producto, o sea el cliente, es capaz de decirnos que es lo que hace falta. Es un proceso conjunto entre el cliente que es el que está requiriendo algo, y el ingeniero de requerimientos que es el que está tratando de entender que es lo que necesita. Esto es un proceso conjunto porque se tiene que lograr garantizar de que se entendió que es lo que se necesita y que se logró que ese cliente sea capaz de desglosar tanto que es lo que se necesita para que asi se pueda entender sin ambigüedades. Resolver la ambigüedad es uno de los mayores problemas de software.

Diseño e implementación del software: debe desarrollarse el software para cumplir con las especificaciones. Se diseña y programa el software, además del modelado y la construcción. Se pasa de un proyecto a un ente ejecutable.

Validación del software: hay que validar el software para asegurarse de que cumple lo que el cliente quiere. Es decir, verificamos si hicimos lo que el cliente necesitaba, y si lo hicimos bien o lo hicimos mal.

Evolución del software: el software tiene que evolucionar para satisfacer las necesidades cambiantes del cliente y el mercado. Esto es cuando el software ya está en operación y debemos corregir las cosas que se nos hayan escapado, y evolucionar, agrandar, mejorar, etc, todo lo que vaya siendo necesario hacer.

_ En cierta forma, tales actividades forman parte de todos los procesos de software. En la práctica éstas son actividades complejas en sí mismas e incluyen subactividades tales como la validación de requerimientos, el diseño arquitectónico, la prueba de unidad, etc. También existen actividades de soporte al proceso, como la documentación y el manejo de la configuración del software.

Descripciones de procesos de software

_ Cuando los procesos se discuten y describen, por lo general se habla de actividades como especificar un modelo de datos, diseñar una interfaz de usuario, etc, así como del orden de dichas actividades. Sin embargo, al igual que las actividades, también las descripciones de los procesos deben incluir:

- Productos: que son los resultados de una actividad del proceso. Por ejemplo, el resultado de la actividad del diseño arquitectónico es un modelo de la arquitectura de software.
- Roles: que reflejan las responsabilidades de la gente que interviene en el proceso. Ejemplos de roles: gerente de proyecto, gerente de configuración, programador, etcétera.
- Precondiciones y postcondiciones: que son declaraciones válidas antes y después de que se realice una actividad del proceso o se cree un producto. Por ejemplo, antes de comenzar el diseño arquitectónico, una precondición es que el cliente haya aprobado todos los requerimientos; después de terminar esta actividad, una postcondición podría ser que se revisen aquellos modelos UML que describen la arquitectura.

_ No hay un proceso ideal la mayoría de las organizaciones han diseñado sus propios procesos de desarrollo de software. Para algunos sistemas, como los sistemas críticos, se requiere de un proceso de desarrollo muy estructurado. Para los sistemas empresariales, con requerimientos rápidamente cambiantes, es probable que sea más efectivo un proceso menos formal y flexible. En ocasiones, los procesos de software se clasifican como dirigidos por un plan (plan driven) o como procesos ágiles:

Procesos dirigidos por un plan: son aquellos donde todas las actividades del proceso se planean por anticipado y el avance se mide contra dicho plan. Además se ve como las actividades se encadenan unas con otras.

Procesos ágiles: la planeación es incremental y es más fácil modificar el proceso para reflejar los requerimientos cambiantes del cliente.

_ Cada enfoque es adecuado para diferentes tipos de software. Por lo general, uno necesita encontrar un equilibrio entre procesos dirigidos por un plan y procesos ágiles.

Detalles generales que afectan la mayoría del software

_ La mayoría de los problemas que nos enfrentamos cuando generamos software

Heterogeneidad: cada vez con mayor frecuencia se requieren sistemas que operen como sistemas distribuidos a través de redes que incluyan diferentes tipos de computadoras y dispositivos móviles. Es decir, cada vez el software pasa por todos lados y no es que hacemos un desarrollo para una cosa puntual, sino que es probable que esa cosa o desarrollo tenga que interactuar con otros, tenga que usarse en distintas plataformas y tenga que compartir y generar resultados a hacia distintos lugares, entonces ya no queda una cosa muy acotada.

Cambio empresarial y social: los negocios y la sociedad cambian de manera rápida, conforme se desarrollan las economías emergentes y nuevas tecnologías están a la disposición. Ambos necesitan tener la posibilidad de cambiar su software existente y desarrollar rápidamente uno nuevo. El software está inmerso en un mundo que generalmente es empresarial o tiene características sociales y que va cambiando, entonces el software se tiene que ir adaptando a esos cambios porque cada vez va a ir necesitando resolver cada vez más cosas, y el problema se va a ir poniendo cada vez más grande.

Seguridad y confianza: dado que el software está vinculado con todos los aspectos de la vida, es esencial confiar en dicho software.

Fundamentos de la ingeniería de software

_ Algunos principios fundamentales se aplican a todos los tipos de sistema de software, con independencia de las técnicas de desarrollo utilizados:

- Los sistemas deben ser desarrollados mediante un proceso de desarrollo dirigido y entendido. Diferentes procesos se utilizan para diferentes tipos de software.
- La fiabilidad y el rendimiento son importantes para todos los tipos de sistema.
- La comprensión y la gestión de la especificación de requisitos de software y (lo que el software debe hacer) son fundamentales.
- Siempre que sea posible reutilizar el software ya creado en lugar de desarrollar uno nuevo.

Modelos de proceso de software

_ Un modelo de proceso de software es una representación simplificada de este proceso. Cada modelo del proceso representa a otro desde una particular perspectiva y, por lo tanto, ofrece sólo información parcial acerca de dicho proceso. Tales modelos genéricos no son descripciones definitivas de los procesos de software, más bien, son abstracciones del proceso que se utilizan para explicar los diferentes enfoques del desarrollo de software. Se pueden considerar marcos del proceso que se extienden y se adaptan para crear procesos más específicos de ingeniería de software. Los modelos del proceso que se examinan aquí son:

Modelo en cascada (waterfall): esta toma las actividades fundamentales del proceso de especificación, desarrollo, validación y evolución y, luego, los representa como fases separadas del proceso, tal como especificación de requerimientos, diseño de software, implementación, pruebas, etcétera. Este modelo es dirigido por plan, ya que es un modelo super estructurado y sumamente rígido, donde hay diferentes fases separadas con cada una un objetivo específico y una cronología especifica cuya dinámica no se puede modificar.

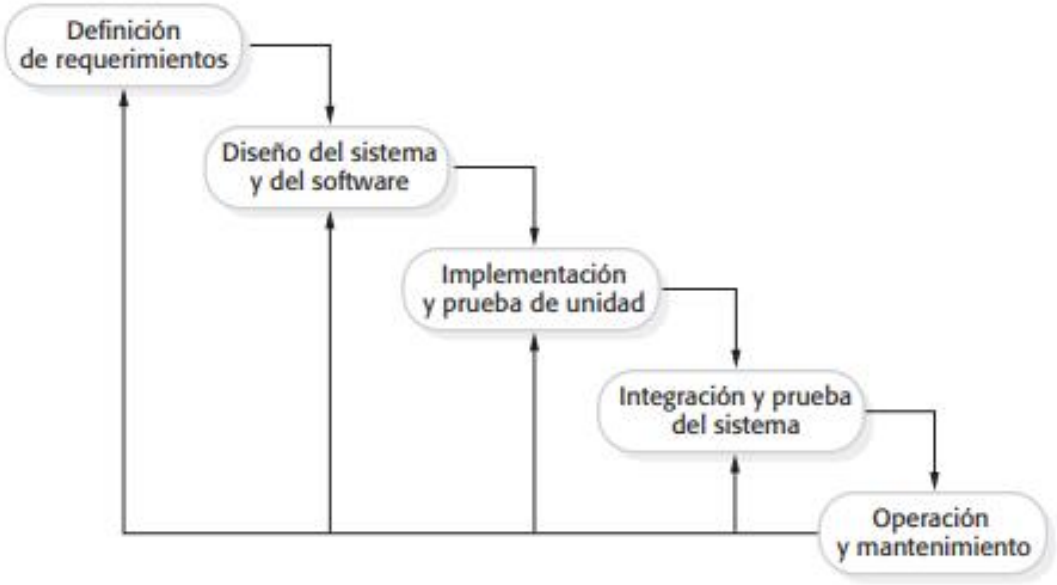
Desarrollo incremental: no se centra en una fase, donde solo se haga una especificación y cuando esté lista pasar a otra fase, sino que permite trabajar de una forma más fluida que incluya o vincule las actividades de especificación, desarrollo y validación. El sistema se desarrolla como una serie de versiones (incrementos), y cada versión añade funcionalidad a la versión anterior, en donde el producto final no se obtiene de una sola vez, sino que va creciendo y se va utilizando a medida que va creciendo. Este puede ser el dirigido por plan o ágil.

Ingeniería de software orientada a la reutilización: este enfoque se basa en la existencia de un número significativo de componentes reutilizables. El proceso de desarrollo del sistema se enfoca en la integración de estos componentes en un sistema, en vez de desarrollarlo desde cero. El sistema se ensambla a partir de componentes existentes Puede ser el dirigido por plan o ágil.

_ Dichos modelos no son mutuamente excluyentes y con frecuencia se usan en conjunto, sobre todo para el desarrollo de grandes sistemas. Para este tipo de sistemas, tiene sentido combinar algunas de las mejores características de los modelos de desarrollo en cascada e incremental.

El modelo en cascada

_ El modelo en cascada recibe su nombre justamente por el esquema que vemos a continuación, donde tiene fases muy bien definidas, y cuando termina una fase se puede pasar a la que sigue y así sucesivamente hasta llegar a un sistema entregado y en operación. Este modelo se conoce además como ciclo de vida del software, y es un ejemplo de un proceso dirigido por un plan. En este tenemos que planear y programar todas las actividades del proceso, antes de comenzar a trabajar con ellas.



_ Las principales etapas del modelo en cascada son:

Análisis y definición de requerimientos: los servicios, las restricciones y las metas del sistema se establecen mediante consulta a los usuarios del sistema. Luego, se definen con detalle y sirven como una especificación del sistema. Se determina todo lo que el sistema debe y no debe hacer y documentarlo de manera adecuada.

Diseño del sistema y del software: el proceso de diseño de sistemas asigna los requerimientos, para sistemas de hardware o de software, al establecer una arquitectura de sistema global. El diseño del software implica identificar y describir las abstracciones fundamentales del sistema de software y sus relaciones.

Implementación y prueba de unidad: durante esta etapa, el diseño de software se rea liza como un conjunto de programas o unidades del programa. La prueba de unidad consiste en verificar que cada unidad cumpla con su especificación.

Integración y prueba de sistema: las unidades del programa o los programas individuales se integran y prueban como un sistema completo para asegurarse de que se cumplan los requerimientos de software. Después de probarlo, se libera el sistema de software al cliente.

Operación y mantenimiento: por lo general (aunque no necesariamente), ésta es la fase más larga del ciclo de vida, donde el sistema se instala y se pone en práctica. El mantenimiento incluye corregir los errores que no se detectaron en etapas anteriores del ciclo de vida, mejorar la implementación de las unidades del sistema e incrementar los servicios del sistema conforme se descubren nuevos requerimientos.

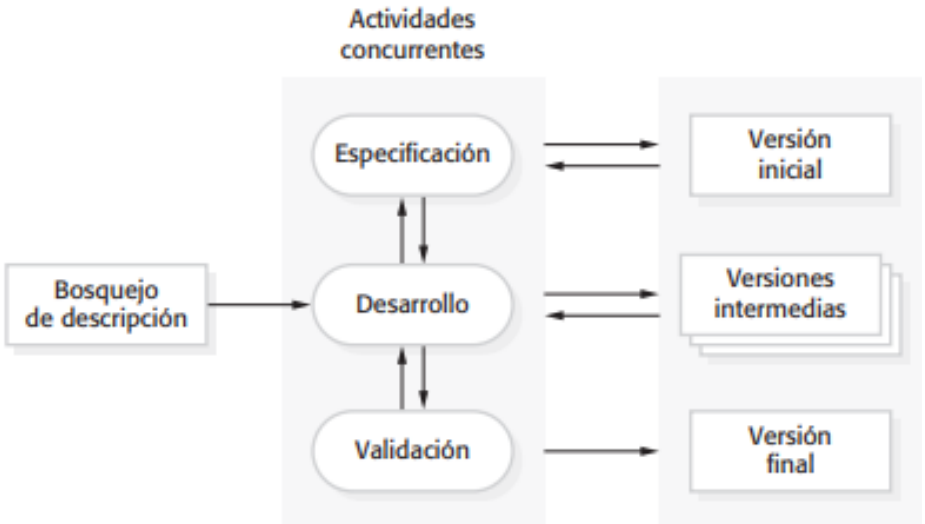
_ La siguiente fase no debe comenzar sino hasta que termine la fase previa. En la práctica, dichas etapas se traslapan y se nutren mutuamente de información. El proceso de software no es un simple modelo lineal, sino que implica retroalimentación de una fase a otra. Entonces, es posible que los documentos generados en cada fase deban modificarse para reflejar los cambios que se realizan. Debido a los costos de

producción y aprobación de documentos, las iteraciones suelen ser onerosas e implicar un rediseño significativo. Por lo tanto, después de un pequeño número de iteraciones, es normal detener partes del desarrollo, como la especificación, y continuar con etapas de desarrollo posteriores. Los problemas se dejan para una resolución posterior, se ignoran o se programan.

_ Su principal problema es la partición inflexible del proyecto en distintas etapas. Tienen que establecerse compromisos en una etapa temprana del proceso, lo que dificulta responder a los requerimientos cambiantes del cliente. En principio, el modelo en cascada sólo debe usarse cuando los requerimientos se entiendan bien y sea improbable el cambio radical durante el desarrollo del sistema, es decir es beneficioso cuando los requerimientos no cambian y donde el tiempo de desarrollo puede ser muy largo. No utilizamos esto en software para juegos o facturación, por ejemplo, ya que se utiliza para proyectos grandes de área científica, militar, etc. Y otro problema muy grande que tiene este modelo, es que el cliente no ve nada hasta que se lo entregamos de forma completa, por lo tanto, no hay feedback con el cliente.

Desarrollo incremental

_ El desarrollo incremental se basa en la idea de diseñar una implementación inicial, exponer ésta al comentario del usuario, y luego desarrollarla en sus diversas versiones hasta producir un sistema adecuado. Las actividades de especificación, desarrollo y validación están entrelazadas en vez de separadas, con rápida retroalimentación a través de las actividades.



_ El desarrollo de software incremental, que es una parte fundamental de los enfoques ágiles, es mejor que un enfoque en cascada para la mayoría de los sistemas empresariales, de comercio electrónico y personales. Refleja la forma en que se resuelven problemas. Se avanza en una serie de pasos hacia una solución y se retrocede cuando se detecta que se cometieron errores.

_ Resulta más barato y fácil realizar cambios en el software conforme éste se diseña. Cada incremento o versión del sistema incorpora algunas de las funciones que necesita el cliente. Por lo general, los primeros incrementos del sistema incluyen la función más importante o la más urgente. Esto significa que el cliente puede evaluar el desarrollo del sistema en una etapa relativamente temprana, para constatar si se entrega lo que se requiere. En caso contrario, sólo el incremento actual debe cambiarse y, posiblemente, definir una nueva función para incrementos posteriores.

_ Ventajas por sobre el modelo en cascada:

- Se reduce el costo de adaptar los requerimientos cambiantes del cliente. La cantidad de análisis y la documentación que tiene que reelaborarse son mucho menores de lo requerido con el modelo en cascada.
- Es más sencillo obtener retroalimentación o feedback del cliente sobre el trabajo de desarrollo que se realizó. Los clientes pueden comentar las demostraciones del software y darse cuenta de cuánto se ha implementado. Los clientes encuentran difícil juzgar el avance a partir de documentos de diseño de software.
- Es posible que sea más rápida la entrega e implementación de software útil al cliente, aun si no se ha incluido toda la funcionalidad. Los clientes tienen posibilidad de usar y ganar valor del software más temprano de lo que sería posible con un proceso en cascada.

_ Desventajas del desarrollo incremental:

- Es difícil tener una métrica de cómo se va avanzando, pero además de eso los sucesivos incrementos pueden degradar la calidad tanto de código como de la documentación si es que no se es riguroso con eso. Mientras más degradada este la estructura del sistema, las entregas necesariamente se van a retrasar porque no hay forma de llegar. Entonces es fundamental buscar mecanismos para evitar esto.
- El proceso no es visible. Los administradores necesitan entregas regulares para medir el avance. Si los sistemas se desarrollan rápidamente, resulta poco efectivo en términos de costos producir documentos que reflejen cada versión del sistema.
- La estructura del sistema tiende a degradarse conforme se tienen nuevos incrementos. A menos que se gaste tiempo y dinero en la refactorización para mejorar el software, el cambio regular tiende a corromper su estructura. La incorporación de más cambios de software se vuelve cada vez más difícil y costosa.

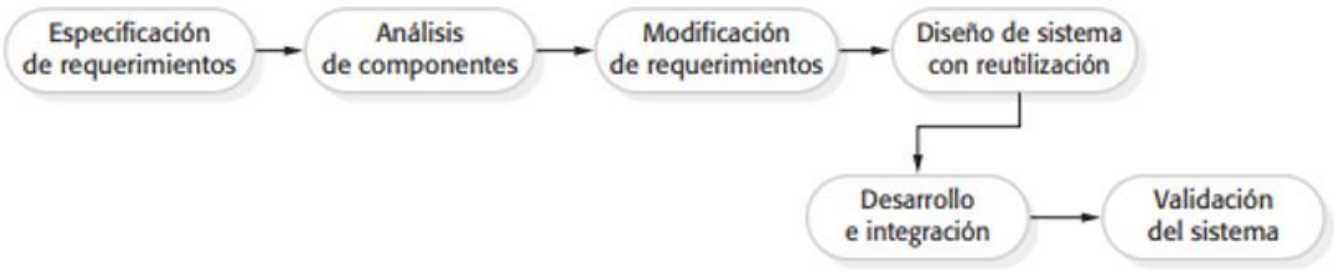
_ Los problemas del desarrollo incremental se tornan particularmente agudos para sistemas grandes, complejos y de larga duración, donde diversos equipos desarrollan diferentes partes del sistema. Naturalmente lo que pasa en el desarrollo incremental es que se deteriora la documentación, por ende, se deteriora la calidad del código, por lo que debemos ser muy estrictos para aplicar el método de la manera correcta y evitar que eso pase. Si se deteriora la documentación y la mantenibilidad del código, se

desvirtúa también el objetivo del desarrollo incremental, porque al ser incremental estamos agregando cosas sobre el mismo código, entonces si no es mantenible cada vez va a ser peor y más difícil a volver a tener el sistema como corresponde.

Ingeniería de software orientada a la reutilización

_ En la mayoría de los proyectos de software hay cierta reutilización de software. Sucede con frecuencia de manera informal, cuando las personas que trabajan en el proyecto conocen diseños o códigos que son similares a lo que se requiere. Los buscan, los modifican según se necesite y los incorporan en sus sistemas. Esta reutilización informal ocurre independientemente del proceso de desarrollo que se emplee.

_ No es que sea una metodología en sí, sino que son cosas que se pueden aplicar en cualquiera de las metodologías y hoy por hoy está muy difundido la cuestión de reutilizar, ya que si tenemos algo que funciona y fue testeado u homologado de alguna manera, no habría razón para no usarlo. Sin embargo las cosas no se pueden reutilizar tan fácilmente, entonces en general se dice que la ingeniería de software orientada a la reutilización se basa en la reutilización sistemática de código, los sistemas se integran a partir de componentes o sistemas existentes.



_ Dichas etapas son:

Especificación de requerimientos: del producto que tenemos que generar, esta es la etapa inicial y de validación.

Análisis de componentes: dada la especificación de requerimientos, se realiza una búsqueda de componentes para implementar dicha especificación. Por lo general, no hay coincidencia exacta y los componentes que se usan proporcionan sólo parte de la funcionalidad requerida.

Modificación de requerimientos: durante esta etapa se analizan los requerimientos usando información de los componentes descubiertos. Luego se modifican para reflejar los componentes disponibles. Donde las modificaciones son imposibles, puede regresarse a la actividad de análisis de componentes para buscar soluciones alternativas.

Diseño de sistema con reutilización: durante esta fase se diseña el marco conceptual del sistema o se reutiliza un marco conceptual existente. Los creadores toman en cuenta los componentes que se reutilizan y organizan el marco de referencia para atenderlo. Es posible que deba diseñarse algo de software nuevo, si no están disponibles los componentes reutilizables.

Desarrollo e integración: se diseña el software que no puede procurarse de manera externa, y se integran los componentes y los sistemas COTS para crear el nuevo sistema. La integración del sistema, en este modelo, puede ser parte del proceso de desarrollo, en vez de una actividad independiente

_ Existen tres tipos de componentes de software que pueden usarse en un proceso orientado a la reutilización:

- Servicios Web: que se desarrollan en concordancia para atender servicios estándares y que están disponibles para la invocación remota.
- Colecciones de objetos: que se desarrollan como un paquete para su integración con un marco de componentes como .NET o J2EE.
- Sistemas de software independientes: que se configuran para usar en un entorno particular.

_ La ingeniería de software orientada a la reutilización tiene la clara ventaja de reducir la cantidad de software a desarrollar y, por lo tanto, la de disminuir costos y riesgos; por lo general, también conduce a entregas más rápidas del software. Sin embargo, son inevitables los compromisos de requerimientos y esto conduciría hacia un sistema que no cubra las necesidades reales de los usuarios. Más aún, se pierde algo de control sobre la evolución del sistema, conforme las nuevas versiones de los componentes reutilizables no estén bajo el control de la organización que los usa.

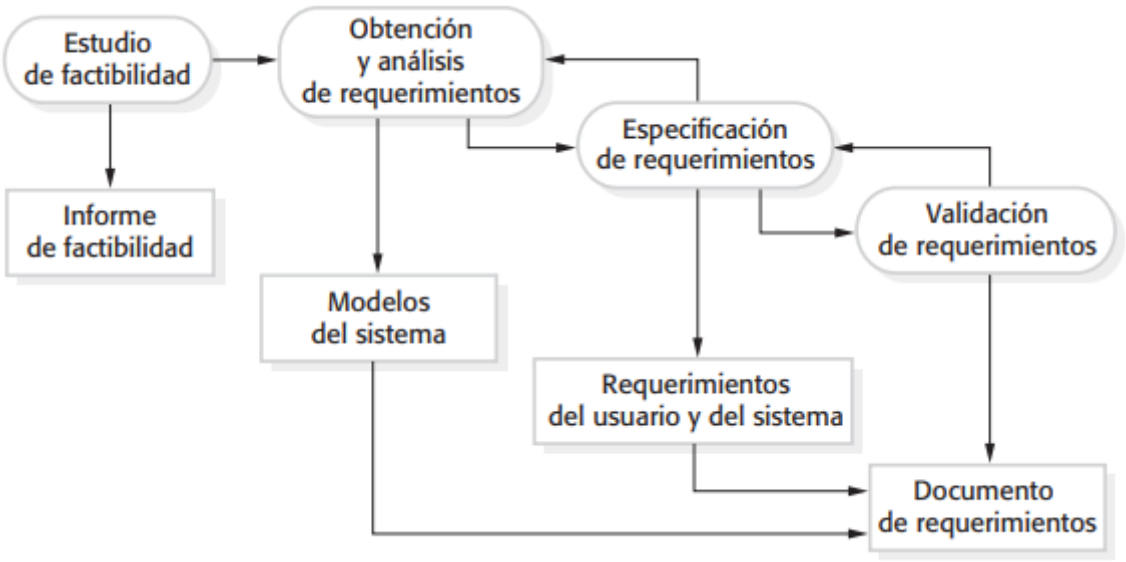
Actividades del proceso

_ Los procesos de software reales son secuencias entrelazadas de actividades técnicas, colaborativas y administrativas con la meta general de especificar, diseñar, implementar y probar un sistema de software. Las herramientas son útiles particularmente para dar apoyo a la edición de distintos tipos de documento y para manejar el inmenso volumen de información detallada que se reproduce en un gran proyecto de software. Las cuatro actividades básicas de proceso de especificación, desarrollo, validación y evolución se organizan de diversa manera en diferentes procesos de desarrollo. En el modelo en cascada se organizan en secuencia, mientras que se entrelazan en el desarrollo incremental.

Especificación del software

_ La especificación del software o la ingeniería de requerimientos consisten en el proceso de comprender y definir qué servicios se requieren del sistema, así como la identificación de las restricciones sobre la operación y el desarrollo del sistema, por otro lado es importante determinar no solamente las funcionalidades que debe brindar el sistema sino también sus límites. La ingeniería de requerimientos es una etapa particularmente crítica del proceso de software, ya que los errores en esta etapa conducen de manera inevitable a problemas posteriores tanto en el diseño como en la implementación del sistema. Si uno pensara en una metodología en cascada, la primera etapa de esta es la definición de los requerimientos, por lo que básicamente

eso es la especificación de software, es decir, poder determinar para qué va a servir, cuáles son las necesidades que este producto de software va a llevar a resolver.



_ El proceso de ingeniería de requerimientos se enfoca en producir un documento de requerimientos convenido que especifique los requerimientos de los interesados que cumplirá el sistema. Existen cuatro actividades principales en el proceso de ingeniería de requerimientos:

Estudio de factibilidad: se realiza una estimación sobre si las necesidades identificadas del usuario se cubren con las actuales tecnologías de software y hardware. El estudio considera si el sistema propuesto tendrá un costo-beneficio desde un punto de vista empresarial, y si éste puede desarrollarse dentro de las restricciones presupuestales existentes. Un estudio de factibilidad debe ser rápido y relativamente barato. El resultado debe informar la decisión respecto a si se continúa o no continúa con un análisis más detallado. Entonces, para determinar la factibilidad lo que hacemos es una primera aproximación para tener una idea del volumen del sistema con el cual nos tenemos que enfrentar, y hacemos una estimación para ver si es tecnológicamente y financieramente es posible obtener ese producto, y para saber esto hablamos con el cliente.

Obtención y análisis de requerimientos: este es el proceso de derivar los requerimientos del sistema mediante observación de los sistemas existentes, discusiones con los usuarios y proveedores potenciales, análisis de tareas, etcétera. Esto puede incluir el desarrollo de uno o más modelos de sistemas y prototipos, lo que ayuda a entender el sistema que se va a especificar. Cuando hablamos de requerimientos hablamos de los servicios que el sistema debe proveer.

Especificación de requerimientos: consiste en la actividad de transcribir la información recopilada durante la actividad de análisis, en un documento que define un conjunto de requerimientos. En este documento se incluyen dos clases de requerimientos, uno son los requerimientos del usuario que son informes abstractos de requerimientos del

sistema para el cliente y el usuario final del sistema; y el otro son los requerimientos de sistema, que son una descripción detallada de la funcionalidad a ofrecer.

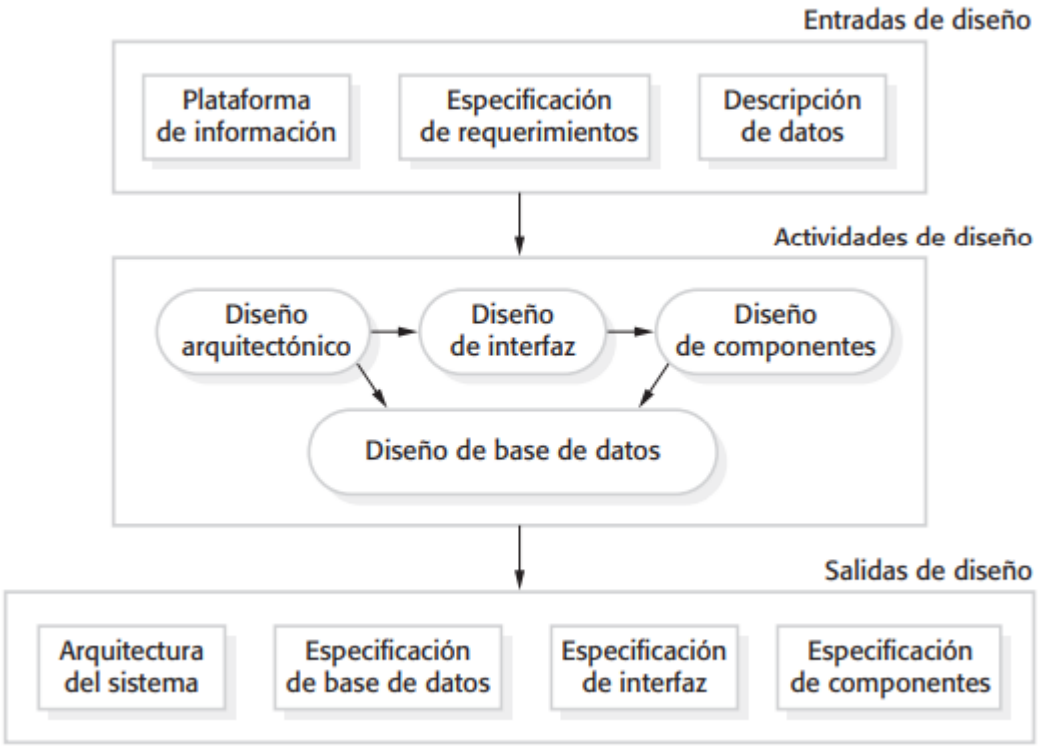
Validación de requerimientos: esta actividad verifica que los requerimientos sean realistas, coherentes y completos. Durante este proceso es inevitable descubrir errores en el documento de requerimientos. En consecuencia, deberían modificarse con la finalidad de corregir dichos problemas. Realizamos la comprobación de la validez de los requisitos.

_ Las actividades en el proceso de requerimientos no se realizan simplemente en una secuencia estricta. El análisis de requerimientos continúa durante la definición y especificación, y a lo largo del proceso salen a la luz nuevos requerimientos; por lo tanto, las actividades de análisis, definición y especificación están vinculadas.

Diseño e implementación del software

Implementación de software: esta etapa corresponde al proceso de convertir una especificación del sistema en un sistema ejecutable. Siempre incluye procesos de diseño y programación de software, aunque también puede involucrar la corrección en la especificación del software, si se utiliza un enfoque incremental de desarrollo.

Diseño de software: se entiende como una descripción de la estructura del software que se va a implementar, los modelos y las estructuras de datos utilizados por el sistema, las interfaces entre componentes del sistema y, en ocasiones, los algoritmos usados. Los diseñadores no llegan inmediatamente a una creación terminada, sino que desarrollan el diseño de manera iterativa.



_ En el grafico vemos las entradas al proceso de diseño, las actividades del proceso y los documentos generados como salidas de este proceso. El diagrama sugiere que las etapas del proceso de diseño son secuenciales. De hecho, las actividades de proceso de diseño están vinculadas. En todos los procesos de diseño es inevitable la retroalimentación de una etapa a otra y la consecuente reelaboración del diseño.

_ Las actividades en el proceso de diseño varían dependiendo del tipo de sistema a desarrollar. Las cuatro actividades que podrían formar parte del proceso de diseño para sistemas de información son:

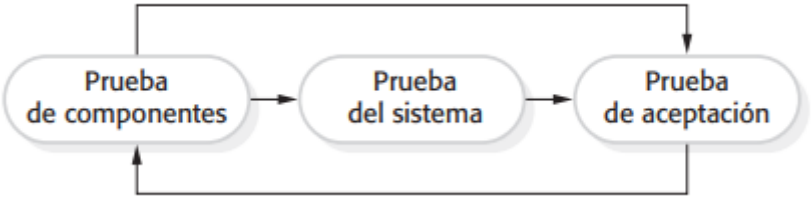
- Diseño arquitectónico: aquí se identifica la estructura global del sistema, los principales componentes (llamados en ocasiones subsistemas o módulos), sus relaciones y cómo se distribuyen.
- Diseño de interfaz: en éste se definen las interfaces entre los componentes de sistemas. Esta especificación de interfaz no tiene que presentar ambigüedades. Con una interfaz precisa, es factible usar un componente sin que otros tengan que saber cómo se implementó. Una vez que se acuerdan las especificaciones de interfaz, los componentes se diseñan y se desarrollan de manera concurrente.
- Diseño de componentes: en él se toma cada componente del sistema y se diseña cómo funcionará. Esto puede ser un simple dato de la funcionalidad que se espera implementar, y al programador se le deja el diseño específico. Como alternativa, habría una lista de cambios a realizar sobre un componente que se reutiliza o sobre un modelo de diseño detallado. El modelo de diseño sirve para generar en automático una implementación.
- Diseño de base de datos: donde se diseñan las estructuras del sistema de datos y cómo se representarán en una base de datos. De nuevo, el trabajo aquí depende de si una base de datos se reutilizará o se creará una nueva.

_ Tales actividades conducen a un conjunto de salidas de diseño.

Validación de software

_ La validación de software o su verificación y validación (V&V), se crea para mostrar que un sistema cumple tanto con sus especificaciones como con las expectativas del cliente. Las pruebas del programa, donde el sistema se ejecuta a través de datos de prueba simulados, son la principal técnica de validación. Esta última también puede incluir procesos de comprobación, como inspecciones y revisiones en cada etapa del proceso de software, desde la definición de requerimientos del usuario hasta el desarrollo del programa.

_ A continuación vemos un proceso de prueba en tres etapas, donde los componentes del sistema se ponen a prueba; luego, se hace lo mismo con el sistema integrado y, finalmente, el sistema se pone a prueba con los datos del cliente.



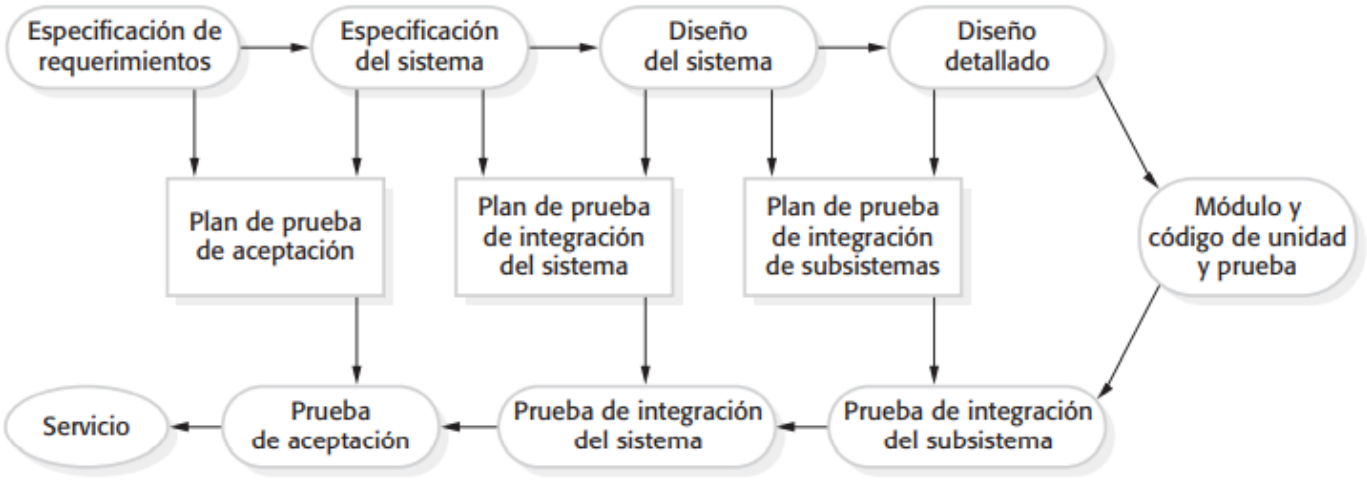
_ Conforme se descubran los defectos, el programa deberá depurarse y esto quizá requiera la repetición de otras etapas en el proceso de pruebas. Los errores en los componentes del programa pueden salir a la luz durante las pruebas del sistema. En consecuencia, el proceso es iterativo, con información retroalimentada desde etapas posteriores hasta las partes iniciales del proceso.

_ Las etapas en el proceso de pruebas son:

Pruebas de componentes o desarrollo: las personas que desarrollan el sistema ponen a prueba los componentes que constituyen el sistema. Cada componente se prueba de manera independiente, es decir, sin otros componentes del sistema. Éstos pueden ser simples entidades, como funciones o clases de objeto, o agrupamientos coherentes de dichas entidades. Esta prueba tiene que tratar de forzar estos componentes para tratar de ver todos los posibles casos en los cuales fallan. Si esta prueba se hace bien podemos pasar a la siguiente.

Pruebas del sistema: los componentes del sistema se integran para crear un sistema completo o un todo. Este proceso tiene la finalidad de descubrir errores que resulten de interacciones no anticipadas entre componentes y problemas de interfaz de componente, así como de mostrar que el sistema cubre sus requerimientos funcionales y no funcionales, y poner a prueba las propiedades emergentes del sistema. Para sistemas grandes, esto puede ser un proceso de múltiples etapas, donde los componentes se conjuntan para formar subsistemas que se ponen a prueba de manera individual, antes de que dichos subsistemas se integren para establecer el sistema final. Cuando se llega a la conclusión de que esto bien, se pasa a las pruebas de aceptación.

Pruebas de aceptación: es la etapa final en el proceso de pruebas, antes de que el sistema se acepte para uso operacional. El sistema se pone a prueba con datos suministrados por el cliente del sistema, en vez de datos de prueba simulados. Las pruebas de aceptación revelan los errores y las omisiones en la definición de requerimientos del sistema, ya que los datos reales ejercitan el sistema en diferentes formas a partir de los datos de prueba. Asimismo, las pruebas de aceptación revelan problemas de requerimientos, donde las instalaciones del sistema en realidad no cumplan las necesidades del usuario o cuando sea inaceptable el rendimiento del sistema. Algunas pruebas se hacen dentro del equipo de desarrollo con la participación del cliente final, otras se hacen en el ambiente de producción del cliente y se ve si el sistema está respondiendo como corresponde.



_ Por lo general, los procesos de desarrollo y de pruebas de componentes están entrelazados. Los programadores construyen sus propios datos de prueba y experimentan el código de manera incremental conforme lo desarrollan. Además de estas pruebas que serían las que prueban funcionalidades, también tenemos que tener en cuenta otro tipo de pruebas que son más funcionales, como por ejemplo:

Pruebas de estrés: sirven para ver si el sistema tiene que trabajar satisfactoriamente con una cierta cantidad de usuarios en concurrencia y fijarse si con mucho más que eso sigue trabajando, tenemos que asegurar que una cierta cantidad media funcione bien.

_ Verificación y validación son aquellas que definen las fases, donde verificamos que el software cumple con los requerimientos y además validamos que sea lo que el cliente quiere. Entonces, dentro de esa fase tenemos todas las pruebas y todas las revisiones. Son dos las actividades de verificación y validación, tenemos:

Dinámicas: que son las pruebas mismas, que se pueden hacer sobre un producto que ya se puede ejecutar

Estáticas: son las revisiones, que se puede hacer sobre cualquier cosa.

_ Por ende decimos que las pruebas y la revisión son dos actividades que están dentro de la fase de verificación y validación. El proceso de prueba puede ser manual o automatizado.

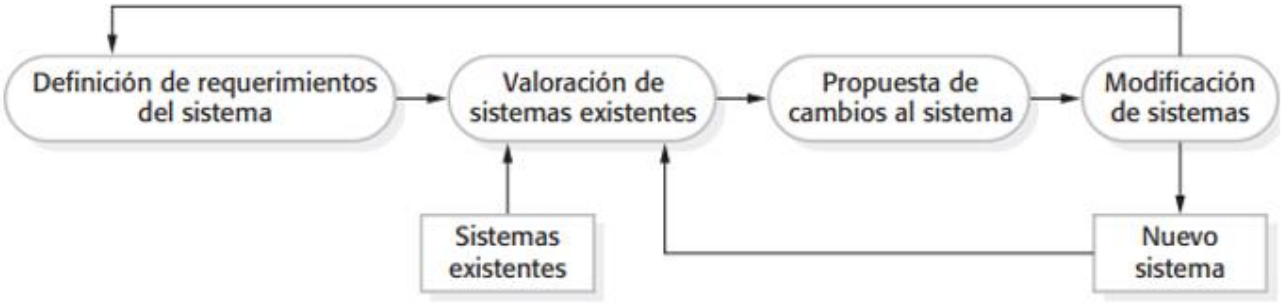
Evolución del software

_ Suponiendo que las pruebas anduvieron bien, el producto fue entregado al cliente y entra en operación, o sea que superamos la etapa de verificación y validación, ahora viene la etapa de la evolución del software, donde necesariamente va a haber una etapa de evolución, y esta esa evolución puede responder a distintas cosas, las más críticas es que aparezcan errores del sistema y del producto cuando ya están en ejecución, por eso en la etapa de evolución y mantenimiento es crítico hacer lo que se llama:

- Mantenimiento correctivo: ir y arreglar todos los errores que nuestro producto presente.
- Mantenimiento adaptativo: adaptamos el software para nuevas situaciones, por ejemplo, por cuestiones tecnológicas o cambios en la organización (diseño responsive, etc).

_ La flexibilidad de los sistemas de software es una de las razones principales por las que cada vez más software se incorpora en los sistemas grandes y complejos. Una vez tomada la decisión de fabricar hardware, resulta muy costoso hacer cambios a su diseño. Sin embargo, en cualquier momento durante o después del desarrollo del sistema, pueden hacerse cambios al software. Incluso los cambios mayores son todavía más baratos que los correspondientes cambios al hardware del sistema.

_ Los procesos de mantenimiento se consideran en ocasiones como menos desafiantes que el desarrollo de software original. Es muy difícil que cualquier sistema de software sea un sistema completamente nuevo, y tiene mucho más sentido ver el desarrollo y el mantenimiento como un continuo. En lugar de dos procesos separados, pensamos en la ingeniería de software como un proceso evolutivo donde el software cambia continuamente a lo largo de su vida, en función de los requerimientos y las necesidades cambiantes del cliente.



Cómo enfrentar el cambio

_ El cambio es inevitable en todos los grandes proyectos de software. Los requerimientos del sistema varían conforme la empresa procura que el sistema responda a presiones externas y se modifican las prioridades administrativas. A medida que se ponen a disposición nuevas tecnologías, surgen nuevas posibilidades de diseño e implementación. Por ende, cualquiera que sea el modelo del proceso de software utilizado, es esencial que ajuste los cambios al software a desarrollar.

_ El cambio se agrega a los costos del desarrollo de software debido a que, por lo general, significa que el trabajo ya terminado debe volver a realizarse. A esto se le llama rehacer. Entonces, es necesario rediseñar el sistema para entregar los nuevos requerimientos, cambiar cualquier programa que se haya desarrollado y volver a probar el sistema.

_ Existen dos enfoques relacionados que se usan para reducir los costos del rehacer:

Evitar el cambio: donde el proceso de software incluye actividades que anticipan cambios posibles antes de requerirse la labor significativa de rehacer. Por ejemplo, puede desarrollarse un sistema prototipo para demostrar a los clientes algunas características clave del sistema. Ellos podrán experimentar con el prototipo y refinar sus requerimientos, antes de comprometerse con mayores costos de producción de software.

Tolerancia al cambio: donde el proceso se diseña de modo que los cambios se ajusten con un costo relativamente bajo. Por lo general, esto comprende algunas formas de desarrollo incremental. Los cambios propuestos pueden implementarse en incrementos que aún no se desarrollan. Si no es posible, entonces tal vez sólo un incremento (una pequeña parte del sistema) tendría que alterarse para incorporar el cambio.

_ Tenemos dos formas de enfrentar el cambio y los requerimientos cambiantes del sistema:

Prototipo de sistema: donde rápidamente se desarrolla una versión del sistema o una parte del mismo, para comprobar los requerimientos del cliente y la factibilidad de algunas decisiones de diseño. Esto apoya el hecho de evitar el cambio, al permitir que los usuarios experimenten con el sistema antes de entregarlo y así refinar sus requerimientos. Como resultado, es probable que se reduzca el número de propuestas de cambio de requerimientos posterior a la entrega.

Entrega incremental: donde los incrementos del sistema se entregan al cliente para su comentario y experimentación. Esto apoya tanto al hecho de evitar el cambio como a tolerar el cambio. Por un lado, evita el compromiso prematuro con los requerimientos para todo el sistema y, por otro, permite la incorporación de cambios en incrementos mayores a costos relativamente bajos. La noción de refactorización, esto es, el mejoramiento de la estructura y organización de un programa, es también un mecanismo importante que apoya la tolerancia al cambio.

Creación del prototipo

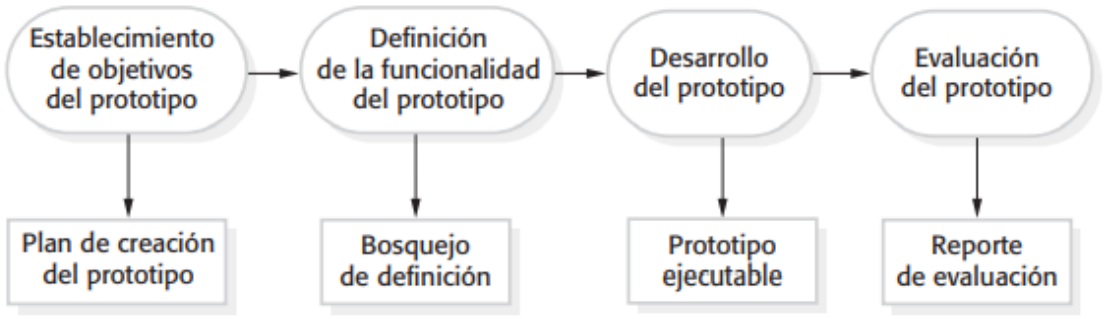
_ Un prototipo es una versión inicial de un sistema de software que se usa para demostrar conceptos, tratar opciones de diseño y encontrar más sobre el problema y sus posibles soluciones. El rápido desarrollo iterativo del prototipo es esencial, de modo que se controlen los costos, y los interesados en el sistema experimenten por anticipado con el prototipo durante el proceso de software. Un prototipo de software se usa en un proceso de desarrollo de software para contribuir a anticipar los cambios que se requieran:

1. En el proceso de ingeniería de requerimientos, un prototipo ayuda con la selección y validación de requerimientos del sistema.

2. En el proceso de diseño de sistemas, un prototipo sirve para buscar soluciones específicas de software y apoyar el diseño de interfaces del usuario.

_ Los prototipos del sistema permiten a los usuarios ver qué tan bien el sistema apoya su trabajo. Pueden obtener nuevas ideas para requerimientos y descubrir áreas de fortalezas y debilidades en el software. Entonces, proponen nuevos requerimientos del sistema. Un prototipo del mismo sirve para comprobar la factibilidad de un diseño propuesto. Cuando hablamos de prototipo para el software, lo que estamos diciendo es que vamos a tener una herramienta que nos va a permitir clarificar algunas cosas, y básicamente este es el objetivo, además de mejorar la comunicación, mostrarle al cliente los aspectos fundamentales y de qué manera va a interactuar con esos aspectos, y en ese caso el prototipo esta como muy orientado a la interfaz de usuario. Este tiene que ser desarrollado en muy poco tiempo, por lo tanto, no vamos a generar funcionalidades ni va a ser una base sobre la que después vamos a desarrollar, ya que estos se tiran, lo podemos hacer en un mockup.

_ A continuación se muestra un modelo del proceso para desarrollo de prototipos:



- Puede ser imposible corregir el prototipo para cubrir requerimientos no funcionales, como los requerimientos de rendimiento, seguridad, robustez y fiabilidad, ignorados durante el desarrollo del prototipo.
- El cambio rápido durante el desarrollo significa claramente que el prototipo no está documentado. La única especificación de diseño es el código del prototipo. Esto no es muy bueno para el mantenimiento a largo plazo.
- Probablemente los cambios realizados durante el desarrollo de prototipos degradarán la estructura del sistema, y este último será difícil y costoso de mantener.
- Por lo general, durante el desarrollo de prototipos se hacen más flexibles los estándares de calidad de la organización.

_ Los prototipos no tienen que ser ejecutables para ser útiles. A continuación, tenemos aspectos a tener en cuenta sobre la prototipado:

- Mejora de la usabilidad del sistema, la calidad del diseño, la capacidad de mantenimiento y reducen el esfuerzo de desarrollo
- Son una aproximación más exacta a las necesidades reales de los usuarios.
- Estos deben centrarse en las áreas del producto que no se conocen bien.
- La comprobación de errores y recuperación pueden no estar incluidos en el prototipo.
- Centrarse en los requisitos funcionales y no en los no funcionales tales como la fiabilidad y la seguridad.
- Son normalmente indocumentados.

Entrega incremental

_ La entrega incremental es un enfoque al desarrollo de software donde algunos de los incrementos diseñados se entregan al cliente y se implementan para usarse en un entorno operacional. En un proceso de entrega incremental, los clientes identifican, en un bosquejo, los servicios que proporciona el sistema, identifican cuáles servicios son más importantes y cuáles son menos significativos para ellos. Entonces, se define un número de incrementos de entrega, y cada incremento proporciona un subconjunto de la funcionalidad del sistema.

_ Una vez identificados los incrementos del sistema, se definen con detalle los requerimientos de los servicios que se van a entregar en el primer incremento, y se desarrolla ese incremento. Una vez completado y entregado el incremento, los clientes lo ponen en servicio. A medida que se completan nuevos incrementos, se integran con los incrementos existentes, de modo que con cada incremento entregado mejore la funcionalidad del sistema.

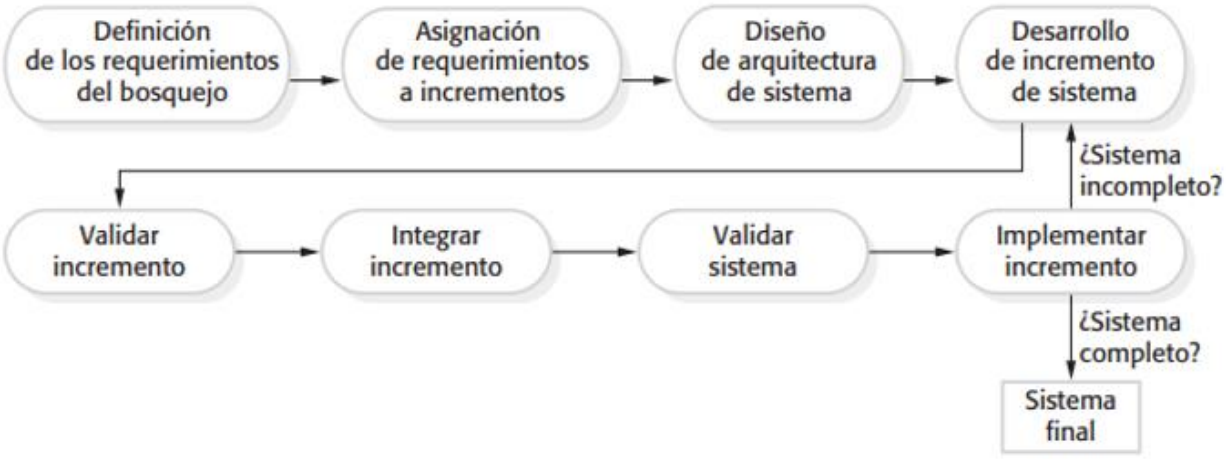
_ La entrega incremental tiene algunas ventajas:

- Los clientes pueden usar los primeros incrementos como prototipos y adquirir experiencia que informe sobre sus requerimientos, para posteriores incrementos del sistema. A diferencia de los prototipos, éstos son parte del sistema real, de manera que no hay reaprendizaje cuando está disponible el sistema completo.
- Los clientes deben esperar hasta la entrega completa del sistema, antes de ganar valor del mismo. El primer incremento cubre sus requerimientos más críticos, de modo que es posible usar inmediatamente el software.
- El proceso mantiene los beneficios del desarrollo incremental en cuanto a que debe ser relativamente sencillo incorporar cambios al sistema.
- Puesto que primero se entregan los servicios de mayor prioridad y luego se integran los incrementos, los servicios de sistema más importantes reciben mayores pruebas.

_ Esto significa que los clientes tienen menos probabilidad de encontrar fallas de software en las partes más significativas del sistema.

_ La entrega incremental tiene algunas desventajas:

- La mayoría de los sistemas requieren de una serie de recursos que se utilizan para diferentes partes del sistema. Dado que los requerimientos no están definidos con detalle sino hasta que se implementa un incremento, resulta difícil identificar recursos comunes que necesiten todos los incrementos.
- Asimismo, el desarrollo iterativo resulta complicado cuando se diseña un sistema de reemplazo. Los usuarios requieren de toda la funcionalidad del sistema antiguo, ya que es común que no deseen experimentar con un nuevo sistema incompleto. Por lo tanto, es difícil conseguir retroalimentación útil del cliente.
- La esencia de los procesos iterativos es que la especificación se desarrolla en conjunto con el software. Sin embargo, esto se puede contradecir con el modelo de adquisiciones de muchas organizaciones, donde la especificación completa del sistema es parte del contrato de desarrollo del sistema. En el enfoque incremental, no hay especificación completa del sistema, sino hasta que se define el incremento final. Esto requiere una nueva forma de contrato que los grandes clientes, como las agencias gubernamentales, encontrarían difícil de adoptar.



Puntos clave

Procesos de software: son actividades implicadas en la producción de un sistema de software. Los modelos de proceso de software consisten en representaciones abstractas de dichos procesos.

Modelos de proceso general: describen la organización de los procesos de software. Los ejemplos de estos modelos generales incluyen el modelo en cascada, el desarrollo incremental y el desarrollo orientado a la reutilización.

Ingeniería de requerimientos: es el proceso de desarrollo de una especificación de software. Las especificaciones tienen la intención de comunicar las necesidades de sistema del cliente a los desarrolladores del sistema.

Procesos de diseño e implementación: tratan de transformar una especificación de requerimientos en un sistema de software ejecutable. Pueden usarse métodos de diseño sistemáticos como parte de esta transformación.

Validación del software: es el proceso de comprobar que el sistema se conforma a su especificación y que satisface las necesidades reales de los usuarios del sistema.

Evolución del software: tiene lugar cuando cambian los sistemas de software existentes para satisfacer nuevos requerimientos. Los cambios son continuos y el software debe evolucionar para seguir siendo útil.

Procesos: deben incluir actividades para lidiar con el cambio. Esto puede implicar una fase de creación de prototipos que ayude a evitar malas decisiones sobre los requerimientos y el diseño. Los procesos pueden estructurarse para desarrollo y entrega iterativos, de forma que los cambios se realicen sin perturbar al sistema como un todo.

Proceso Unificado Racional: es un modelo de proceso genérico moderno que está organizado en fases (concepción, elaboración, construcción y transición), pero separa las actividades (requerimientos, análisis y diseño, etcétera) de dichas fases. El enfoque

práctico del RUP describe las buenas prácticas de ingeniería de software que se recomiendan para su uso en el desarrollo de sistemas. Las seis mejores prácticas fundamentales que se recomiendan son:

- Desarrollo de software de manera iterativa: incrementar el plan del sistema con base en las prioridades del cliente, y desarrollar oportunamente las características del sistema de mayor prioridad en el proceso de desarrollo.
- Gestión de requerimientos: documentar de manera explícita los requerimientos del cliente y seguir la huella de los cambios a dichos requerimientos. Analizar el efecto de los cambios sobre el sistema antes de aceptarlos.
- Usar arquitecturas basadas en componentes: estructurar la arquitectura del sistema en componentes, como se estudió anteriormente en este capítulo.
- Software modelado visualmente: usar modelos UML gráficos para elaborar representaciones de software estáticas y dinámicas.
- Verificar la calidad del software: garantizar que el software cumpla con los estándares de calidad de la organización.
- Controlar los cambios al software: gestionar los cambios al software con un sistema de administración del cambio, así como con procedimientos y herramientas de administración de la configuración.

Desarrollo ágil de software

Introducción

_ Los procesos de desarrollo del software rápido se diseñan para producir rápidamente un software útil. El software no se desarrolla como una sola unidad, sino como una serie de incrementos, y cada uno de ellos incluye una nueva funcionalidad del sistema.

_ Características fundamentales del desarrollo rápido de software:

1)_ Los procesos de especificación, diseño e implementación están entrelazados. No existe una especificación detallada del sistema, y la documentación del diseño se minimiza o es generada automáticamente por el entorno de programación que se usa para implementar el sistema. El documento de requerimientos del usuario define sólo las características más importantes del sistema.

2)_ El sistema se desarrolla en diferentes versiones. Los usuarios finales y otros colaboradores del sistema intervienen en la especificación y evaluación de cada versión. Ellos podrían proponer cambios al software y nuevos requerimientos que se implementen en una versión posterior del sistema.

3)_ Las interfaces de usuario del sistema se desarrollan usando con frecuencia un sistema de elaboración interactivo, que permita que el diseño de la interfaz se cree rápidamente en cuanto se dibujan y colocan iconos en la interfaz. En tal situación, el sistema puede generar una interfaz basada en la Web para un navegador o una interfaz para una plataforma específica, como Microsoft Windows.

_ Los métodos ágiles son métodos de desarrollo incremental donde los incrementos son mínimos y, por lo general, se crean las nuevas liberaciones del sistema, y cada dos o tres semanas se ponen a disposición de los clientes. Involucran a los clientes en el proceso de desarrollo para conseguir una rápida retroalimentación sobre los requerimientos cambiantes. Minimizan la cantidad de documentación con el uso de comunicaciones informales, en vez de reuniones formales con documentos escritos.

Métodos ágiles

_ Los enfoques basados en un plan incluyen costos operativos significativos en la planeación, el diseño y la documentación del sistema. Dichos gastos se justifican cuando debe coordinarse el trabajo de múltiples equipos de desarrollo, cuando el sistema es un sistema crítico y cuando numerosas personas intervendrán en el mantenimiento del software a lo largo de su vida. Sin embargo, cuando este engorroso enfoque de desarrollo basado en la planeación se aplica a sistemas de negocios pequeños y medianos, los costos que se incluyen son tan grandes que dominan el proceso de desarrollo del software. Se invierte más tiempo en diseñar el sistema, que en el desarrollo y la prueba del programa. Conforme cambian los requerimientos del sistema, resulta esencial la reelaboración y, en principio al menos, la especificación y el diseño deben modificarse con el programa.

_ En la década de 1990 se proponen nuevos “métodos ágiles”, los cuales permitieron que el equipo de desarrollo se enfocara en el software en lugar del diseño y la documentación.

_ Los métodos ágiles se apoyan universalmente en el enfoque incremental para la especificación, el desarrollo y la entrega del software. Son más adecuados para el diseño de aplicaciones en que los requerimientos del sistema cambian, por lo general, rápidamente durante el proceso de desarrollo. Tienen la intención de entregar con prontitud el software operativo a los clientes, quienes entonces propondrán requerimientos nuevos y variados para incluir en posteriores iteraciones del sistema.

_ La filosofía detrás de los métodos ágiles se refleja en el manifiesto ágil, que acordaron muchos de los desarrolladores líderes de estos métodos. Este manifiesto afirma: *“Estamos descubriendo mejores formas para desarrollar software, al hacerlo y al ayudar a otros a hacerlo. Gracias a este trabajo llegamos a valorar: A los individuos y las interacciones sobre los procesos y las herramientas Al software operativo sobre la documentación exhaustiva La colaboración con el cliente sobre la negociación del contrato La respuesta al cambio sobre el seguimiento de un plan Esto es, aunque exista valor en los objetos a la derecha, valoraremos más los de la izquierda”.*

_ Los métodos ágiles han tenido mucho éxito para ciertos tipos de desarrollo de sistemas:

- Desarrollo del producto, donde una compañía de software elabora un producto pequeño o mediano para su venta.

- Diseño de sistemas a la medida dentro de una organización, donde hay un claro compromiso del cliente por intervenir en el proceso de desarrollo, y donde no existen muchas reglas ni regulaciones externas que afecten el software.

Los métodos ágiles y el mantenimiento de Software

_ Una cuestión es cuando estamos desarrollando y logramos completar el producto, pero todo producto entra después en una etapa de mantenimiento y de evolución. Se pueden usar las metodologías agiles en el proceso de mantenimiento al igual que el proceso de desarrollo. El problema puede estar cuando se haya hecho el desarrollo original del producto, utilizando una metodología ágil, por lo cual la documentación existe, pero si después una empresa de software hace el desarrollo inicial y después resulta que le mantenimiento lo tiene que hacer otra organización, y acá puede haber problemas en el proceso de mantenimiento porque no tenemos una documentación tan formal. Entonces el principal problema es si el equipo de trabajo no puede mantenerse.

_ La mayoría de las organizaciones gastan más en el mantenimiento del software existente que en el desarrollo de nuevo software. Los métodos ágiles deben facilitar tanto el mantenimiento como el desarrollo inicial.

_ Tenemos dos problemas fundamentales:

- ¿Son los sistemas que se desarrollan utilizando un enfoque ágil mantenibles, haciendo énfasis en la minimización de la documentación formal?
- ¿Se pueden utilizar los métodos ágiles con eficacia para la evolución de un sistema?

_ Pueden surgir problemas si el equipo de desarrollo original no puede mantenerse.

Principios del desarrollo ágil

Participación del cliente: los clientes deben intervenir estrechamente durante el proceso de desarrollo. Su función consiste en ofrecer y priorizar nuevos requerimientos del sistema y evaluar las iteraciones del mismo.

Entrega incremental: el software se desarrolla en incrementos y el cliente especifica los requerimientos que se van a incluir en cada incremento.

Personas, no procesos: tienen que reconocerse y aprovecharse las habilidades del equipo de desarrollo. Debe permitirse a los miembros del equipo desarrollar sus propias formas de trabajar sin procesos establecidos.

Adoptar el cambio: esperar a que cambien los requerimientos del sistema y, de este modo, diseñar el sistema para adaptar dichos cambios.

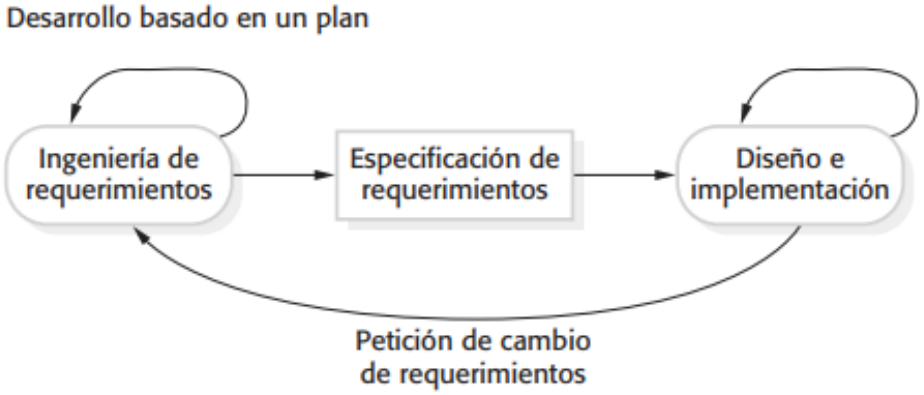
Mantener simplicidad: enfocarse en la simplicidad tanto en el software a desarrollar como en el proceso de desarrollo. Siempre que sea posible, trabajar de manera activa para eliminar la complejidad del sistema.

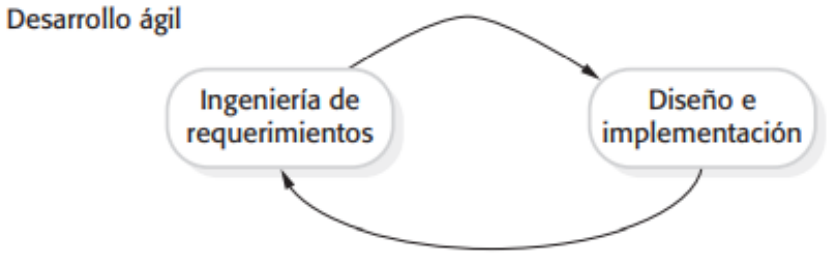
_ Problemas con los métodos agiles:

- Aunque es atractiva la idea del involucramiento del cliente en el proceso de desarrollo, su éxito radica en tener un cliente que desee y pueda pasar tiempo con el equipo de desarrollo, y éste represente a todos los participantes del sistema. Los representantes del cliente están comúnmente sujetos a otras presiones, así que no intervienen por completo en el desarrollo del software.
- Quizás algunos miembros del equipo no cuenten con la personalidad adecuada para la participación intensa característica de los métodos ágiles y, en consecuencia, no podrán interactuar adecuadamente con los otros integrantes del equipo.
- Priorizar los cambios sería extremadamente difícil, sobre todo en sistemas donde existen muchos participantes. Cada uno por lo general ofrece diversas prioridades a diferentes cambios.
- Mantener la simplicidad requiere trabajo adicional. Bajo la presión de fechas de entrega, es posible que los miembros del equipo carezcan de tiempo para realizar las simplificaciones deseables al sistema.
- Muchas organizaciones, especialmente las grandes compañías, pasan años cambiando su cultura, de tal modo que los procesos se definan y continúen. Para ellas, resulta difícil moverse hacia un modelo de trabajo donde los procesos sean informales y estén definidos por equipos de desarrollo.

Desarrollo dirigido por un plan y desarrollo ágil

_ Los enfoques ágiles en el desarrollo de software consideran el diseño y la implementación como las actividades centrales en el proceso del software. Incorporan otras actividades en el diseño y la implementación, como la adquisición de requerimientos y pruebas. En contraste, un enfoque basado en un plan para la ingeniería de software identifica etapas separadas en el proceso de software con salidas asociadas a cada etapa. Las salidas de una etapa se usan como base para planear la siguiente actividad del proceso.





_ En el grafico vemos las distinciones entre los enfoques ágil y el basado en un plan para la especificación de sistemas.

Enfoque basado en un plan: la iteración ocurre dentro de las actividades con documentos formales usados para comunicarse entre etapas del proceso. Esto entonces es una entrada al proceso de diseño y la implementación. Un proceso de software dirigido por un plan soporta el desarrollo y la entrega incrementales. Es perfectamente factible asignar requerimientos y planear tanto la fase de diseño y desarrollo como una serie de incrementos. Identifica etapas separadas en el proceso de software con salidas asociadas a cada etapa. Las salidas de una etapa se usan como base para planear la siguiente actividad del proceso.

Enfoque desarrollo ágil: la iteración ocurre a través de las actividades. Por lo tanto, los requerimientos y el diseño se desarrollan en conjunto, no por separado, es decir, la especificación, diseño, implementación y pruebas están intercalados.

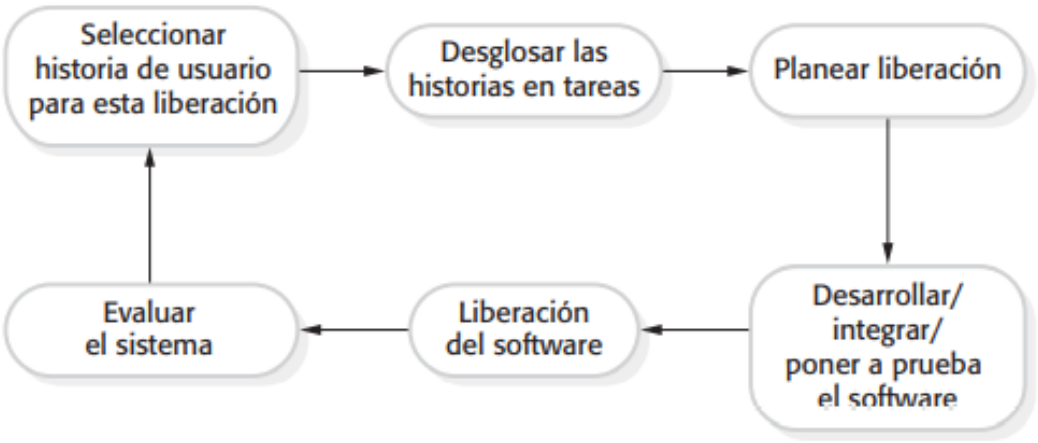
_ Un proceso ágil no está inevitablemente enfocado al código y puede producir cierta documentación de diseño. El equipo de desarrollo ágil puede incluir un “pico” de documentación donde, en vez de producir una nueva versión de un sistema, el equipo generará documentación del sistema. De hecho, la mayoría de los proyectos de software incluyen prácticas de los enfoques ágil y basado en un plan.

_ En realidad, es irrelevante el conflicto sobre si un proyecto puede considerarse dirigido por un plan o ágil. A final de cuentas, la principal inquietud de los compradores de un sistema de software es si cuentan o no con un sistema de software ejecutable, que cubra sus necesidades y realice funciones útiles para el usuario de manera individual o dentro de una organización. En la práctica, muchas compañías que afirman haber usado métodos ágiles adoptaron algunas habilidades ágiles y las integraron con sus procesos dirigidos por un plan. Fundamentalmente la selección del modelo de desarrollo depende del proyecto, además puede depender de la estructura del equipo de desarrollo.

Programación extrema

_ La programación extrema (XP) es quizás el método ágil mejor conocido y más ampliamente usado. Por ejemplo, en la XP muchas versiones actuales de un sistema pueden desarrollarse mediante diferentes programadores, integrarse y ponerse a prueba en un solo día.

_ En la programación extrema, los requerimientos se expresan como escenarios (llamados historias de usuario), que se implementan directamente como una serie de tareas. Los programadores trabajan en pares y antes de escribir el código desarrollan pruebas para cada tarea. Todas las pruebas deben ejecutarse con éxito una vez que el nuevo código se integre en el sistema. Entre las liberaciones del sistema existe un breve lapso. A continuación, se ilustra el proceso XP para producir un incremento del sistema por desarrollar:



Planeación incremental: los requerimientos se registran en tarjetas de historia (story cards) y las historias que se van a incluir en una liberación se determinan por el tiempo disponible y la prioridad relativa. Los desarrolladores desglosan dichas historias en "tareas" de desarrollo.

Liberaciones pequeñas: al principio se desarrolla el conjunto mínimo de funcionalidad útil, que ofrece valor para el negocio. Las liberaciones del sistema son frecuentes y agregan incrementalmente funcionalidad a la primera liberación.

Diseño simple: se realiza un diseño suficiente para cubrir sólo aquellos requerimientos actuales.

Desarrollo de la primera prueba: se usa un marco de referencia de prueba de unidad automatizada al escribir las pruebas para una nueva pieza de funcionalidad, antes de que esta última se implemente.

Refactorización: se espera que todos los desarrolladores refactoricen de manera continua el código y, tan pronto como sea posible, se encuentren mejoras de éste. Lo anterior conserva el código simple y mantenible.

Programación en pares: los desarrolladores trabajan en pares, y cada uno comprueba el trabajo del otro; además, ofrecen apoyo para que se realice siempre un buen trabajo.

Propiedad colectiva: los desarrolladores en pares trabajan en todas las áreas del sistema, de manera que no se desarrollan islas de experiencia, ya que todos los desarrolladores se responsabilizan por todo el código. Cualquiera puede cambiar cualquier función.

Integración continua: tan pronto como esté completa una tarea, se integra en todo el sistema. Después de tal integración, deben aprobarse todas las pruebas de unidad en el sistema.

Ritmo sustentable: grandes cantidades de tiempo extra no se consideran aceptables, pues el efecto neto de este tiempo libre con frecuencia es reducir la calidad del código y la productividad de término medio.

Cliente en sitio: un representante del usuario final del sistema (el cliente) tiene que disponer de tiempo completo para formar parte del equipo XP. En un proceso de programación extrema, el cliente es miembro del equipo de desarrollo y responsable de llevar los requerimientos del sistema al grupo para su implementación.

_ La programación extrema incluye algunas prácticas, resumidas a continuación, las cuales reflejan los principios de los métodos ágiles:

1)_ El desarrollo incremental se apoya en pequeñas y frecuentes liberaciones del sistema. Los requerimientos se fundamentan en simples historias del cliente, o bien, en escenarios usados como base para decidir qué funcionalidad debe incluirse en un incremento del sistema.

2)_ La inclusión del cliente se apoya a través de un enlace continuo con el cliente en el equipo de desarrollo. El representante del cliente participa en el desarrollo y es responsable de definir las pruebas de aceptación para el sistema.

3)_ Las personas, no los procesos, se basan en la programación en pares, en la propiedad colectiva del código del sistema y en un proceso de desarrollo sustentable que no incluya jornadas de trabajo excesivamente largas.

4)_ El cambio se acepta mediante liberaciones regulares del sistema a los clientes, desarrollo de primera prueba, refactorización para evitar degeneración del código e integración continua de nueva funcionalidad.

5)_ Mantener la simplicidad se logra mediante la refactorización constante, que mejora la calidad del código, y con el uso de diseños simples que no anticipan innecesariamente futuros cambios al sistema.

Refactorización

_ El hecho de trabajar en pares, hace que la refactorización sea continua, es decir, no importa solo si funciona bien, sino que, si el código no está limpio o no es correcto, se mejora, porque el código simple es mantenible y es más fácil después agregar los cambios. Entonces uno de los aspectos que son muy importantes es la refactorización.

_ El equipo de programación busca posibles mejoras de código y realiza las mejoras aunque no sean de inmediata necesidad. Esto mejora la comprensibilidad del software y reduce la necesidad de documentación. Los cambios son más fáciles de hacer ya que el código es bien estructurado y claro. Sin embargo, algunos cambios requieren reconstrucción de la arquitectura y esto es mucho más caro. Algunos ejemplos de refactorización son:

- Re-organización de una jerarquía de clases para eliminar código duplicado.
- Ordenar y cambiar el nombre de los atributos y métodos para facilitar su comprensión.
- La sustitución de líneas de código con llamadas a métodos que se han incluido en una biblioteca de programas.

Pruebas en XP

_ Una de las diferencias importantes entre desarrollo incremental y desarrollo dirigido por un plan está en la forma en que el sistema se pone a prueba. Con el desarrollo incremental, no hay especificación de sistema que pueda usar un equipo de prueba externo para desarrollar pruebas del sistema. En consecuencia, algunos enfoques del desarrollo incremental tienen un proceso de pruebas muy informal, comparado con las pruebas dirigidas por un plan. Para evitar varios de los problemas de prueba y validación del sistema, XP enfatiza la importancia de la prueba de programa. La XP incluye un enfoque para probar que reduce las posibilidades de introducir errores no detectados en la versión actual del sistema.

_ Las características clave de poner a prueba en XP son:

- Desarrollo de primera prueba
- Desarrollo de pruebas incrementales a partir de escenarios.
- Involucramiento del usuario en el desarrollo y la validación de pruebas.
- El uso de marcos de pruebas automatizadas.

_ El desarrollo de la primera prueba es una de las innovaciones más importantes en XP. En lugar de escribir algún código y luego las pruebas para dicho código, las pruebas se elaboran antes de escribir el código. Esto significa que la prueba puede correrse conforme se escribe el código y descubrir problemas durante el desarrollo. Escribir pruebas implícitamente define tanto una interfaz como una especificación del comportamiento para la funcionalidad a desarrollar. Se reducen los problemas de la mala interpretación de los requerimientos y la interfaz.

La automatización de pruebas

_ La automatización de pruebas significa que las pruebas se escriben como componentes ejecutables antes de que la tarea se implemente. Estos componentes de prueba deben ser independientes, deberían simular la presentación de la entrada para ser probado y debe comprobar que el resultado cumple con las especificaciones de salida. Un marco de prueba automatizada (por ejemplo Junit) es un sistema que hace que sea fácil de escribir pruebas ejecutables y presentar un conjunto de pruebas para su ejecución. Cuando se automatizan las pruebas, siempre hay un conjunto de pruebas que puede ser rápida y fácilmente ejecutado.

- Siempre que se agrega alguna funcionalidad al sistema, las pruebas se pueden corren y los problemas que ha introducido el nuevo código puede ser atrapadas inmediatamente

Programación en pares

_ Otra práctica innovadora que se introdujo en XP es que los programadores trabajan en pares para desarrollar el software. En realidad, trabajan juntos en la misma estación de trabajo para desarrollar el software. Sin embargo, los mismos pares no siempre programan juntos. En vez de ello, los pares se crean dinámicamente, de manera que todos los miembros del equipo trabajen entre sí durante el proceso de desarrollo.

_ El uso de la programación en pares tiene algunas ventajas:

1)_ Apoya la idea de la propiedad y responsabilidad colectivas para el sistema, en donde el software es propiedad del equipo como un todo (con responsabilidad colectiva para resolver dichos problemas) y los individuos no son responsables por los problemas con el código.

2)_ Actúa como un proceso de revisión informal, porque al menos dos personas observan cada línea de código. Las inspecciones y revisiones de código son muy eficientes para detectar un alto porcentaje de errores de software. Sin embargo, consumen tiempo en su organización y, usualmente, presentan demoras en el proceso de desarrollo.

3)_ Ayuda a la refactorización, que es un proceso de mejoramiento del software. La dificultad de implementarlo en un entorno de desarrollo normal es que el esfuerzo en la refactorización se utiliza para beneficio a largo plazo. Un individuo que practica la refactorización podría calificarse como menos eficiente que uno que simplemente realiza desarrollo del código. Donde se usan la programación en pares y la propiedad colectiva, otros se benefician inmediatamente de la refactorización, de modo que es probable que apoyen el proceso.

Administración de un proyecto ágil

_ La responsabilidad principal de los administradores del proyecto de software es dirigir el proyecto, de modo que el software se entregue a tiempo y con el presupuesto planeado para ello. Supervisan el trabajo de los ingenieros de software y monitorizan el avance en el desarrollo del software. El enfoque estándar de la administración de proyectos es el basado en un plan.

_ Un enfoque basado en un plan requiere en realidad que un administrador tenga una visión equilibrada de todo lo que debe diseñarse y de los procesos de desarrollo. Sin embargo, no funciona bien con los métodos ágiles, donde los requerimientos se desarrollan incrementalmente, donde el software se entrega en rápidos incrementos cortos, y donde los cambios a los requerimientos y el software son la norma.

_ Como cualquier otro proceso de diseño de software profesional, el desarrollo ágil tiene que administrarse de tal modo que se busque el mejor uso del tiempo y de los recursos disponibles para el equipo. Esto requiere un enfoque diferente a la administración del proyecto, que se adapte al desarrollo incremental y a las fortalezas particulares de los métodos ágiles.

Enfoque de Scrum

_ Scrum es un método ágil general, su enfoque está en la administración iterativa del desarrollo, y no en enfoques técnicos específicos para la ingeniería de software ágil. La siguiente figura representa un diagrama del proceso de administración de Scrum:

_ Este proceso no prescribe el uso de prácticas de programación, como la programación en pares y el desarrollo de primera prueba. Por lo tanto, puede usarse con enfoques ágiles más técnicos, como XP, para ofrecer al proyecto un marco administrativo. Existen tres fases con Scrum:

- La primera es la planeación del bosquejo, donde se establecen los objetivos generales del proyecto y el diseño de la arquitectura de software.
- La segunda es una serie de ciclos sprint, donde cada ciclo desarrolla un incremento del sistema.
- La tercera es la fase de cierre del proyecto concluye el mismo, completa la documentación requerida, como los marcos de ayuda del sistema y los manuales del usuario, y valora las lecciones aprendidas en el proyecto.

Pilares de scrum: estos son transparencia, inspección y adaptación.

Sprint de Scrum: es una unidad de planeación en la que se valora el trabajo que se va a realizar, se seleccionan las particularidades por desarrollar y se implementa el software. Al final de un sprint, la funcionalidad completa se entrega a los participantes. Las características clave de este proceso son las siguientes:

1)_ Los sprints tienen longitud fija, por lo general de dos a cuatro semanas. Corresponden al desarrollo de una liberación del sistema en XP.

2)_ El punto de partida para la planeación es la cartera del producto, que es la lista de trabajo por realizar en el proyecto. Durante la fase de valoración del sprint, esto se revisa, y se asignan prioridades y riesgos. El cliente interviene estrechamente en este proceso y al comienzo de cada sprint puede introducir nuevos requerimientos o tareas.

3)_ La fase de selección incluye a todo el equipo del proyecto que trabaja con el cliente, con la finalidad de seleccionar las características y la funcionalidad a desarrollar durante el sprint.

4)_ Una vez acordado, el equipo se organiza para desarrollar el software. Con el objetivo de revisar el progreso y, si es necesario, volver a asignar prioridades al trabajo, se realizan reuniones diarias breves con todos los miembros del equipo. Durante esta etapa, el equipo se aísla del cliente y la organización, y todas las comunicaciones se canalizan a través del llamado “maestro de Scrum”. El papel de este último es proteger al equipo de desarrollo de distracciones externas. La forma en que el trabajo se realiza depende del problema y del equipo. A diferencia de XP, Scrum no hace sugerencias específicas sobre cómo escribir requerimientos, desarrollar la primera prueba, etcétera. Sin embargo, dichas prácticas XP se usan cuando el equipo las considera adecuadas.

5)_ Al final del sprint, el trabajo hecho se revisa y se presenta a los participantes. Luego comienza el siguiente ciclo de sprint.

_ Cuando se acaba el Sprint tiene que haber una integración donde va a haber un resultado que se entrega al cliente, y después se hace una retrospectiva para ver cuáles fueron los resultados, que se aprendió de las cosas bien o mal hechas.

_ La idea detrás de Scrum es que debe autorizarse a todo el equipo para tomar decisiones, de modo que se evita deliberadamente el término “administrador del proyecto”. En lugar de ello tenemos

Scrum master: que es el facilitador que ordena las reuniones diarias, rastrea el atraso del trabajo a realizar, registra las decisiones, mide el progreso del atraso, y se comunica con los clientes y administradores fuera del equipo.

Equipo de trabajo: todo el equipo asiste a las reuniones diarias, que en ocasiones son reuniones en las que los participantes no se sientan, para hacerlas breves y enfocadas. Durante la reunión, todos los miembros del equipo comparten información, describen sus avances desde la última reunión, los problemas que han surgido y los planes del día siguiente. Ello significa que todos en el equipo conocen lo que acontece y, si surgen problemas, replantean el trabajo en el corto plazo para enfrentarlo. Todos participan en esta planeación; no hay dirección descendente desde el maestro de Scrum.

_ Beneficios de scrum:

- El producto se divide en un conjunto de fragmentos manejables y comprensibles.
- Requisitos inestables no sostienen el progreso.
- Todo el equipo tiene visibilidad de todo y por lo tanto se mejora la comunicación del equipo.
- Los clientes ven la entrega a tiempo y se obtiene retroalimentación sobre cómo funciona el producto.
- Se genera confianza entre los clientes y los desarrolladores . Se crea una cultura positiva en la que todos esperan que el proyecto tenga éxito.

Escalamiento de métodos ágiles

_ Los métodos ágiles se desarrollaron para usarse en pequeños equipos de programación, que podían trabajar juntos en la misma habitación y comunicarse de manera informal. Por lo tanto, los métodos ágiles se emplean principalmente para el diseño de sistemas pequeños y medianos. Desde luego, la necesidad de entrega más rápida del software, que es más adecuada para las necesidades del cliente, se aplica también a sistemas más grandes. Por consiguiente, hay un enorme interés en escalar los métodos ágiles para enfrentar los sistemas de mayor dimensión, desarrollados por grandes organizaciones.

_ El desarrollo de grandes sistemas de software difiere en algunas formas del desarrollo de sistemas pequeños:

1)_ Los grandes sistemas son, por lo general, colecciones de sistemas separados en comunicación, donde equipos separados desarrollan cada sistema. Dichos equipos trabajan con frecuencia en diferentes lugares, en ocasiones en otras zonas horarias. Es prácticamente imposible que cada equipo tenga una visión de todo el sistema. En consecuencia, sus prioridades son generalmente completar la parte del sistema sin considerar asuntos de los sistemas más amplios.

2)_ Los grandes sistemas son “sistemas abandonados” (Hopkins y Jenkins, 2008); esto es, incluyen e interactúan con algunos sistemas existentes. Muchos de los requerimientos del sistema se interesan por su interacción y, por lo tanto, en realidad no se prestan a la flexibilidad y al desarrollo incremental. Aquí también podrían ser relevantes los conflictos políticos y a menudo la solución más sencilla a un problema es cambiar un sistema existente. Sin embargo, esto requiere negociar con los administradores de dicho sistema para convencerlos de que los cambios pueden implementarse sin riesgo para la operación del sistema.

3)_ Donde muchos sistemas se integran para crear un solo sistema, una fracción significativa del desarrollo se ocupa en la configuración del sistema, y no en el desarrollo del código original. Esto no necesariamente es compatible con el desarrollo incremental y la integración frecuente del sistema.

4)_ Los grandes sistemas y sus procesos de desarrollo por lo común están restringidos por reglas y regulaciones externas, que limitan la forma en que pueden desarrollarse, lo cual requiere de ciertos tipos de documentación del sistema que se va a producir, etcétera.

5)_ Los grandes sistemas tienen un tiempo prolongado de adquisición y desarrollo. Es difícil mantener equipos coherentes que conozcan el sistema durante dicho periodo, pues resulta inevitable que algunas personas se cambien a otros empleos y proyectos.

6)_ Los grandes sistemas tienen por lo general un conjunto variado de participantes. Por ejemplo, cuando enfermeras y administradores son los usuarios finales de un sistema médico, el personal médico ejecutivo, los administradores del hospital,

etcétera, también son participantes en el sistema. En realidad, es imposible involucrar a todos estos participantes en el proceso de desarrollo.

_ Existen dos perspectivas en el escalamiento de los métodos ágiles:

Perspectiva de “expansión” (Scaling Up): que se interesa por el uso de dichos métodos para el desarrollo de grandes sistemas de software que no logran desarrollarse con equipos pequeños.

Perspectiva de “ampliación” (Scaling Out): que se interesa por que los métodos ágiles se introduzcan en una organización grande con muchos años de experiencia en el desarrollo de software.

_ Es difícil introducir los métodos ágiles en las grandes compañías por algunas razones:

1)_ Los gerentes del proyecto carecen de experiencia con los métodos ágiles; pueden ser reticentes para aceptar el riesgo de un nuevo enfoque, pues no saben cómo afectará sus proyectos particulares.

2)_ Las grandes organizaciones tienen a menudo procedimientos y estándares de calidad que se espera sigan todos los proyectos y, dada su naturaleza burocrática, es probable que sean incompatibles con los métodos ágiles. En ocasiones, reciben apoyo de herramientas de software (por ejemplo, herramientas de gestión de requerimientos), y el uso de dichas herramientas es obligatorio para todos los proyectos.

3. Los métodos ágiles parecen funcionar mejor cuando los miembros del equipo tienen un nivel de habilidad relativamente elevado. Sin embargo, dentro de grandes organizaciones, probablemente haya una amplia gama de habilidades y destrezas, y los individuos con niveles de habilidad inferiores quizá no sean miembros de equipos efectivos en los procesos ágiles.

4)_ Quizás haya resistencia cultural contra los métodos ágiles, en especial en aquellas organizaciones con una larga historia de uso de procesos convencionales de ingeniería de sistemas.

Puntos clave

Métodos ágiles: son métodos de desarrollo incremental que se enfocan en el diseño rápido, liberaciones frecuentes del software, reducción de gastos en el proceso y producción de código de alta calidad. Hacen que el cliente intervenga directamente en el proceso de desarrollo.

_ La decisión acerca de si se usa un enfoque de desarrollo ágil o uno basado en un plan depende del tipo de software que se va a elaborar, las capacidades del equipo de desarrollo y la cultura de la compañía que diseña el sistema.

Programación extrema: es un método ágil bien conocido que integra un rango de buenas prácticas de programación, como las liberaciones frecuentes del software, el mejoramiento continuo del software y la participación del cliente en el equipo de desarrollo.

_ Una fortaleza particular de la programación extrema, antes de crear una característica del programa, es el desarrollo de pruebas automatizadas. Todas las pruebas deben ejecutarse con éxito cuando un incremento se integra en un sistema.

Método de Scrum: es un método ágil que ofrece un marco de referencia para la administración del proyecto. Se centra alrededor de un conjunto de sprints, que son periodos fijos cuando se desarrolla un incremento de sistema. La planeación se basa en priorizar un atraso de trabajo y seleccionar las tareas de importancia más alta para un sprint.

_ Resulta difícil el escalamiento de los métodos ágiles para sistemas grandes, ya que éstos necesitan diseño frontal y cierta documentación. La integración continua es prácticamente imposible cuando existen muchos equipos de desarrollo separados que trabajan en un proyecto

Ingeniería de requerimientos

Introducción

Requerimientos: para un sistema, estos son descripciones de lo que el sistema debe hacer, es decir, el servicio que ofrece y las restricciones en su operación. Tales requerimientos reflejan las necesidades de los clientes por un sistema que atienda cierto propósito, como sería controlar un dispositivo, colocar un pedido o buscar información.

_ El término “requerimiento” no se usa de manera continua en la industria del software. En algunos casos, un requerimiento es simplemente un enunciado abstracto de un servicio que debe proporcionar un sistema, o bien, una restricción sobre un sistema, en donde no se busca como implementarlo sino entenderlo.

_ Otra de las cosas importante de los requerimientos es que estos pueden ser ambiguos, y con esto nos referimos a que nunca podemos asumir de que estamos entendiendo de la manera correcta lo que un cliente o que un futuro usuario nos está diciendo, y esto tiene que ver porque en general las personas que desarrollan productos de software son técnicos pero no son especialistas para el dominio en el cual el producto se está desarrollando, por ende necesitamos que los especialistas en el dominio sean capaces de transmitirnos las necesidades, pero además de eso debemos poder entenderlas perfectamente. Lo que se utiliza para transmitir las necesidades es el lenguaje natural que en sí mismo es ambiguo, donde una misma palabra en distintos dominios puede querer decir cosas muy diferentes, entonces es necesario que se unifique el lenguaje con el cual uno está hablando y no sea ambiguo

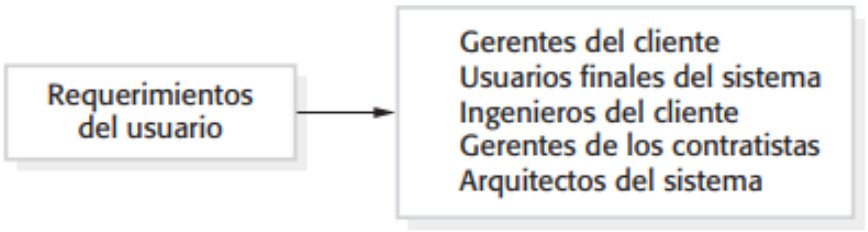
para que se entienda el significado de tales palabras en ese entorno. Este es uno de los orígenes de los problemas en los requerimientos. Entonces:

- Debe estar abierto a la interpretación.
- Debe estar definido con detalle.
- Estas declaraciones pueden ser llamados requerimientos.

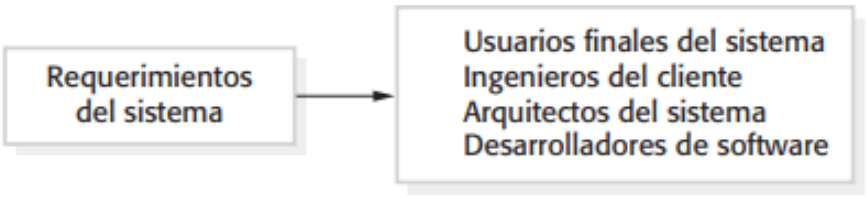
Ingeniería de requerimientos: se conoce asi al proceso de descubrir, analizar, documentar y verificar estos servicios y restricciones.

_ Algunos de los problemas que surgen durante el proceso de ingeniería de requerimientos son resultado del fracaso de hacer una separación clara entre esos diferentes niveles de descripción. Podemos distinguir el uso del término "requerimientos del usuario" para representar los requerimientos abstractos; y "requerimientos del sistema" para caracterizar la descripción detallada de lo que el sistema debe hacer.

Requerimientos del usuario: son enunciados, en un lenguaje natural junto con diagramas, acerca de qué servicios esperan los usuarios del sistema, y de las restricciones con las cuales éste debe operar.



Requerimientos del sistema: son descripciones más detalladas de las funciones, los servicios y las restricciones operacionales del sistema de software. El documento de requerimientos del sistema (llamado en ocasiones especificación funcional) tiene que definir con exactitud lo que se implementará. Puede formar parte del contrato entre el comprador del sistema y los desarrolladores del software.



_ Puede que en el requerimiento de usuario se entienda cual es el objetivo, pero no es clave a nivel de requerimiento como para poder pensar después como lo podemos hacer. Entonces a ese requerimiento, se lo tiene que trabajar para llevarlo a un requerimiento de sistema. A los requerimientos de sistema se los tiene que controlar con la persona que sabe del dominio.

Requerimientos funcionales y no funcionales

_ A menudo, los requerimientos del sistema de software se clasifican como:

Requerimientos funcionales

_ Estos requerimientos son enunciados acerca de servicios que el sistema debe proveer, de cómo debería reaccionar el sistema a entradas particulares y de cómo debería comportarse el sistema en situaciones específicas. En algunos casos, los requerimientos funcionales también explican lo que no debe hacer el sistema.

_ Los requerimientos funcionales para un sistema refieren lo que el sistema debe hacer. Tales requerimientos dependen del tipo de software que se esté desarrollando, de los usuarios esperados del software y del enfoque general que adopta la organización cuando se escriben los requerimientos. Al expresarse como requerimientos del usuario, los requerimientos funcionales se describen por lo general de forma abstracta que entiendan los usuarios del sistema. Sin embargo, requerimientos funcionales más específicos del sistema detallan las funciones del sistema, sus entradas y salidas, sus excepciones, etcétera.

- Los requerimientos funcionales del sistema varían desde requerimientos generales que cubren lo que tiene que hacer el sistema, hasta requerimientos muy específicos que reflejan maneras locales de trabajar o los sistemas existentes de una organización.
- La inexactitud en la especificación de requerimientos causa muchos problemas en la ingeniería de software. Es natural que un desarrollador de sistemas interprete un requerimiento ambiguo de forma que simplifique su implementación. Sin embargo, con frecuencia, esto no es lo que desea el cliente. Tienen que establecerse nuevos requerimientos y efectuar cambios al sistema. Desde luego, esto aplaza la entrega del sistema y aumenta los costos.

Requerimientos no funcionales

_ Estos requerimientos son limitaciones sobre servicios o funciones que ofrece el sistema. Incluyen restricciones tanto de temporización y del proceso de desarrollo, como impuestas por los estándares. Los requerimientos no funcionales se suelen aplicar al sistema como un todo, más que a características o a servicios individuales del sistema.

_ Los requerimientos no funcionales, son requerimientos que no se relacionan directamente con los servicios específicos que el sistema entrega a sus usuarios. Pueden relacionarse con propiedades emergentes del sistema, como fiabilidad, tiempo de respuesta y uso de almacenamiento. De forma alternativa, pueden definir restricciones sobre la implementación del sistema, como las capacidades de los dispositivos I/O, o las representaciones de datos usados en las interfaces con otros sistemas.

_ Los requerimientos no funcionales, como el rendimiento, la seguridad o la disponibilidad, especifican o restringen por lo general características del sistema como un todo. Los requerimientos no funcionales a menudo son más significativos que los requerimientos funcionales individuales. Es común que los usuarios del sistema encuentren formas para trabajar en torno a una función del sistema que realmente no cubre sus necesidades. No obstante, el fracaso para cubrir los requerimientos no funcionales haría que todo el sistema fuera inútil. Por ejemplo, si un sistema de aeronave no cubre sus requerimientos de fiabilidad, no será certificado para su operación como dispositivo seguro; si un sistema de control embebido fracasa para cubrir sus requerimientos de rendimiento, no operarán correctamente las funciones de control.

_ Los requerimientos no funcionales provienen de características requeridas del software (requerimientos del producto), la organización que desarrolla el software (requerimientos de la organización) o de fuentes externas:

Requerimientos del producto: estos requerimientos especifican o restringen el comportamiento del software. Los tipos son:

- Disponibilidad o rendimiento
- Seguridad
- Usabilidad

Requerimientos de la organización: son requerimientos de sistemas amplios, derivados de políticas y procedimientos en la organización del cliente y del desarrollador. Los ejemplos incluyen requerimientos del proceso operacional que definen cómo se usará el sistema, requerimientos del proceso de desarrollo que especifican el lenguaje de programación, estándares del entorno o el proceso de desarrollo a utilizar, y requerimientos ambientales que definen el entorno de operación del sistema.

Requerimientos externos: este término cubre todos los requerimientos derivados de factores externos al sistema y su proceso de desarrollo. En ellos se incluyen requerimientos regulatorios que establecen lo que debe hacer el sistema para ser aprobado en su uso por un regulador, como sería un banco central; requerimientos legislativos que tienen que seguirse para garantizar que el sistema opere conforme a la ley, y requerimientos éticos que garanticen que el sistema será aceptable para sus usuarios y el público en general.

_ A continuación vemos las métricas para especificar los requerimientos no funcionales:

- Rapidez
- Tamaño
- Facilidad de uso
- Fiabilidad
- Robustez
- Portabilidad

_ Estos requerimientos son difíciles de separar y además entran en conflicto e interactúan con otros requerimientos funcionales o no funcionales.

El documento de requerimientos de software

Documento de requerimientos de software: (llamado algunas veces especificación de requerimientos de software o SRS) es un comunicado oficial de lo que deben implementar los desarrolladores del sistema. Incluye tanto los requerimientos del usuario para un sistema, como una especificación detallada de los requerimientos del sistema. En ocasiones, los requerimientos del usuario y del sistema se integran en una sola descripción. En otros casos, los requerimientos del usuario se definen en una introducción a la especificación de requerimientos del sistema. Si hay un gran número de requerimientos, los requerimientos del sistema detallados podrían presentarse en un documento aparte. Entonces, este no es un documento de diseño. En la medida de lo posible, debería establecer de lo que el sistema debe hacer y no, y como es que debe hacerlo.

Usuarios de un documento de requerimientos

Clientes del sistema: especifican los requerimientos y los leen para comprobar que cubren sus necesidades. Los clientes especifican los cambios a los requerimientos.

Administradores: usan el documento de requerimientos para planear una cotización para el sistema y el proceso de desarrollo del sistema.

Ingenieros del sistema: usan los requerimientos para entender qué sistema debe desarrollarse.

Ingenieros de prueba del sistema: usan los requerimientos para desarrollar pruebas de validación para el sistema.

Ingenieros de mantenimiento del sistema: usan los requerimientos para comprender el sistema y las relaciones entre sus componentes.

_ La flexibilidad del lenguaje natural, que es tan útil para la especificación, causa problemas frecuentemente. Hay espacio para escribir requerimientos poco claros, y los lectores (los diseñadores) pueden malinterpretar los requerimientos porque tienen un antecedente diferente al del usuario. Es fácil mezclar muchos requerimientos en una sola oración y quizá sea difícil estructurar los requerimientos en lenguaje natural.

Especificación de requerimientos

Especificación de requerimientos: es el proceso de escribir, en un documento de requerimientos, los requerimientos del usuario y del sistema. De manera ideal, los requerimientos del usuario y del sistema deben ser claros, sin ambigüedades, fáciles de entender, completos y consistentes. Esto en la práctica es difícil de lograr, ya que los participantes interpretan los requerimientos de formas diferentes y con frecuencia en los requerimientos hay conflictos e inconsistencias inherentes. Los requerimientos del usuario para un sistema deben describir los requerimientos funcionales y no

funcionales, de forma que sean comprensibles para los usuarios del sistema que no cuentan con un conocimiento técnico detallado. De manera ideal, deberían especificar sólo el comportamiento externo del sistema.

_ El documento de requerimientos no debe incluir detalles de la arquitectura o el diseño del sistema. En consecuencia, si usted escribe los requerimientos del usuario, no tiene que usar jerga de software, anotaciones estructuradas o formales. Debe escribir los requerimientos del usuario en lenguaje natural, con tablas y formas sencillas, así como diagramas intuitivos.

_ A continuación tenemos formas de escribir una especificación de requerimientos del sistema:

- Enunciados en lenguaje natural
- Lenguaje natural estructurado
- Lenguajes de descripción de diseño: usa un lenguaje como un lenguaje de programación.
- Anotaciones gráficas: modelos gráficos.
- Especificaciones matemáticas: anotaciones basadas en conceptos matemáticos.

Problemas del lenguaje natural

_ Los requerimientos del usuario se escriben casi siempre en lenguaje natural, complementado con diagramas y tablas adecuados en el documento de requerimientos. Los requerimientos del sistema se escriben también en lenguaje natural, pero de igual modo se utilizan otras notaciones basadas en formas, modelos gráficos del sistema o modelos matemáticos del sistema. Algunos problemas con este son:

- Es ambiguo.
- En general cuando una persona está pidiendo funcionalidades, por ahí nos dan varios requerimientos juntos y funcionales y no funcionales como la misma cosa.
- Falta de claridad: la precisión es difícil sin hacer que el documento sea difícil de leer.
- Confusión de requerimientos: requisitos funcionales y no funcionales tienden a ser confusos.
- Fusión de requerimientos: varios requerimientos diferentes pueden expresarse juntos.

Procesos de ingeniería de requerimientos

_ Los procesos de ingeniería de requerimientos incluyen cuatro actividades de alto nivel. Éstas se enfocan en valorar si el sistema es útil para la empresa (estudio de factibilidad), descubrir requerimientos (adquisición y análisis), convertir dichos requerimientos en alguna forma estándar (especificación) y comprobar que los requerimientos definan realmente el sistema que quiere el cliente (validación). En la

práctica, la ingeniería de requerimientos es un proceso iterativo donde las actividades están entrelazadas.

_ Las actividades están organizadas como un proceso iterativo alrededor de una espiral, y la salida es un documento de requerimientos del sistema. La cantidad de tiempo y esfuerzo dedicados a cada actividad en cada iteración depende de la etapa del proceso global y el tipo de sistema que está siendo desarrollado. En el inicio del proceso, se empleará más esfuerzo para comprender los requerimientos empresariales de alto nivel y los no funcionales, así como los requerimientos del usuario para el sistema. Estas son las cuatro actividades fundamentales:

Obtención de requerimientos: hablar, preguntar, pensar, modificar

Análisis de requerimientos: al igual que el anterior va a ser iterativo.

Validación de requerimientos: tenemos que ver si los requerimientos son ciertos.

Gestión de requerimientos: en una organización con muchas áreas, es probable que los requerimientos de un área se interpongan con los de otra, para esto tendremos que buscar desde un punto de vista técnico y resolverlo para poder llevar adelante el proceso habiendo conciliado esos requerimientos que son contrapuestos.

Obtención y análisis de requerimientos

_ Después de un estudio de factibilidad inicial, la siguiente etapa del proceso de ingeniería de requerimientos es la adquisición y el análisis de requerimientos. En esta actividad, los ingenieros de software trabajan con clientes y usuarios finales del sistema para descubrir el dominio de aplicación, qué servicios debe proporcionar el sistema, el desempeño requerido de éste, las restricciones de hardware, etcétera.

_ En una organización, la adquisición y el análisis de requerimientos pueden involucrar a diversas clases de personas. Un participante en el sistema es quien debe tener alguna influencia directa o indirecta sobre los requerimientos del mismo. Los

participantes incluyen a usuarios finales que interactuarán con el sistema, y a cualquiera en una organización que resultará afectada por él. Otros participantes del sistema pueden ser los ingenieros que desarrollan o mantienen otros sistemas relacionados, administradores de negocios, expertos de dominio y representantes de asociaciones sindicales.

_ Las actividades del proceso son:

Descubrimiento de requerimientos: es el proceso de interactuar con los participantes del sistema para descubrir sus requerimientos. También los requerimientos de dominio de los participantes y la documentación se descubren durante esta actividad. Existen numerosas técnicas complementarias que pueden usarse para el descubrimiento de requerimientos, las cuales se estudian más adelante en esta sección.

Clasificación y organización de requerimientos: esta actividad toma la compilación no estructurada de requerimientos, agrupa requerimientos relacionados y los organiza en grupos coherentes. La forma más común de agrupar requerimientos es usar un modelo de la arquitectura del sistema, para identificar subsistemas y asociar los requerimientos con cada subsistema. En la práctica, la ingeniería de requerimientos y el diseño arquitectónico no son actividades separadas completamente.

Priorización y negociación de requerimientos: inevitablemente, cuando intervienen diversos participantes, los requerimientos entrarán en conflicto. Esta actividad se preocupa por priorizar los requerimientos, así como por encontrar y resolver conflictos de requerimientos mediante la negociación. Por lo general, los participantes tienen que reunirse para resolver las diferencias y estar de acuerdo con el compromiso de los requerimientos.

Especificación de requerimientos: los requerimientos se documentan e ingresan en la siguiente ronda de la espiral. Pueden producirse documentos de requerimientos formales o informales.

Entrevistas

_ Las entrevistas formales o informales con participantes del sistema son una parte de la mayoría de los procesos de ingeniería de requerimientos. En estas entrevistas, el equipo de ingeniería de requerimientos formula preguntas a los participantes sobre el sistema que actualmente usan y el sistema que se va a desarrollar. Los requerimientos se derivan de las respuestas a dichas preguntas. Las entrevistas son de dos tipos:

Entrevistas cerradas: donde los participantes responden a un conjunto de preguntas pre-establecidas.

Entrevistas abiertas: en las cuales no hay agenda predefinida. El equipo de ingeniería de requerimientos explora un rango de conflictos con los participantes del sistema y, como resultado, desarrolla una mejor comprensión de sus necesidades.

_ Las entrevistas tampoco son una técnica efectiva para adquirir conocimiento sobre los requerimientos y las restricciones de la organización, porque existen relaciones sutiles de poder entre los diferentes miembros en la organización. En la práctica, las entrevistas con los participantes son por lo general una combinación de ambas. Quizá se deba obtener la respuesta a ciertas preguntas, pero eso a menudo conduce a otros temas que se discuten en una forma menos estructurada. Rara vez funcionan bien las discusiones completamente abiertas. Con frecuencia debe plantear algunas preguntas para comenzar y mantener la entrevista enfocada en el sistema que se va a desarrollar.

Escenarios

_ Los escenarios son particularmente útiles para detallar un bosquejo de descripción de requerimientos. Se trata de ejemplos sobre descripciones de sesiones de interacción. Cada escenario abarca comúnmente una interacción o un número pequeño de interacciones posibles. Se desarrollan diferentes formas de escenarios y se ofrecen varios tipos de información con diversos niveles de detalle acerca del sistema. Las historias de usuario que se usan en programación extrema, son un tipo de escenario de requerimientos.

_ Un escenario comienza con un bosquejo de la interacción. Durante el proceso de adquisición, se suman detalles a éste para crear una representación completa de dicha interacción. En su forma más general, un escenario puede incluir:

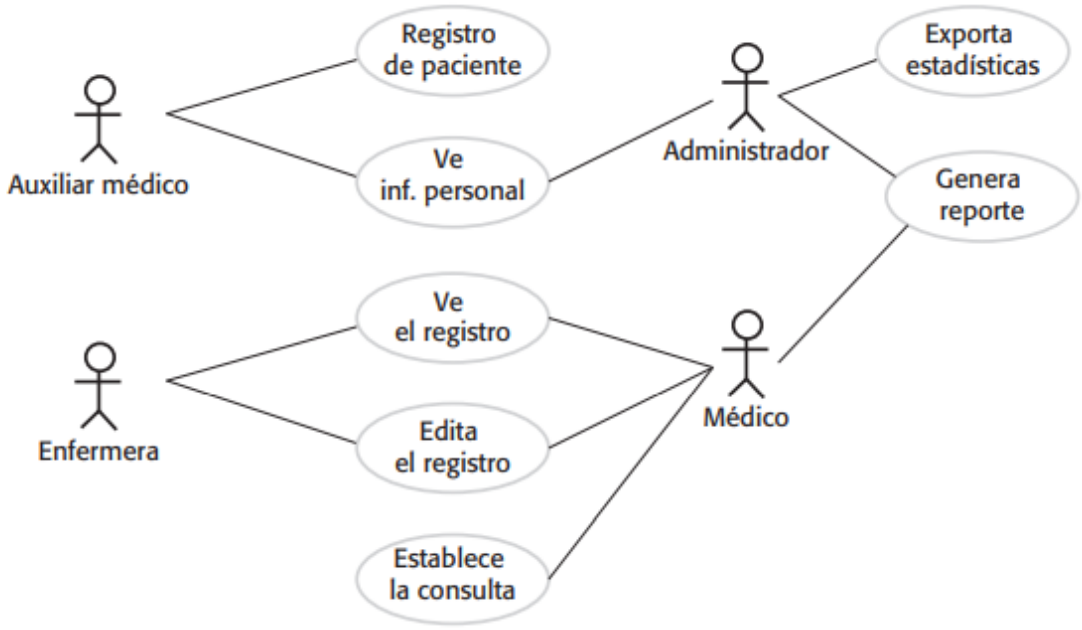
- Una descripción de qué esperan el sistema y los usuarios cuando inicia el escenario.
- Una descripción en el escenario del flujo normal de los eventos.
- Una descripción de qué puede salir mal y cómo se manejaría.
- Información de otras actividades que estén en marcha al mismo tiempo.
- Una descripción del estado del sistema cuando termina el escenario.

Casos de uso

_ Los casos de uso son una técnica de descubrimiento de requerimientos que se introdujo por primera vez en el método Objectory. Un caso de uso identifica a los actores implicados en una interacción, y nombra el tipo de interacción. Entonces, esto se complementa con información adicional que describe la interacción con el sistema. La información adicional puede ser una descripción textual, o bien, uno o más modelos gráficos como una secuencia UML o un gráfico de estado.

_ Los casos de uso se documentan con el empleo de un diagrama de caso de uso de alto nivel. El conjunto de casos de uso representa todas las interacciones posibles que se describirán en los requerimientos del sistema. Los actores en el proceso, que pueden ser individuos u otros sistemas, se representan como figuras sencillas. Cada clase de interacción se constituye como una elipse con etiqueta. Líneas vinculan a los actores con la interacción. De manera opcional, se agregan puntas de flecha a las líneas para mostrar cómo se inicia la interacción.

_ Los escenarios y los casos de uso son técnicas efectivas para adquirir requerimientos de los participantes que interactúan directamente con el sistema.



Puntos clave

Requerimientos: para un sistema de software establecen lo que debe hacer el sistema y definen las restricciones sobre su operación e implementación.

Requerimientos funcionales: son enunciados de los servicios que debe proporcionar el sistema, o descripciones de cómo deben realizarse algunos cálculos.

Requerimientos no funcionales: restringen con frecuencia el sistema que se va a desarrollar y el proceso de desarrollo a usar. Éstos pueden ser requerimientos del producto, requerimientos organizacionales o requerimientos externos. A menudo se relacionan con las propiedades emergentes del sistema y, por lo tanto, se aplican al sistema en su conjunto.

Documento de requerimientos de software: es un enunciado acordado sobre los requerimientos del sistema. Debe organizarse de forma que puedan usarlo tanto los clientes del sistema como los desarrolladores del software.

Proceso de ingeniería de requerimientos: incluye un estudio de factibilidad, adquisición y análisis de requerimientos, especificación de requerimientos, validación de requerimientos y administración de requerimientos.

Adquisición y el análisis de requerimientos: es un proceso iterativo que se representa como una espiral de actividades: descubrimiento de requerimientos, clasificación y organización de requerimientos, negociación de requerimientos y documentación de requerimientos.

Validación de requerimientos: es el proceso de comprobar la validez, la consistencia, la totalidad, el realismo y la verificabilidad de los requerimientos.

_ Los cambios empresariales, organizacionales y técnicos conducen inevitablemente a cambios en los requerimientos para un sistema de software. La administración de requerimientos es el proceso de gestionar y controlar dichos cambios.

Usabilidad

Definición

_ La usabilidad es la facilidad con que las personas pueden utilizar una herramienta particular u otro objeto fabricado por humanos con el fin de alcanzar un objetivo concreto. La usabilidad es la cualidad que tiene un sistema por la cual permite a sus usuarios alcanzar objetivos específicos con:

- Efectividad
- Eficiencia
- Satisfacción

_ El concepto en torno al cual gravita la usabilidad es la calidad de uso. El grado de usabilidad de un sistema es una medida empírica y relativa de su usabilidad. Se mide a partir de las siguientes pruebas:

Empíricas: porque no se basa en opiniones o sensaciones, sino en pruebas de usabilidad realizadas en laboratorio u observadas mediante trabajo de campo.

Relativas: porque el resultado no es ni bueno ni malo, sino que depende de las metas planteadas.

_ La usabilidad es un atributo de calidad que, dependiendo de los usuarios, las tareas y el contexto, mide:

- La facilidad de aprendizaje
- La eficiencia motriz y cognitiva
- La capacidad de recordar lo aprendido
- El manejo de errores
- La satisfacción subjetiva

Principios

Facilidad de Aprendizaje

_ Facilidad con la que nuevos usuarios desarrollan una interacción efectiva con el sistema o producto. Está relacionada con:

- La predicibilidad
- La sintetización
- La familiaridad

- La generalización de los conocimientos previos
- La consistencia

Facilidad de Uso

_ Facilidad con la que el usuario hace uso de la herramienta, con menos pasos o más naturales a su formación específica. Está relacionada con:

- La eficacia
- La eficiencia

Flexibilidad

_ Relativa a la variedad de posibilidades con las que el usuario y el sistema pueden intercambiar información. Abarca la posibilidad de diálogo, la multiplicidad de vías para realizar la tarea, similitud con tareas anteriores y la optimización entre el usuario y el sistema.

Robustez

_ Es el nivel de apoyo al usuario que facilita el cumplimiento de sus objetivos. Está relacionado con la capacidad de observación del usuario, de recuperación de información y de ajuste de la tarea al usuario.

Diseño centrado en el usuario (DCU)

_ Este tipo de diseños reconocen las necesidades y los intereses de los usuarios, y se basan en el diseño basado en las capacidades y limitaciones naturales de las personas a fin de lograr una mayor facilidad de uso, y permiten la operación sin necesidad de instrucciones o únicamente durante la primera vez. En el ámbito de las interacciones humano-computadora, el DCU pone al usuario como eje central de todos los procesos relacionados a la ingeniería de software.

Objetivos:

- Satisfacer las necesidades de todos sus usuarios potenciales.
- Adaptar la tecnología utilizada a las expectativas de los usuarios.
- Crear interfaces que faciliten la consecución de los objetivos de los usuarios.

Beneficios:

- Aumento de la productividad: un sistema diseñado siguiendo los principios de usabilidad, y adaptado de manera de trabajar del usuario, permitirá más efectividad en lugar de perder el tiempo luchando con un complejo conjunto de funciones. Un sistema usable permitirá al usuario concentrarse en la tarea en lugar de la herramienta.
- Reducción de errores: una significativa proporción de error humano a menudo se le puede atribuir a una interfaz de usuario mal diseñada. Evitar incoherencias, ambigüedades u otras faltas del diseño de la interfaz generan menor cantidad de errores del usuario.

- Menor capacitación y entrenamiento: un sistema bien diseñado y usable puede reforzar el aprendizaje, reduciendo así el tiempo de formación y la necesidad de apoyo humano.
- Aceptación: mejorar la aceptación del usuario es a menudo una medida de resultado indirecta del diseño de un sistema utilizable. La mayoría de los usuarios prefieren utilizar, un sistema bien diseñado que proporcione información que pueda ser fácilmente accedida y que se presenta en un formato que sea fácil de asimilar y utilizar.

Fases

Entender y especificar el contexto de uso: identificar a las personas a las que se dirige el producto, para qué lo usarán y en qué condiciones.

Especificar requisitos: identificar los objetivos del usuario y del proveedor del producto a satisfacer.

Producir soluciones de diseño: esta fase se puede subdividir en diferentes etapas secuenciales, desde las primeras soluciones conceptuales hasta la solución final de diseño; generalmente utilizando técnicas de prototipado.

Evaluar: es la fase más importante del proceso, en la que se validan las soluciones de diseño (el sistema satisface los requisitos), o por el contrario se detectan problemas de usabilidad, normalmente a través de pruebas con usuarios.

Principios básicos

Participación activa de los usuarios: entendimiento de estos y de las tareas que requieren. Al incorporar a los usuarios finales al proceso desde el inicio, me mejora además la aceptación y el compromiso con el sistema, al sentir que ha sido diseñado teniendo en cuenta sus necesidades y no ha sido impuesto.

Iteración en el diseño: esto implica recibir realimentación por parte de los usuarios finales después de su uso en varias etapas, las cuales pueden ir desde simples maquetas con papel hasta prototipos de software con mayor grado de fidelidad. La respuesta de cada ciclo de iteración se utiliza para desarrollar el siguiente diseño. Se intenta además lograr las condiciones más similares al mundo real en el cual se desenvolverán los usuarios con el sistema.

Utilizar un equipo multidisciplinario: el DCU es un proceso de colaboración que se beneficia de la participación activa de diversas partes cada una de las cuales tiene conocimiento y experiencia específicos para compartir con el resto. El equipo podría incluir gerentes, especialistas en usabilidad, usuarios finales, ingenieros de software, diseñadores gráficos, diseñadores de interacción y personal de capacitación y apoyo.

Metodologías y técnicas del DCU

Identificación de los usuarios:

- Usuarios primarios: las personas que usarán el producto final para realizar una tarea.
- Usuarios secundarios: los que ocasionalmente pueden usar el producto, o aquellos que lo consumen a través de un intermediario.
- Usuarios terciarios: los que se verán afectados por el uso del producto, o pueden tomar decisiones sobre el mismo

_ Para que el diseño de un producto sea exitoso se deben tener en cuenta los tres niveles de usuarios.

Prototipado: una vez que las partes interesadas han sido identificadas y se ha hecho una investigación completa de sus necesidades, los diseñadores pueden desarrollar soluciones con diseños alternativos, los cuales pueden ser evaluados por los usuarios. Estas alternativas pueden ser simples dibujos en lápiz y papel en la fase inicial del proceso. También es posible escuchar a los usuarios discutir sobre los diseños alternativos presentados, lo cual posibilita amplificar la comprensión de los diseñadores y pueden proporcionar información que no se obtuvo en las entrevistas iniciales, en otras palabras, las observaciones y el análisis de necesidades ya realizado.

Proceso de evaluación: a medida que avanza el ciclo del diseño, los prototipos (versiones iniciales y limitadas del producto) pueden ser producidos y luego probados por los usuarios. Las evaluaciones ayudarán a identificar criterios medibles de usabilidad:

- Eficacia: en relación por ejemplo a cuántas veces los usuarios logran terminar las tareas, si lograron terminar las tareas sin dificultad, etc.
- Facilidad de Aprendizaje (Learnability): en relación por ejemplo a cuán fácil resulta para los usuarios llevar a cabo tareas básicas la primera vez que se enfrentan al diseño, etc.
- Eficiencia: en relación por ejemplo a una vez que los usuarios han aprendido el funcionamiento básico del diseño, cuánto tardan en la realización de tareas, etc.
- Cualidad de ser recordado (Memorability): en relación por ejemplo a en que momento los usuarios vuelven a usar el diseño después de un periodo sin hacerlo, cuánto tardan en volver a adquirir el conocimiento necesario para usarlo eficientemente, etc.
- Tasa de errores de usuario: en relación por ejemplo a, durante la realización de una tarea, cuántos errores comete el usuario, qué tan graves son las consecuencias de esos errores, qué tan rápido puede el usuario deshacer las consecuencias de sus propios errores, etc.
- Satisfacción subjetiva: en relación por ejemplo a cuán agradable y sencillo le ha parecido al usuario la realización de las tareas, etc.

_ Las evaluaciones también revelarán la satisfacción de los usuarios con el producto. Esto se logra solamente a través de comentarios recogidos en un proceso interactivo e iterativo con la participación de los usuarios.

Pruebas de usabilidad

_ Se utilizan metodologías que requieren la participación de usuarios reales o de personas que concuerdan con el perfil de los futuros usuarios.

Investigación Etnográfica: se observa a los usuarios en el lugar donde normalmente se utiliza el sistema (por ejemplo, trabajo, hogar, etc.) para recopilar datos sobre quiénes son sus usuarios, cuáles son las tareas y metas que se han relacionado con el sistema, y el contexto en el que trabajan para lograr sus objetivos. A partir de esta investigación cualitativa, se puede desarrollar perfiles de usuario, personajes arquetípicos (los usuarios), los escenarios y descripciones de tareas.

Diseño Participativo: si bien no se considera una técnica en sí misma, es más bien una forma de realización del DCU, el diseño participativo emplea a uno o más usuarios representativos durante todo el proceso de desarrollo. Este enfoque posiciona al usuario final en el corazón del proceso de diseño, desde el inicio mismo del proyecto, aprovechando el conocimiento del usuario, habilidades, e incluso las reacciones emocionales en el diseño.

Grupos Focales (Focus Group): son utilizados en las primeras etapas de un proyecto para evaluar conceptos preliminares con usuarios representativos. El objetivo es identificar si los conceptos principales, son satisfactorios o no, y cómo podrían ser más aceptables y útiles Se explora sobre cómo los usuarios finales piensan y sienten. Un grupo focal es bueno para la información general, cualitativa, pero no para aprender sobre los problemas de rendimiento y los comportamientos reales del sistema. Las pruebas de usabilidad se consideran mejores para la observación de los comportamientos y la medición de los problemas de rendimiento.

Entrevistas: con usuarios son una poderosa herramienta cualitativa, pero no para evaluar la usabilidad de un diseño, sino para descubrir deseos, motivaciones, valores y experiencias de nuestros usuarios. Durante estas entrevistas, el entrevistador debe mostrarse neutral y no dirigir o condicionar las respuestas del entrevistado. Lo que se pretende es descubrir información que oriente en el diseño, no confirmar creencias sobre cómo son los usuarios.

Encuestas: también son una herramienta útil y se pueden utilizar en cualquier momento del ciclo de vida, pero se utiliza con mayor frecuencia en las primeras etapas para comprender mejor el potencial usuario. Son una herramienta de investigación cuantitativa que complementan a las entrevistas, y permiten medir con validez estadística los conceptos hallados con técnicas cualitativas.

Card sorting: consiste en solicitar a un grupo de participantes (los cuales deben tener un perfil acorde con la audiencia a la que se dirige el producto) que agrupen los conceptos representados en tarjetas por su similitud semántica. Con el objetivo de

identificar qué conceptos, de los representados en cada tarjeta, tienen relación semántica entre sí, e incluso cuál es el grado de esa relación.

_ El card sorting es una prueba destinada a comprender la forma en que los usuarios estructuran la información para asegurar que será comprendida fácilmente, por tanto, tiene lugar en etapas tempranas del proyecto.

Usability Testing: emplea técnicas para recoger datos empíricos mediante la observación de usuarios finales que utilizan el sistema para realizar tareas en tiempo real. Las pruebas se dividen en dos enfoques principales

- Primero: pruebas formales realizadas como verdaderos experimentos, con el fin de confirmar o refutar hipótesis específicas.
- Segundo: menos formal, pero riguroso, emplea a un ciclo repetitivo de pruebas destinadas a exponer las deficiencias de usabilidad y moldear el producto.

"Think-aloud" (pensamiento en voz alta): consiste en solicitar al participante que exprese verbalmente durante la prueba qué está pensando, qué no entiende, por qué lleva a cabo una acción o duda.

Estudios de Seguimiento: se realizan después de la liberación formal del sistema. La idea es recoger datos que se utilizarán para la próxima versión, a través de encuestas, entrevistas y observaciones. Una vez estructurados, los estudios de seguimiento son, probablemente, las apreciaciones más adecuadas y más precisas de la facilidad de uso de un sistema.

Evaluación Heurística: se realiza por inspección, varios expertos inspeccionan y analizan el diseño en busca de potenciales problemas de usabilidad, comprobando para ello el cumplimiento de principios de diseño usable (principios heurísticos) previamente establecidos.

Las ocho reglas de oro de Shneiderman

1)_ Consistencia: es importante el uso de iconos, colores, botones, etc. que sean familiares aprovechando el conocimiento previo que tiene el usuario. Los usuarios se usan algo nuevo. Esto ayuda a que los usuarios puedan realizar lo que desean más rápidamente.

2)_ Permitir que los usuarios frecuentes usen atajos: con el constante uso de un producto o servicio, se demandan formas más rápidas para realizar las tareas. Como ejemplo, secuencias del teclado para copiar, pegar, etc. mientras el usuario va adquiriendo experiencia, pueden navegar y utilizar el interfaz más rápido y sin esfuerzo.

3)_ Retroalimentación informativa: los usuarios deben saber en donde están y que es lo que está pasando todo el tiempo. Cada acción, debe tener una retroalimentación legible y razonable. UN mal ejemplo son los mensajes de alerta que muestran códigos de error incomprensibles para el usuario.

4)_ Diseñar textos de diálogo para cerrar procesos: los usuarios deben saber cuál ha sido el resultado de sus acciones o actos. Por ejemplo, el cuándo se completa una transacción en línea es necesario que sea informado todo lo relativo a la operación apenas concluida.

5)_ Manejo de errores: ofrecer una forma sencilla de corregir errores. Los sistemas deben diseñarse para evitar que los usuarios comentan de errores, pero, cuando esto suceda, deben recibir una solución simple para resolverlo. Por ejemplo, si hay campos obligatorios en un formulario, pueden resaltarse para mejorar la identificación.

6)_ Permitir deshacer operaciones: se deben ofrecer formas obvias y sencillas de retroceder o revertir acciones. Esto debe de permitirse en varios puntos, ya sea después de una acción, una captura de datos o unas secuencias de acciones. “Esta función libera ansiedad, como el usuario se da cuenta que el error puede corregirse, le da el valor para explorar opciones, funciones o características desconocidas”

7)_ Fomentar la sensación de control: permitir que el usuario sea el que inicia las cosas. Es importante que tenga la sensación de que están en completo control de los eventos que ocurren en el espacio digital.

8)_ Reducir la carga de memoria a corto plazo: la atención humana es limitada. La interfaz debe ser lo más sencilla posible y con una jerarquía de información evidente. Elegir reconocimiento en vez de recuerdo. Reconocer es más fácil que recordar, el reconocimiento incluye claves que ayudan a recordar objetos almacenados en la memoria.

Los diez principios Heurísticos de Nielsen

1)_ Visibilidad del estado del sistema: el sistema siempre debe mantener a los usuarios informados sobre lo que ocurre, a través de una retroalimentación apropiada en un tiempo razonable.

2)_ Consistencia entre el sistema y el mundo real: el sistema debe hablar en el lenguaje del usuario, con palabras, frases y conceptos familiares para él. Utilizar convenciones del mundo real, haciendo que la información aparezca en un orden natural y lógico.

3)_ Control y libertad del usuario: es frecuente que los usuarios elijan funcionalidades por error y necesitan un modo fácil para resolver la situación. Es importante ofrecer soporte para deshacer y rehacer acciones.

4)_ Consistencia y estándares: los usuarios no deben tener que preguntarse si las diversas palabras, situaciones, o acciones significan la misma cosa. Que se sigan las normas y convenciones de la plataforma sobre la que está implementando el sistema.

5)_ Prevención de errores: antes que diseñar buenos mensajes de error, es mejor evitar que el problema ocurra.

6)_ Reconocer es mejor que recordar: minimizar la carga de memoria del usuario haciendo que los objetos, las acciones y las opciones estén visibles. El usuario no debería tener que recordar la información de una parte del diálogo a otra.

7)_ Flexibilidad y eficiencia de uso: los aceleradores, no vistos por el usuario principiante, mejoran la interacción para el usuario experto de tal manera que el sistema puede servir para usuarios inexpertos y experimentados. Es importante que el sistema permita personalizar acciones frecuentes.

8)_ Diseño estético y minimalista: los diálogos no deberían contener información irrelevante. Cada unidad extra de información en un diálogo compite con la información importante, disminuyendo su visibilidad relativa.

9)_ Ayudar a reconocer, diagnosticar y recuperarse de errores: los mensajes de error deben estar expresados en lenguaje simple (sin códigos), indicando con precisión el problema y sugiriendo una solución.

10)_ Ayuda y documentación: aunque es mejor que se pueda usar el sistema sin documentación, es necesario proveer al usuario de ayuda y documentación. Esta tiene que ser fácil de buscar y entender, centrada en las tareas del usuario, con información de las etapas a realizar y no muy extensa.

Modelado de sistemas

Introducción

Modelado de sistemas: es el proceso para desarrollar modelos abstractos de un sistema, donde cada modelo presenta una visión o perspectiva diferente de dicho sistema. En general, el modelado de sistemas se ha convertido en un medio para representar el sistema usando algún tipo de notación gráfica, que ahora casi siempre se basa en notaciones en el Lenguaje de Modelado Unificado (UML).

_ Los modelos se usan durante el proceso de ingeniería de requerimientos para ayudar a derivar los requerimientos de un sistema, durante el proceso de diseño para describir el sistema a los ingenieros que implementan el sistema, y después de la implementación para documentar la estructura y la operación del sistema. Es posible desarrollar modelos tanto del sistema existente como del sistema a diseñar:

- Los modelos del sistema existente se usan durante la ingeniería de requerimientos. Ayudan a aclarar lo que hace el sistema existente y pueden utilizarse como base para discutir sus fortalezas y debilidades.
- Los modelos del sistema nuevo se emplean durante la ingeniería de requerimientos para ayudar a explicar los requerimientos propuestos a otros participantes del sistema.

_ El aspecto más importante de un modelo del sistema es que deja fuera los detalles. Un modelo es una abstracción del sistema a estudiar, y no una representación alternativa de dicho sistema. De manera ideal, una representación de un sistema debe

mantener toda la información sobre la entidad a representar. Una abstracción simplifica y recoge deliberadamente las características más destacadas.

_ La mayoría de los usuarios del UML consideraban que cinco tipos de diagrama podrían representar lo esencial de un sistema:

- Diagramas de actividad: muestran las actividades incluidas en un proceso o en el procesamiento de datos.
- Diagramas de caso de uso: exponen las interacciones entre un sistema y su entorno.
- Diagramas de secuencias: muestran las interacciones entre los actores y el sistema, y entre los componentes del sistema.
- Diagramas de clase: revelan las clases de objeto en el sistema y las asociaciones entre estas clases.
- Diagramas de estado: explican cómo reacciona el sistema frente a eventos internos y externos.

Modelos de contexto

_ Se utilizan modelos de contexto para ilustrar el contexto operativo de un sistema, muestran lo que se encuentra fuera de los límites del sistema. Las cuestiones organizacionales pueden influir en la decisión sobre dónde situar los límites del sistema. Los modelos de contexto muestran el sistema y su relación con otros sistemas.

_ Los modelos de contexto, simplemente muestran los otros sistemas en ambiente, no cómo se utiliza el sistema que está siendo desarrollado en ese entorno.

Límites del sistema

_ Los límites del sistema se establecen para definir lo que está dentro y lo que está fuera del sistema, muestran otros sistemas que se utilizan o dependen del sistema que está siendo desarrollado. La posición de los límites del sistema tiene un efecto profundo en los requisitos del sistema.

_ La definición del límite del sistema es fundamental, ya que si los límites del sistema aumentan y/o disminuyen, cambia la carga de trabajo de las diferentes partes de una organización.

Modelos de interacción

_ Todos los sistemas incluyen interacciones de algún tipo. Éstas son interacciones del usuario, que implican entradas y salidas del usuario; interacciones entre el sistema a desarrollar y otros sistemas; o interacciones entre los componentes del sistema.

- El modelado de interacción del usuario es importante, porque ayuda a identificar los requerimientos del usuario.
- El modelado de la interacción sistema a sistema destaca los problemas de comunicación que se lleguen a presentar.

- El modelado de interacción de componentes ayuda a entender si es probable que una estructura de un sistema propuesto obtenga el rendimiento y la confiabilidad requeridos por el sistema.

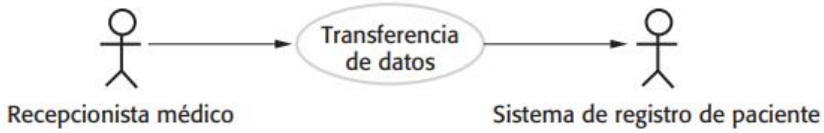
_ Hay dos enfoques relacionados con el modelado de interacción:

1. Modelado de caso de uso: se utiliza principalmente para modelar interacciones entre un sistema y actores externos (usuarios u otros sistemas).
2. Diagramas de secuencia: se emplean para modelar interacciones entre componentes del sistema, aunque también pueden incluirse agentes externos.

_ Los modelos de caso de uso y los diagramas de secuencia presentan la interacción a diferentes niveles de detalle y, por lo tanto, es posible utilizarlos juntos. Los detalles de las interacciones que hay en un caso de uso de alto nivel se documentan en un diagrama de secuencia.

Modelado de casos de uso

_ El modelado de casos de uso se utiliza ampliamente para apoyar la adquisición de requerimientos. Un caso de uso puede tomarse como un simple escenario que describa lo que espera el usuario de un sistema. Cada caso de uso representa una tarea discreta que implica interacción externa con un sistema. En su forma más simple, un caso de uso se muestra como una elipse, con los actores que intervienen en el caso de uso representados como figuras humanas.



Diagramas de secuencia

_ Los diagramas de secuencia en el UML se usan principalmente para modelar las interacciones entre los actores y los objetos en un sistema, así como las interacciones entre los objetos en sí. El UML tiene una amplia sintaxis para diagramas de secuencia, lo cual permite muchos tipos diferentes de interacción a modelar. Como sugiere el nombre, un diagrama de secuencia muestra la sucesión de interacciones que ocurre durante un caso de uso particular o una instancia de caso de uso. Los objetos y actores que intervienen se mencionan a lo largo de la parte superior del diagrama, con una línea punteada que se dibuja verticalmente a partir de éstos. Las interacciones entre los objetos se indican con flechas dirigidas. El rectángulo sobre las líneas punteadas indica la línea de vida del objeto tratado (es decir, el tiempo que la instancia del objeto está involucrada en la computación). La secuencia de interacciones se lee de arriba abajo. Las anotaciones sobre las flechas señalan las llamadas a los objetos, sus parámetros y los valores que regresan.

Modelos estructurales

_ Los modelos estructurales de software muestran la organización de un sistema, en términos de los componentes que constituyen dicho sistema y sus relaciones. Estos pueden ser:

- Modelos estáticos: muestran la estructura del diseño del sistema.
- Modelos dinámicos: revelan la organización del sistema cuando se ejecuta.

_ Los modelos estructurales de un sistema se crean cuando se discute y diseña la arquitectura del sistema.

Diagramas de clase

_ Los diagramas de clase pueden usarse cuando se desarrolla un modelo de sistema orientado a objetos para mostrar las clases en un sistema y las asociaciones entre dichas clases. De manera holgada, una clase de objeto se considera como una definición general de un tipo de objeto del sistema. Una asociación es un vínculo entre clases, que indica que hay una relación entre dicha clases. En consecuencia, cada clase puede tener algún conocimiento de esta clase asociada.

_ Cuando se desarrollan modelos durante las primeras etapas del proceso de ingeniería de software, los objetos representan algo en el mundo real, como un paciente, una receta, un médico, etcétera. Conforme se desarrolla una implementación, por lo general se necesitan definir los objetos de implementación adicionales que se usan para dar la funcionalidad requerida del sistema.

_ Los diagramas de clase en el UML pueden expresarse con diferentes niveles de detalle. Cuando se desarrolla un modelo, la primera etapa con frecuencia implica buscar en el mundo, identificar los objetos esenciales y representarlos como clases. La forma más sencilla de hacer esto es escribir el nombre de la clase en un recuadro. También puede anotar la existencia de una asociación dibujando simplemente una línea entre las clases.

Puntos claves

Modelo: es una visión abstracta de un sistema que ignora algunos detalles del sistema. Pueden desarrollarse modelos complementarios del sistema para mostrar el contexto, las interacciones, la estructura y el comportamiento del sistema.

Modelos de contexto: muestran cómo un sistema a modelar se coloca en un entorno con otros sistemas y procesos. Ayudan a definir las fronteras del sistema a desarrollar.

Diagramas de caso de uso y diagramas de secuencia: se emplean para describir las interacciones entre usuario/sistema a diseñar y usuarios/otros sistemas. Los casos de uso describen interacciones entre un sistema y actores externos; los diagramas de secuencia agregan más información a éstos al mostrar las interacciones entre objetos del sistema.

Modelos estructurales: indican la organización y arquitectura de un sistema. Los diagramas de clase se usan para definir la estructura estática de clases en un sistema y sus asociaciones.

Modelos del comportamiento: se usan para describir la conducta dinámica de un sistema en ejecución. Pueden modelarse desde la perspectiva de los datos procesados por el sistema, o mediante los eventos que estimulan respuestas de un sistema.

Diagramas de actividad: se utilizan para modelar el procesamiento de datos, en que cada actividad representa un paso del proceso.

Diagramas de estado: se utilizan para modelar el comportamiento de un sistema en respuesta a eventos internos o externos.

Ingeniería dirigida por modelo: es un enfoque al desarrollo del software donde un sistema se representa como un conjunto de modelos que pueden transformarse automáticamente a código ejecutable.

Diseño arquitectónico

Introducción

_ El diseño arquitectónico se interesa por entender cómo debe organizarse un sistema y cómo tiene que diseñarse la estructura global de ese sistema. En el modelo del proceso de desarrollo de software, el diseño arquitectónico es la primera etapa en el proceso de diseño del software. Es el enlace crucial entre el diseño y la ingeniería de requerimientos, ya que identifica los principales componentes estructurales en un sistema y la relación entre ellos. La salida del proceso de diseño arquitectónico consiste en un modelo arquitectónico que describe la forma en que se organiza el sistema como un conjunto de componentes en comunicación.

_ En la práctica, hay un significativo traslape entre los procesos de ingeniería de requerimientos y el diseño arquitectónico. De manera ideal, una especificación de sistema no debe incluir cierta información de diseño. La descomposición arquitectónica es por lo general necesaria para estructurar y organizar la especificación. Por lo tanto, como parte del proceso de ingeniería de requerimientos, se podría proponer una arquitectura de sistema abstracta donde se asocien grupos de funciones de sistemas o características con componentes o subsistemas a gran escala.

_ Las arquitecturas de software se diseñan en dos niveles de abstracción:

- Arquitectura en pequeño: se interesa por la arquitectura de programas individuales.
- Arquitectura en grande: se interesa por la arquitectura de sistemas empresariales complejos que incluyen otros sistemas, programas y componentes de programa.

_ La arquitectura de software es importante porque afecta el desempeño y la potencia, así como la capacidad de distribución y mantenimiento de un sistema.

Decisiones en el diseño arquitectónico

_ El diseño arquitectónico es un proceso creativo en el cual se diseña una organización del sistema que cubrirá los requerimientos funcionales y no funcionales de éste. Puesto que se trata de un proceso creativo, las actividades dentro del proceso dependen del tipo de sistema que se va a desarrollar, los antecedentes y la experiencia del arquitecto del sistema, así como de los requerimientos específicos del sistema. Por lo tanto, es útil pensar en el diseño arquitectónico como un conjunto de decisiones a tomar en vez de una secuencia de actividades.

_ Durante el proceso de diseño arquitectónico, los arquitectos del sistema deben tomar algunas decisiones estructurales que afectarán profundamente el sistema y su proceso de desarrollo. Con base en su conocimiento y experiencia, deben considerar las siguientes preguntas fundamentales sobre el sistema:

- ¿Existe alguna arquitectura de aplicación genérica que actúe como plantilla para el sistema que se está diseñando?
- ¿Cómo se distribuirá el sistema a través de algunos núcleos o procesadores?
- ¿Qué patrones o estilos arquitectónicos pueden usarse?
- ¿Cuál será el enfoque fundamental usado para estructurar el sistema?
- ¿Cómo los componentes estructurales en el sistema se separarán en subcomponentes?
- ¿Qué estrategia se usará para controlar la operación de los componentes en el sistema?
- ¿Cuál organización arquitectónica es mejor para entregar los requerimientos no funcionales del sistema?
- ¿Cómo se evaluará el diseño arquitectónico?
- ¿Cómo se documentará la arquitectura del sistema?

_ Aunque cada sistema de software es único, los sistemas en el mismo dominio de aplicación tienen normalmente arquitecturas similares que reflejan los conceptos fundamentales del dominio.

_ Debido a la estrecha relación entre los requerimientos no funcionales y la arquitectura de software, el estilo y la estructura arquitectónicos particulares que se elijan para un sistema dependerán de los requerimientos de sistema no funcionales:

Rendimiento: si el rendimiento es un requerimiento crítico, la arquitectura debe diseñarse para localizar operaciones críticas dentro de un pequeño número de componentes, con todos estos componentes desplegados en la misma computadora en vez de distribuirlos por la red. Esto significaría usar algunos componentes relativamente grandes, en vez de pequeños componentes de grano fino, lo cual reduce el número de comunicaciones entre componentes. También puede considerar organizaciones del sistema en tiempo de operación que permitan a este ser replicable y ejecutable en diferentes procesadores.

Seguridad: si la seguridad es un requerimiento crítico, será necesario usar una estructura en capas para la arquitectura, con los activos más críticos protegidos en las capas más internas, y con un alto nivel de validación de seguridad aplicado a dichas capas.

Protección: si la protección es un requerimiento crítico, la arquitectura debe diseñarse de modo que las operaciones relacionadas con la protección se ubiquen en algún componente individual o en un pequeño número de componentes. Esto reduce los costos y problemas de validación de la protección, y hace posible ofrecer sistemas de protección relacionados que, en caso de falla, desactiven con seguridad el sistema.

Disponibilidad: si la disponibilidad es un requerimiento crítico, la arquitectura tiene que diseñarse para incluir componentes redundantes de manera que sea posible sustituir y actualizar componentes sin detener el sistema. En el capítulo 13 se describen dos arquitecturas de sistema tolerantes a fallas en sistemas de alta disponibilidad.

Mantenibilidad: si la mantenibilidad es un requerimiento crítico, la arquitectura del sistema debe diseñarse usando componentes autocontenidos de grano fino que puedan cambiarse con facilidad. Los productores de datos tienen que separarse de los consumidores y hay que evitar compartir las estructuras de datos.

Vistas arquitectónicas

_ Es imposible representar toda la información relevante sobre la arquitectura de un sistema en un solo modelo arquitectónico, ya que cada uno presenta únicamente una vista o perspectiva del sistema. Ésta puede mostrar cómo un sistema se descompone en módulos, cómo interactúan los procesos de tiempo de operación o las diferentes formas en que los componentes del sistema se distribuyen a través de una red. Todo ello es útil en diferentes momentos de manera que, para el diseño y la documentación, por lo general se necesita presentar múltiples vistas de la arquitectura de software. Se sugiere que deben existir cuatro vistas arquitectónicas fundamentales, que se relacionan usando casos de uso o escenarios. Las vistas que él sugiere son:

Vista lógica: que indique las abstracciones clave en el sistema como objetos o clases de objeto. En este tipo de vista se tienen que relacionar los requerimientos del sistema con entidades.

Vista de proceso: que muestre cómo, en el tiempo de operación, el sistema está compuesto de procesos en interacción. Esta vista es útil para hacer juicios acerca de las características no funcionales del sistema, como el rendimiento y la disponibilidad.

Vista de desarrollo: que muestre cómo el software está descompuesto para su desarrollo, esto es, indica la descomposición del software en elementos que se implementen mediante un solo desarrollador o equipo de desarrollo. Esta vista es útil para administradores y programadores de software.

Vista física: que exponga el hardware del sistema y cómo los componentes de software se distribuyen a través de los procesadores en el sistema. Esta vista es útil para los ingenieros de sistemas que planean una implementación de sistema.

Patrones arquitectónicos

_ La idea de los patrones como una forma de presentar, compartir y reutilizar el conocimiento sobre los sistemas de software se usa ahora ampliamente. Un patrón arquitectónico se puede considerar como una descripción abstracta estilizada de buena práctica, que se ensayó y puso a prueba en diferentes sistemas y entornos. De este modo, un patrón arquitectónico debe describir una organización de sistema que ha tenido éxito en sistemas previos.

Modelo vista controlador

_ Este separa presentación e interacción de los datos del sistema. El sistema se estructura en tres componentes lógicos que interactúan entre sí y estos son:

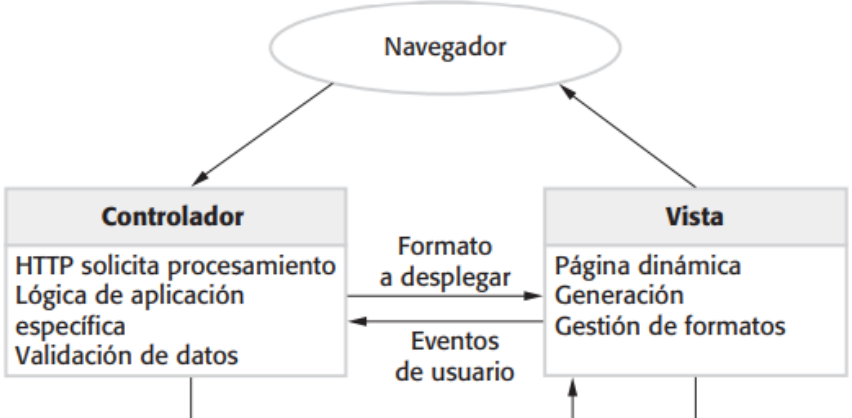
Modelo: maneja los datos del sistema y las operaciones asociadas a esos datos.

Vista: define y gestiona cómo se presentan los datos al usuario.

Controlador: dirige la interacción del usuario (por ejemplo, teclas oprimidas, clics del mouse, etcétera) y pasa estas interacciones a Vista y Modelo.

_ Este patrón se usa cuando existen múltiples formas de ver e interactuar con los datos. También se utiliza al desconocerse los requerimientos futuros para la interacción y presentación.

_ Tomando las ventajas, permite que los datos cambien de manera independiente de su representación y viceversa, soporta en diferentes formas la presentación de los mismos datos, y los cambios en una representación se muestran en todos ellos. Y en cuanto a las desventajas tenemos que puede implicar código adicional y complejidad de código cuando el modelo de datos y las interacciones son simples.



Arquitectura en capas

_ Esta arquitectura organiza el sistema en capas con funcionalidad relacionada con cada capa. Una capa da servicios a la capa de encima, de modo que las capas de nivel inferior representan servicios núcleo que es probable se utilicen a lo largo de todo el sistema. Esta se usa al construirse nuevas facilidades encima de los sistemas existentes; cuando el desarrollo se dispersa a través de varios equipos de trabajo, y cada uno es responsable de una capa de funcionalidad; cuando exista un requerimiento para seguridad multinivel.

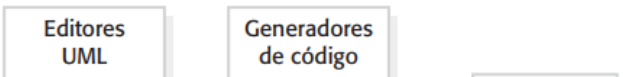
_ Como ventajas tenemos que permite la sustitución de capas completas en tanto se conserve la interfaz, para aumentar la confiabilidad del sistema, en cada capa pueden incluirse facilidades redundantes (por ejemplo, autenticación). Y en cuanto a desventajas, en la práctica, suele ser difícil ofrecer una separación limpia entre capas, y es posible que una capa de nivel superior deba interactuar directamente con capas de nivel inferior, en vez de que sea a través de la capa inmediatamente abajo de ella, el rendimiento suele ser un problema, debido a múltiples niveles de interpretación de una solicitud de servicio mientras se procesa en cada capa.

Arquitectura de repositorio

_ En esta arquitectura todos los datos en un sistema se gestionan en un repositorio central, accesible a todos los componentes del sistema. Los componentes no interactúan directamente, sino tan sólo a través del repositorio. Cada herramienta de software genera información que, en ese momento, está disponible para uso de otras herramientas.

_ Este patrón se usa cuando se tiene un sistema donde los grandes volúmenes de información generados deban almacenarse durante mucho tiempo. También puede usarse en sistemas dirigidos por datos, en los que la inclusión de datos en el repositorio active una acción o herramienta.

_ Como ventajas tenemos que los componentes pueden ser independientes, no necesitan conocer la existencia de otros componentes, los cambios hechos por un componente se pueden propagar hacia todos los componentes, la totalidad de datos se puede gestionar de manera consistente (por ejemplo, respaldos realizados al mismo tiempo), pues todos están en un lugar. Y respecto a las desventajas, el repositorio es un punto de falla único, de modo que los problemas en el repositorio afectan a todo el sistema, es posible que haya ineficiencias al organizar toda la comunicación a través



del repositorio, quizá sea difícil distribuir el repositorio por medio de varias computadoras, se interesa por la estructura estática de un sistema sin mostrar su organización en tiempo de operación.

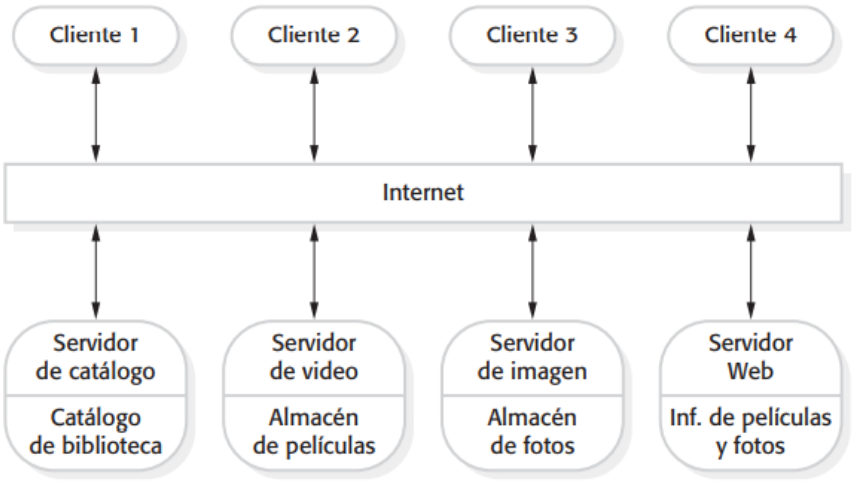
Arquitectura en cliente-servidor

_ En una arquitectura cliente-servidor, la funcionalidad del sistema se organiza en servicios, y cada servicio lo entrega un servidor independiente. Los clientes son usuarios de dichos servicios y para utilizarlos ingresan a los servidores. Esta arquitectura se usa cuando, desde varias ubicaciones, se tiene que ingresar a los datos en una base de datos compartida. Como los servidores se pueden replicar, también se usan cuando la carga de un sistema es variable.

_ La principal ventaja de este modelo es que los servidores se pueden distribuir a través de una red, la funcionalidad general (por ejemplo, un servicio de impresión) estaría disponible a todos los clientes, así que no necesita implementarse en todos los servicios. Como desventajas tenemos que cada servicio es un solo punto de falla, de modo que es susceptible a ataques de rechazo de servicio o a fallas del servidor, el rendimiento resultará impredecible porque depende de la red, así como del sistema. Quizás haya problemas administrativos cuando los servidores sean propiedad de diferentes organizaciones.

_ Un sistema que sigue el patrón cliente-servidor se organiza como un conjunto de servicios y servidores asociados, y de clientes que acceden y usan los servicios. Las arquitecturas cliente-servidor se consideran a menudo como arquitecturas de sistemas distribuidos. Los principales componentes de este modelo son:

- Un conjunto de servidores que ofrecen servicios a otros componentes.
- Un conjunto de clientes que solicitan los servicios que ofrecen los servidores.
- Una red que permite a los clientes acceder a dichos servicios.



Puntos claves

Arquitectura de software: es una descripción de cómo se organiza un sistema de software. Las propiedades de un sistema, como rendimiento, seguridad y disponibilidad, están influidas por la arquitectura utilizada.

Decisiones de diseño arquitectónico: incluyen decisiones sobre el tipo de aplicación, la distribución del sistema, los estilos arquitectónicos a usar y las formas en que la arquitectura debe documentarse y evaluarse.

Documentación de arquitecturas: las arquitecturas pueden documentarse desde varias perspectivas o diferentes vistas. Las posibles vistas incluyen la conceptual, la lógica, la de proceso, la de desarrollo y la física.

Patrones arquitectónicos: son medios para reutilizar el conocimiento sobre las arquitecturas de sistemas genéricos. Describen la arquitectura, explican cuándo debe usarse, y exponen sus ventajas y desventajas.

Tipos de patrones: los patrones arquitectónicos usados comúnmente incluyen el modelo de vista del controlador, arquitectura en capas, repositorio, cliente-servidor, y tubería y filtro.

Modelos genéricos de las arquitecturas de sistemas de aplicación: ayudan a entender la operación de las aplicaciones, comparar aplicaciones del mismo tipo, validar diseños del sistema de aplicación y valorar componentes para reutilización a gran escala.

Sistemas de procesamiento de transacción: son sistemas interactivos que permiten el acceso y la modificación remota de la información, en una base de datos por parte de varios usuarios. Los sistemas de información y los sistemas de gestión de recursos son ejemplos de sistemas de procesamiento de transacciones.

Sistemas de procesamiento de lenguaje: se usan para traducir textos de un lenguaje a otro y para realizar las instrucciones especificadas en el lenguaje de entrada. Incluyen un traductor y una máquina abstracta que ejecuta el lenguaje generado.

Diseño e implementación

Introducción

_ El diseño y la implementación del software es la etapa del proceso de ingeniería de software en que se desarrolla un sistema de software ejecutable.

- Para algunos sistemas simples, el diseño y la implementación del software es ingeniería de software, y todas las demás actividades se fusionan con este proceso.
- Para sistemas grandes, el diseño y la implementación del software son sólo uno de una serie de procesos (ingeniería de requerimientos, verificación y validación, etcétera) implicados en la ingeniería de software.

_ Las actividades de diseño e implementación de software se encuentran invariablemente entrelazadas. El diseño de software es una actividad creativa donde se identifican los componentes del software y sus relaciones, con base en los requerimientos de un cliente. La implementación es el proceso de realizar el diseño como un programa. Algunas veces, hay una etapa de diseño separada y este último se modela y documenta.

Construir o comprar

_ Cuando llegamos a la etapa de diseño e implementación lo único que se ha hecho es superar la etapa de requerimientos, es decir, llegamos a esta etapa teniendo en claro cuáles son las necesidades del producto. Ahora cuando llegamos a este momento, uno puede tomar la decisión de construir un producto nuevo o bien utilizar un producto que ya existe. Cuando hablamos de comprar uno lo podría pensar como comprar algo que ya está desarrollado con los mismos objetivos, entonces antes de decidir en comprar un sistema nuevo es importante analizar si es factible utilizar un producto que ya existe o bien un producto de características generales que puede ser configurable y que pueda cubrir todos los requerimientos que se tenga, esta también es una decisión de índole económica.

_ Cuando decidimos utilizar un producto que ya existe o de propósito general que puede ser configurado, el tiempo que vamos a necesitar para poner en marcha el proyecto es mucho menor ya que el producto va a estar disponible más rápidamente. Pero si se decide construir un producto particular para esa problemática, es muy probable que ese producto se adapte mejor a las necesidades de la organización ya que fue desarrollado exclusivamente para eso, aunque seguramente va a ser más caro y el tiempo de desarrollo va a ser mucho mayor.

Diseño orientado a objetos con el uso del UML

_ Un sistema orientado a objetos se constituye con objetos que interactúan y mantienen su propio estado local y ofrecen operaciones sobre dicho estado. La representación del estado es privada y no se puede acceder directamente desde afuera del objeto. Los procesos de diseño orientado a objetos implican el diseño de clases de objetos y las relaciones entre dichas clases; tales clases definen tanto los objetos en el sistema como sus interacciones. Cuando el diseño se realiza como un programa en ejecución, los objetos se crean dinámicamente a partir de estas definiciones de clase.

_ Los sistemas orientados a objetos son más fáciles de cambiar que aquellos sistemas desarrollados usando enfoques funcionales. Los objetos incluyen datos y operaciones para manipular dichos datos. En consecuencia, pueden entenderse y modificarse como entidades independientes. Cambiar la implementación de un objeto o agregar servicios no afectará a otros objetos del sistema. Puesto que los objetos se asocian con cosas, con frecuencia hay un mapeo claro entre entidades del mundo real (como componentes de hardware) y sus objetos controladores en el sistema. Esto mejora la comprensibilidad y, por ende, la mantenibilidad del diseño.

Etapas del proceso

_ Para desarrollar un diseño de sistema desde el concepto hasta el diseño detallado orientado a objetos, hay muchas cuestiones por hacer:

Contexto e interacciones del sistema: la primera etapa en cualquier proceso de diseño de software es desarrollar la comprensión de las relaciones entre el software que se diseñará y su ambiente externo. Esto es esencial para decidir cómo proporcionar la funcionalidad requerida del sistema y cómo estructurar el sistema para que se comunique con su entorno. La comprensión del contexto permite también determinar las fronteras del sistema. El establecimiento de las fronteras del sistema ayuda a decidir sobre las características que se implementarán en el sistema que se va a diseñar, así como sobre las de otros sistemas asociados.

- Modelo de contexto del sistema: es un modelo estructural, que muestra los otros sistemas en el entorno del sistema a desarrollar. Se utiliza un diagrama de clases.
- Modelo de interacción: es un modelo dinámico que indica la forma en que el sistema interactúa con su entorno conforme se utiliza. Se utiliza un diagrama de casos de uso.

Diseño arquitectónico: una vez definidas las interacciones entre el sistema de software y el entorno del sistema, se aplica esta información como base para diseñar la arquitectura del sistema. Se identifican los principales componentes que conforman el sistema y sus interacciones, y luego puede organizar los componentes utilizando un patrón arquitectónico.

Identificación de clase de objeto: en esta etapa del proceso de diseño, es necesario tener algunas ideas sobre los objetos esenciales en el sistema que se diseña. Conforme aumente su comprensión del diseño, se corregirán estas ideas de los objetos del sistema. La descripción del caso de uso ayuda a identificar objetos y operaciones del sistema. Se basa en la habilidad, la experiencia y el conocimiento del dominio de los diseñadores de sistemas. La Identificación de objetos es un proceso iterativo. Es poco probable hacerlo bien la primera vez. Hay varias propuestas sobre cómo identificar las clases de objetos en los sistemas orientados a objetos:

- Análisis gramatical de una descripción en lenguaje natural del sistema a construir.

- Entidades tangibles (cosas) en el dominio de aplicación.
- Análisis basado en escenarios, donde a la vez se identifiquen y analicen varios escenarios de uso de sistema.

Modelos de diseño: muestran los objetos o clases de objetos en un sistema y también indican las asociaciones y relaciones entre tales entidades. Dichos modelos son el puente entre los requerimientos y la implementación de un sistema. Deben ser abstractos, de manera que el detalle innecesario no oculte las relaciones entre ellos y los requerimientos del sistema. Sin embargo, deben incluir suficiente detalle para que los programadores tomen decisiones de implementación. Por lo general se desarrollará dos tipos de modelo de diseño:

- Modelos estáticos o estructurales: describen la estructura estática del sistema usando las clases de objetos y sus relaciones (las relaciones importantes que pueden documentarse en esta etapa son las relaciones de generalización (herencia)).
- Modelos dinámicos: explican la estructura dinámica del sistema y muestran las interacciones entre los objetos del sistema.

Especificación de interfaz: es necesario especificar las interfaces de modo que los objetos y subsistemas puedan diseñarse en paralelo. Una vez especificada la interfaz, los desarrolladores de otros objetos pueden suponer que se implementará la interfaz. El diseño de interfaz se preocupa por la especificación del detalle de la interfaz hacia un objeto o un grupo de objetos.

Patrones de diseño

_ En la etapa de diseño es importante tener en cuenta los patrones de diseño.

Patrón de diseño: es una forma de reutilizar el conocimiento abstracto sobre un problema y su solución. Es una descripción del problema y la esencia de su solución. Debe ser lo suficientemente abstracto para ser reutilizados en diferentes entornos. En las descripciones de patrón, suelen hacer uso de las características orientadas a objetos como la herencia y el polimorfismo. Los elementos del mismo son:

- Nombre.
- Un identificador de patrón significativo.
- Descripción del problema.
- Descripción de la solución.
- No es un diseño concreto, pero es una plantilla para una solución de diseño que pueden ser instanciado de diferentes maneras.
- Consecuencias.
- Los resultados y las ventajas y desventajas de aplicar el patrón.

Reutilización

_ Este es otro tema importante. Lo que se pueda reutilizar debe ser reutilizado, y cuando se usa reutilización hay menos código para probar porque como ya se probó y

utilizo antes, no tiene sentido probar de nuevo, pero lo que si tiene sentido es probar la interfaz de ese elemento en el nuevo entorno. Si trabajamos orientado a objetos vamos a maximizar la reutilización. Pero cuando hablamos de reutilización no hablamos solamente de código, sino que hablamos de muchos tipos de reutilización, también tenemos que poner en la balanza si el preparar cosas para la reutilización genera un retraso en un proyecto generando también un costo mayor, hay que evaluar si vale la pena ese tiempo. Muchas veces es factible que se puedan reutilizar cosas, pero esas cosas deban ser derivaciones de herencia y demás. Debemos evaluar si es más costoso más allá de la arquitectura.

Niveles de reutilización: hay un montón de cosas que se pueden reutilizar:

- Nivel de abstracción: en este nivel, no se emplea el software directamente, pero se utiliza el conocimiento de las abstracciones exitosas en el diseño de su software. El proceso de abstracción consiste en entender cuáles son las cosas que importan y cuales las cosas que no influyen para la solución del problema.
- Nivel del objeto: en este nivel, vuelve a utilizar directamente los objetos de una biblioteca en lugar de escribir el código manualmente. Implementamos un objeto.
- Nivel de los componentes: los componentes son colecciones de objetos y clases de objetos que se reutilizan en los sistemas de aplicación. Pensamos en una funcionalidad que se resuelve instanciando un objeto de cada una de las clases, y haciéndolos trabajar entre si generando una funcionalidad en el sistema, por ejemplo, un componente de autenticación.
- Nivel de sistema: en este nivel, se vuelve a utilizar los sistemas de aplicación enteros.

Costos de reutilización: hay que considerar:

- Los costos del tiempo invertido en busca de software para la reutilización y la evaluación de si es o no ajustable a sus necesidades.
- Los costos de la compra de software reutilizable.
- Los costos de adaptación y configuración de los componentes o sistemas de software reutilizables para reflejar los requisitos del sistema que se está desarrollando.
- Los costos de la integración de los elementos de software reutilizables entre sí (si está utilizando software de fuentes diferentes) y con el nuevo código que se ha desarrollado.

Gestión o administración de la configuración

_ Todo este proceso de ida y vuelta de tratar de diseñar, encontrar las clases, etc, en algún momento vamos a entrar en el proceso de implementación en sí, y este proceso tiene sus propias cuestiones. Un buen diseño arquitectónico nos va a facilitar después con todo lo que tenga que ver con una implementación. Es necesario también tener ambientes de desarrollo (repositorios) que sean adecuados para facilitar la forma en la

que se va a llevar adelante la implementación y esto nos va permitir gestionar versiones, entre otras cosas.

Gestión de la configuración: es el nombre que recibe el proceso general de la gestión de un sistema de software cambiante. El objetivo de la gestión de la configuración es apoyar el proceso de integración de sistemas para que todos los desarrolladores puedan acceder al código y a la documentación del proyecto de una manera controlada, averiguar qué cambios se han hecho, y compilar y enlazar componentes para crear un sistema.

_ Cuando hablamos de configuración hablamos no solo de un software de propósito general, pero bastante amplio, donde uno después lo puede configurar y parametrizar para usarlo en una aplicación en particular.

Actividades de gestión de configuración

_ Otra cosa importante también durante la etapa de implementación es tener herramientas de seguimiento de problemas, es decir, tener herramientas que nos permitan definir si se van encontrando backs, cuáles son sus problemas, que se puedan recortar, y que se pueda hacer el seguimiento.

Gestión de versiones: para realizar un seguimiento de las diferentes versiones de los componentes de software. Los sistemas de gestión de la versión incluyen facilidades para coordinar el desarrollo de varios programadores.

Integración de sistemas: proporciona apoyo a los desarrolladores a definir qué versiones de los componentes se utilizan para crear cada versión de un sistema. Esta descripción se utiliza entonces para construir un sistema de forma automática mediante la compilación y la vinculación a los componentes necesarios.

Seguimiento de problemas: brinda apoyo para que los usuarios reporten errores y problemas. Los desarrolladores ven quién está trabajando en estos problemas y cuando están resueltos.

Desarrollo de código abierto

_ Uno puede hacer desarrollo de código abierto, con lo cual estamos permitiendo que el código sea posteriormente utilizado y sea modificado por otras personas. No hay normas ni estándares, pero todo lo que un usuario modifique debería ser abierto.

Desarrollo de código abierto: es un enfoque para el desarrollo de software en el que se publica el código fuente de un sistema de software y se invita a los voluntarios a participar en el proceso de desarrollo. El software de código abierto extendió esta idea a través de Internet para reclutar a una población mucho más grande de desarrolladores voluntarios.

- Un principio fundamental de desarrollo de código abierto es que el código fuente debe estar disponible gratuitamente. Legalmente, el desarrollador del código (ya sea una empresa o un individuo) todavía posee el código y puede

imponer restricciones sobre cómo se utiliza mediante una licencia de software de código abierto. Algunos desarrolladores creen que, si un componente de código abierto se utiliza para desarrollar un nuevo sistema, ese sistema debe también ser de código abierto. Otros permiten que su código sea usado sin esta restricción y los sistemas desarrollados pueden ser como sistemas de código cerrado.

Puntos clave

Diseño e implementación del software: son actividades entrelazadas. El nivel de detalle en el diseño depende del tipo de sistema a desarrollar y de si se usa un enfoque dirigido por un plan o uno ágil.

Procesos del diseño orientado a objetos: incluyen actividades para diseñar la arquitectura del sistema, identificar objetos en el sistema, describir el diseño mediante diferentes modelos de objeto y documentar las interfaces de componente.

Proceso de diseño orientado a objetos: durante este puede elaborarse una variedad de modelos diferentes. En ellos se incluyen:

- Modelos estáticos (modelos de clase, modelos de generalización, modelos de asociación).
- Modelos dinámicos (modelos de secuencia, modelos de máquina de estado).

Interfaces de componente: estas deben definirse con precisión, de modo que otros objetos puedan usarlos. Para definir interfaces es posible usar un estereotipo de interfaz UML.

_ Cuando se desarrolla software, siempre debe considerarse la posibilidad de reutilizar el software existente, ya sea como componentes, servicios o sistemas completos.

Administración de la configuración: es el proceso de gestionar los cambios a un sistema de software en evolución. Es esencial cuando un equipo de personas coopera para desarrollar software. La mayoría del desarrollo de software es desarrollo huésped-objetivo. Se usa un IDE en una máquina para desarrollar el huésped, que se transfiere a una máquina objetivo para su ejecución.

Desarrollo de código abierto: este requiere hacer públicamente disponible el código fuente de un sistema. Esto significa que muchos individuos tienen la posibilidad de proponer cambios y mejoras al software.

Pruebas de software

Introducción

_ Las pruebas intentan demostrar que un programa hace lo que se intenta que haga, así como descubrir defectos en el programa antes de usarlo. Al probar el software, se ejecuta un programa con datos artificiales. Hay que verificar los resultados de la prueba que se opera para buscar errores, anomalías o información de atributos no funcionales del programa. El proceso de prueba tiene dos metas distintas:

- Pruebas de validación: demostrar al desarrollador y al cliente que el software cumple con los requerimientos.
- Pruebas de defectos: encontrar situaciones donde el comportamiento del software sea incorrecto, indeseable o no esté de acuerdo con su especificación. Tales situaciones son consecuencia de defectos del software.

Verificación vs validación

_ No podemos hablar de pruebas solamente, ya que esta etapa es un proceso más complejo que se llama verificación y validación, y esta es la verdadera etapa del proceso de desarrollo de software.

Verificación: es ver si construimos el producto correcto, es decir, si funciona bien y si está de acuerdo con los requerimientos tanto funcionales como no funcionales, es decir, de acuerdo a nuestra especificación de requerimientos, probar que lo que construimos cumple con esos requerimientos. Nos fijamos si construimos bien el producto, si el software cumple con los requerimientos funcionales y no funcionales establecidos.

Validación: es la que nos va a decir si además de que funcione, es lo que el cliente necesita. Esto es más profundo, en donde vemos que además de que se cumpla con los requerimientos especificados, es lo que el usuario deseaba. Es importante que hagamos esta primero antes de la etapa de testing ya que, si esta no está bien, el producto no es el solicitado. Nos fijamos si construimos el producto correcto, si el software debe hacer lo que el usuario realmente necesita. Apunta a ver que el análisis de requerimientos se hizo bien.

Inspecciones de software

_ Inspección significa ir y leer lo que tenemos y se puede hacer a todo nivel y por eso es importante. Se enfocan en el código fuente de un sistema con el objetivo de descubrir anomalías y defectos. No requieren la ejecución de un sistema así que puede ser utilizado antes de la implementación. Se pueden aplicar a cualquier representación del sistema (requisitos, diseño, datos de configuración, datos de prueba, etc.). Han demostrado ser una técnica efectiva para descubrir errores de programa.

Ventajas de las inspecciones:

- Durante las pruebas, los errores pueden enmascarar (ocultar) otros errores, pero durante la prueba sale que esta todo bien. La inspección es un proceso estático, no hay interacciones entre los errores.

- Versiones incompletas de un sistema pueden ser inspeccionadas fácilmente. En las pruebas es necesario desarrollar casos de prueba específicos para las partes desarrolladas.
- La inspección permite refactorizar mejorando la calidad del programa (mejora el cumplimiento de las normas, la portabilidad y facilidad de mantenimiento).

Pruebas de desarrollo o integración

_ Las pruebas de desarrollo incluyen todas las actividades de prueba que realiza el equipo que elabora el sistema. Estas se definen como un mecanismo de testeo de software, donde se realiza un análisis de los procesos relacionados con el ensamblaje o unión de los componentes, sus comportamientos con múltiples partes del sistema (ya sea de archivos operativos) o de hardware, entre otras. Durante el desarrollo, las pruebas se realizan en tres niveles de granulación:

Pruebas de unidad o unitarias: donde se ponen a prueba unidades de programa o clases de objetos individuales. Las pruebas de unidad deben enfocarse en comprobar la funcionalidad de objetos o métodos.

Pruebas del componente: donde muchas unidades individuales se integran para crear componentes compuestos. La prueba de componentes debe enfocarse en probar interfaces del componente.

Pruebas del sistema: donde algunos o todos los componentes en un sistema se integran y el sistema se prueba como un todo. Las pruebas del sistema deben enfocarse en poner a prueba las interacciones de los componentes. Hace referencia a muchos componentes interactuando entre sí (se van apilando componentes).

_ Las pruebas de desarrollo son, ante todo, un proceso de prueba de defecto, en las cuales la meta consiste en descubrir bugs en el software. Por lo tanto, a menudo están entrelazadas con la depuración que es el proceso de localizar problemas con el código y cambiar el programa para corregirlos. Estas pruebas representan pruebas de caja blanca, en donde pueden definirse como una técnica de monitorización o prueba de software en la se analiza el diseño, código y estructura interna, con el objetivo de mejorar propiedades como la seguridad y el uso eficiente del sistema.

Desarrollo dirigido por pruebas

_ El desarrollo dirigido por pruebas (TDD, por las siglas de Test-Driven Development) es un enfoque al diseño de programas donde se entrelazan el desarrollo de pruebas y el de código. En esencia, el código se desarrolla incrementalmente, junto con una prueba para ese incremento. No se avanza hacia el siguiente incremento sino hasta que el código diseñado pasa la prueba. Un argumento consistente con el desarrollo dirigido por pruebas es que ayuda a los programadores a aclarar sus ideas acerca de lo que realmente debe hacer un segmento de código. Para escribir una prueba, es preciso

entender lo que se quiere, pues esta comprensión facilita la escritura del código requerido. Además de la mejor comprensión del problema, otros beneficios del desarrollo dirigido por pruebas son:

Cobertura de código: en principio, cualquier segmento de código que escriba debe tener al menos una prueba asociada. Por lo tanto, puede estar seguro de que cualquier código en el sistema se ejecuta realmente. El código se prueba a medida que se escribe, de modo que los defectos se descubren con oportunidad en el proceso de desarrollo.

Pruebas de regresión: siempre es posible correr pruebas de regresión para demostrar que los cambios al programa no introdujeron nuevos bugs. Estas implican correr los conjuntos de pruebas ejecutadas exitosamente después de realizar cambios a un sistema. La prueba de regresión verifica que dichos cambios no hayan introducido nuevos bugs en el sistema o en otras partes del mismo, y que el nuevo código interactúa como se esperaba con el código existente. Estas pruebas son muy costosas y, por lo general, poco prácticas.

- Uno de los beneficios más importantes del desarrollo dirigido por pruebas es que reduce los costos de las pruebas de regresión.

Depuración simplificada: cuando falla una prueba, debe ser evidente dónde yace el problema. Es preciso comprobar y modificar el código recién escrito. No se requieren herramientas de depuración para localizar el problema.

Documentación del sistema: las pruebas en sí actúan como una forma de documentación que describen lo que debe hacer el código. Leer las pruebas suele facilitar la comprensión del código.

Pruebas de versión

_ Las pruebas de versión son el proceso de poner a prueba una versión particular de un sistema que se pretende usar fuera del equipo de desarrollo. Por lo general, la versión del sistema es para clientes y usuarios. Existen dos distinciones importantes entre las pruebas de versión y las pruebas del sistema durante el proceso de desarrollo:

- Un equipo independiente que no intervino en el desarrollo del sistema debe ser el responsable de las pruebas de versión.
- Las pruebas del sistema por parte del equipo de desarrollo deben enfocarse en el descubrimiento de bugs en el sistema (pruebas de defecto). El objetivo de las pruebas de versión es comprobar que el sistema cumpla con los requerimientos y sea suficientemente bueno para uso externo (pruebas de validación).

_ La principal meta del proceso de pruebas de versión es convencer al proveedor del sistema de que éste es suficientemente apto para su uso. Si es así, puede liberarse como un producto o entregarse al cliente. Por lo tanto, las pruebas de versión deben

mostrar que el sistema entrega su funcionalidad, rendimiento y confiabilidad especificados, y que no falla durante el uso normal.

_ Las pruebas de versión, por lo regular, son un proceso de prueba de caja negra, donde las pruebas se derivan a partir de la especificación del sistema. El sistema se trata como una caja negra cuyo comportamiento sólo puede determinarse por el estudio de entradas y salidas relacionadas.

Pruebas basadas en requerimientos: es importante que los requerimientos deban ser comprobables, es decir, los requerimientos tienen que escribirse de forma que pueda diseñarse una prueba para dicho requerimiento. Luego, un examinador comprueba que el requerimiento se cumpla. Entonces, las pruebas basadas en requerimientos son un enfoque sistemático al diseño de casos de prueba, donde se considera cada requerimiento y se deriva un conjunto de pruebas para éste. Estas pruebas son pruebas de validación más que de defecto, ya que se intenta demostrar que el sistema implementó adecuadamente sus requerimientos.

Pruebas de escenario: estas pruebas son un enfoque a las pruebas de versión donde se crean escenarios típicos de uso y se les utiliza en el desarrollo de casos de prueba para el sistema.

- Un escenario es una historia que describe una forma en que puede usarse el sistema. Los escenarios deben ser realistas, y los usuarios reales del sistema tienen que relacionarse con ellos.

Pruebas de rendimiento: una vez integrado completamente el sistema, es posible probar propiedades emergentes, como el rendimiento y la confiabilidad. Las pruebas de rendimiento deben diseñarse para garantizar que el sistema procese su carga pretendida, y esto implica efectuar una serie de pruebas donde se aumenta la carga, hasta que el rendimiento del sistema se vuelve inaceptable. Como con otros tipos de pruebas, las pruebas de rendimiento se preocupan tanto por demostrar que el sistema cumple con sus requerimientos, como por descubrir problemas y defectos en el sistema. Este tipo de pruebas tiene dos funciones:

- Prueba el comportamiento de falla del sistema: circunstancias a través de una combinación inesperada de eventos donde la carga colocada en el sistema supere la carga máxima anticipada.
- Forzar al sistema y puede hacer que salgan a la luz defectos que no se descubrirían normalmente.

Pruebas de usuario

_ Las pruebas de usuario o del cliente son una etapa en el proceso de pruebas donde los usuarios o clientes proporcionan entrada y asesoría sobre las pruebas del sistema. Esto puede implicar probar de manera formal un sistema que se comisionó a un proveedor externo, o podría ser un proceso informal donde los usuarios experimentan con un nuevo producto de software, para ver si les gusta y si hace lo que necesitan. Las pruebas de usuario son esenciales, aun cuando se hayan realizado pruebas

abarcadoras de sistema y de versión. La razón de esto es que la influencia del entorno de trabajo del usuario tiene un gran efecto sobre la fiabilidad, el rendimiento, el uso y la robustez de un sistema. En la práctica, hay tres diferentes tipos de pruebas de usuario:

Pruebas alfa: donde los usuarios del software trabajan con el equipo de diseño para probar el software en el sitio del desarrollador. En las pruebas alfa, los usuarios y desarrolladores trabajan en conjunto para probar un sistema a medida que se desarrolla, y esto significa que los usuarios pueden identificar problemas y conflictos que no son fácilmente aparentes para el equipo de prueba de desarrollo.

Pruebas beta: donde una versión del software se pone a disposición de los usuarios, para permitirles experimentar y descubrir problemas que encuentran con los desarrolladores del sistema. Las pruebas beta se usan sobre todo para productos de software que se emplean en entornos múltiples y diferentes (en oposición a los sistemas personalizados, que se utilizan por lo general en un entorno definido).

Pruebas de aceptación: donde los clientes prueban un sistema para decidir si está o no listo para ser aceptado por los desarrolladores del sistema y desplegado en el entorno del cliente. Las pruebas de aceptación son una parte inherente del desarrollo de sistemas personalizados. Tienen lugar después de las pruebas de versión. Implican a un cliente que prueba de manera formal un sistema, para decidir si debe o no aceptarlo del desarrollador del sistema.

Puntos clave

Pruebas: estas sólo pueden mostrar la presencia de errores en un programa. Si embargo, no pueden garantizar que no surjan fallas posteriores.

Pruebas de desarrollo: estas son responsabilidad del equipo de desarrollo del software. Un equipo independiente debe responsabilizarse de probar un sistema antes de darlo a conocer a los clientes. En el proceso de pruebas de usuario, clientes o usuarios del sistema brindan datos de prueba y verifican que las pruebas sean exitosas. Las pruebas de desarrollo incluyen:

- Pruebas de unidad: donde se examinan objetos y métodos individuales.
- Pruebas de componente: donde se estudian grupos de objetos relacionados.
- Pruebas del sistema: donde se analizan sistemas parciales o completos.

_ Cuando pruebe software, debe tratar de “romperlo” mediante la experiencia y los lineamientos que elijan los tipos de casos de prueba que hayan sido efectivos para descubrir defectos en otros sistemas.

Pruebas automatizadas: siempre que sea posible, se deben escribir estas pruebas. Las pruebas se incrustan en un programa que puede correrse cada vez que se hace un cambio al sistema.

Desarrollo de la primera prueba: es un enfoque de desarrollo, donde las pruebas se escriben antes de que se pruebe el código. Se realizan pequeños cambios en el código, y éste se refactoriza hasta que todas las pruebas se ejecuten exitosamente.

Pruebas de escenario: son útiles porque imitan el uso práctico del sistema. Implican trazar un escenario de uso típico y utilizarlo para derivar casos de prueba.

Pruebas de aceptación: son un proceso de prueba de usuario, donde la meta es decidir si el software es suficientemente adecuado para desplegarse y utilizarse en su entorno operacional.

Evolución y mantenimiento del software

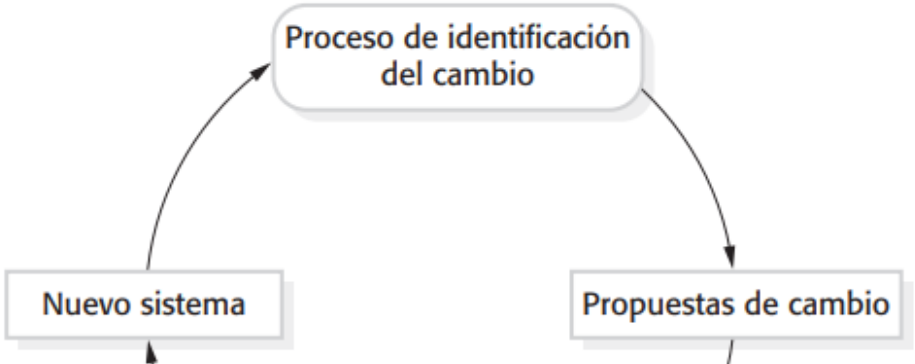
Procesos de evolución

_ Los procesos de evolución del software varían dependiendo del tipo de software que se mantiene, de los procesos de desarrollo usados en la organización y de las habilidades de las personas que intervienen. En algunas organizaciones, la evolución es un proceso informal, donde las solicitudes de cambios provienen sobre todo de conversaciones entre los usuarios del sistema y los desarrolladores. En otras compañías, se trata de un proceso formalizado con documentación estructurada generada en cada etapa del proceso. Las propuestas de cambio al sistema son el motor para la evolución del sistema en todas las organizaciones. Estos cambios provienen de:

- Requerimientos existentes que no se hayan implementado en el sistema liberado.
- Peticiones de nuevos requerimientos.
- Reportes de bugs de los participantes del sistema.
- Nuevas ideas para la mejora del software por parte del equipo de desarrollo del sistema.

Evolución del software: una sola organización es responsable tanto del desarrollo inicial del software como de su evolución posterior.

_ Uno tiene que imaginarse que esto es un proceso cíclico, en el cual lo primero que se hace es identificar la necesidad de cambio y a partir de ahí se definen cuáles son los cambios necesarios y se hacen las propuestas para ese cambio, luego se realizan esos cambios, o sea se hace el proceso de analizar el impacto del cambio, de hacer una especificación, verificar y validar, y después de eso ya se obtiene un nuevo sistema, que es básicamente el sistema anterior pero con incrementos que corresponden a los cambios que se hicieron. Los procesos de identificación de cambios y evolución del sistema son cíclicos y continúan a lo largo de la vida de un sistema.



_ Este es un proceso o actividad normal, lo que pasa es que necesitamos alguna metodología para poderla manejar, y eso depende de que va a haber muchos cambios simultáneamente y entonces no es factible poder responder a esos cambios de manera inmediata y todos juntos. Entonces, el hecho de entrar en este proceso de evolución, tiene también una etapa de análisis de ver cuáles son los cambios necesarios, de qué manera vamos a organizar esos cambios y de qué manera pueden tener prioridades esos cambios los unos sobre los otros. Obviamente las prioridades responden a la reparación de errores que se puedan producir, independientemente de cuál sea la fuente de ese error.

Leyes de Lehman

_ Estas son un conjunto de leyes empíricas desarrolladas por Lehman y Belady relacionadas con la evolución del software pero que, en esencia, pueden ser aplicadas a cualquier "cosa" hardware-software (digamos que podrían ser ampliadas). Son leyes que hacen referencia al cambio continuo.

Cambio continuo: un programa usado en un entorno real debe cambiar; de otro modo, en dicho entorno se volvería progresivamente inútil.

Complejidad creciente: a medida que cambia un programa en evolución, su estructura tiende a volverse más compleja. Deben dedicarse recursos adicionales para conservar y simplificar su estructura.

Evolución de programa grande: la evolución del programa es un proceso autorregulador. Los atributos del sistema, como tamaño, tiempo entre versiones y número de errores reportados, son casi invariantes para cada versión del sistema.

Estabilidad organizacional: durante la vida de un programa, su tasa de desarrollo es aproximadamente constante e independiente de los recursos dedicados al desarrollo del sistema.

Conservación de familiaridad: a lo largo de la existencia de un sistema, el cambio incremental en cada liberación es casi constante.

Crecimiento continuo: la funcionalidad ofrecida por los sistemas tiene que aumentar continuamente para mantener la satisfacción del usuario.

Declive de calidad: la calidad de los sistemas declinará, a menos que se modifiquen para reflejar los cambios en su entorno operacional.

Sistema de retroalimentación: los procesos de evolución incorporan sistemas de retroalimentación multiagente y multiciclo. Además, deben tratarse como sistemas de retroalimentación para lograr una mejora significativa del producto.

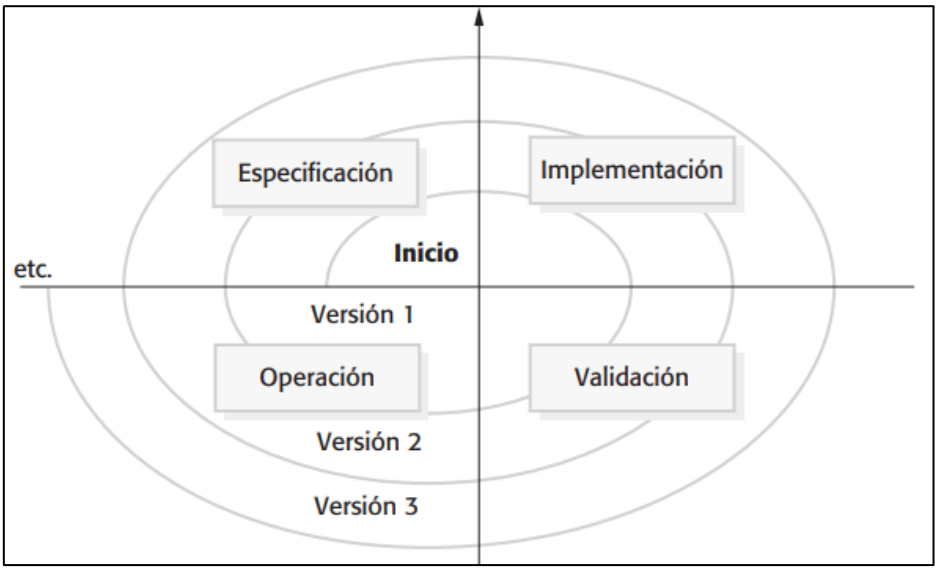
Mantenimiento del software

Mantenimiento del software: es el proceso general de cambiar un sistema después de que éste se entregó. El término usualmente se aplica a software personalizado, en el que grupos de desarrollo separados intervienen antes y después de la entrega. Los cambios realizados al software van desde los simples para corregir errores de codificación, los más extensos para corregir errores de diseño, hasta mejorías significativas para corregir errores de especificación o incorporar nuevos requerimientos. Los cambios se implementan modificando los componentes del sistema existentes y agregándole nuevos componentes donde sea necesario.

_ Entonces decimos que el mantenimiento es el proceso de modificar un producto de software después de su puesta en producción, y este tiene distintos orígenes o disparadores:

- Corregir un mal funcionamiento, este es el principal y el más crítico.
- Mejorar atributos de calidad.
- Adaptarlo a los cambios del entorno.
- Adaptarlo a los cambios empresariales.

_ Esta es una etapa normal y corresponde dentro del proceso de desarrollo del software porque realmente este proceso de desarrollo es un proceso en espiral y más cuando estamos pensando en un desarrollo a nivel iterativo, por lo que siempre vamos a tener lo siguiente:



Costos de mantenimiento

_ En cuanto a porcentaje de costo de mantenimiento según diferentes situaciones tenemos:

- Reparación de fallas de desarrollo: 17%
- Adaptación ambiental: 18%
- Adición o modificación de funcionalidad: 65%

_ En general podemos decir que resulta más costoso agregar funcionalidad después de que un sistema está en operación porque tenemos que hacer un análisis de impacto mucho mayor, que implementar la misma funcionalidad durante el desarrollo ya que será una iteración más. Pero en realidad esto es consecuencia de distintas cosas:

- Estabilidad del equipo.
- Calidad de desarrollo deficiente.
- Habilidades del personal.
- Antigüedad y estructura del programa.

_ Siempre es más difícil tratar de agregar funcionalidades a un producto que ya está funcionando, que si esas funcionalidades se incorporan en el momento en el cual se está haciendo el desarrollo.

Tipos de mantenimiento

Mantenimiento preventivo: se realiza para facilitar las posteriores correcciones, adaptaciones y mejoras. Este no es algo que se hace por una necesidad o para resolver un problema, sino que es un cambio que se hace y no tiene ningún impacto en el producto a nivel de generar funcionalidades nuevas, y en general es uno de los cambios que no son necesitados por el usuario final, sino que en general los realiza el equipo de desarrollo por decisión propia, y que tiene como objetivo mejorar el producto sin que esa mejora se haga visible en una mejora de funcionalidad pero para poder después facilitar correcciones o adaptaciones que si sean necesarias.

- Esto tiene que ver generalmente con refactorización, es decir, mejorar cosas que no se ven y facilitar un mantenimiento posterior.
- Modificar el software identificando componentes reusables.
- Comprobación de la validez de los datos de entrada.

Mantenimiento correctivo: se refiere al mantenimiento para reparación de fallas de desarrollo, es decir, tiene como objetivo corregir defectos.

- De procesamiento: salidas incorrectas.
- De rendimiento: tiempo de respuesta alto en una búsqueda.

- De programación: inconsistencias en el diseño de un programa.
- De documentación: inconsistencias entre la funcionalidad de un programa y el manual de usuario.

Mantenimiento adaptativo: algunas veces quiere decir adaptarse a un nuevo entorno y otras veces significa adaptar el software a nuevos requerimientos. Adaptar el software a las modificaciones del entorno. Y también hay quienes dicen que agregar nuevos cambios o una nueva funcionalidad es un cambio adaptativo ya que nos estamos adaptando a nuevas cosas que necesita la organización.

- Cambios del sistema operativo.
- Cambios o actualizaciones del gestor de bases de datos.
- Cambios de la configuración de hardware.
- Cambios en la organización.
- Cambios legislativos.

Mantenimiento perfectivo: a veces significa perfeccionar el software al implementar nuevos requerimientos; en otros casos representa mantener la funcionalidad del sistema, pero mejorando su estructura y rendimiento. Tiene como objetivo extender los requerimientos funcionales o realizar mejoras de calidad. Podemos confundirlo con el preventivo, porque mejorar el código, o con refactorizarlo también, porque mejora la calidad del producto, pero la diferencia es que en el preventivo no es el cliente el que solicita el cambio sino que es una decisión fundamentalmente del equipo de desarrollo, y el perfectivo también puede ser una decisión del equipo de desarrollo pero en general como consecuencia de algo que se ve en la calidad del producto, por ejemplo que los usuarios se quejen de que el producto anda lento, etc. Todo lo que tienda a mejorar la calidad.

- Agregado de una funcionalidad no contemplada.
- Mejora de la usabilidad del sistema.
- Mejora de la velocidad de respuesta optimizando algoritmos.

Actividades del mantenimiento de software

_ Lo que suele suceder con el mantenimiento es que en general se da menos importancia al mantenimiento que al proceso de desarrollo. Entonces para evitar eso es necesario que el proceso de mantenimiento también respete una serie de actividades. Las actividades de mantenimiento se agrupan en tres categorías funcionales:

- Comprensión del software y de los cambios a realizar (Comprender): es necesario el conocimiento a fondo de la funcionalidad, objetivos, estructura interna y requisitos del software. Alrededor del 50% de tiempo de mantenimiento se dedica a esta actividad.
- Modificación del software (Corregir): crear y modificar las estructuras de datos, la lógica de procesos, las interfaces y la documentación. Los programadores

deben evitar los efectos laterales provocados por sus cambios. Esta actividad representa ¼ del tiempo total de mantenimiento.
- Realización de pruebas (Comprobar): realizar pruebas selectivas que nos aseguren la corrección del software.

_ La etapa de mantenimiento es la más larga del proceso de software y es la que va a durar hasta que el producto deje de estar en funcionamiento o se dé, de baja, y necesitamos poder realizar ese mantenimiento de una manera más fluida, entonces es necesario que cada uno de los cambios respete cada una de estas etapas.

Puntos clave

_ El desarrollo y la evolución del software pueden considerarse como un proceso integrado e iterativo que se representa usando un modelo en espiral.

Sistemas personalizados: para estos, por lo general, los costos del mantenimiento de software superan a los de desarrollo.

Proceso de evolución del software: es conducido por peticiones de cambios e incluye análisis del impacto del cambio, planeación de las versiones e implementación del cambio.

Leyes de Lehman: como la noción de que el cambio es continuo, describen algunas percepciones derivadas de estudios a largo plazo de la evolución del sistema.

_ Existen tres tipos de mantenimiento de software:

- Reparación de bugs.
- Modificación del software para trabajar en un nuevo entorno.
- Implementación de requerimientos nuevos o diferentes.

Reingeniería de software: esta trata la reestructuración y la redocumentación de software para hacerlo más fácil de entender y cambiar.

Refactorización: hace pequeños cambios de programa sin alterar su funcionalidad, podría considerarse como mantenimiento preventivo.

_ El valor empresarial de un sistema heredado y la calidad del software de aplicación y su entorno deben valorarse para determinar si el sistema tiene que sustituirse, transformarse o mantenerse.

Gestión de proyectos

Introducción

_ La gestión de proyectos de software es una parte esencial de la ingeniería de software. La buena gestión no puede garantizar el éxito del proyecto. Sin embargo, la mala gestión, por lo general, da como resultado una falla del proyecto: el software puede entregarse tarde, costar más de lo estimado originalmente o no cumplir las expectativas de los clientes. Desde luego, los criterios de éxito para la gestión del proyecto varían de un proyecto a otro, pero, para la mayoría de los proyectos, las metas importantes son:

- Entregar el software al cliente en el tiempo acordado.
- Mantener costos dentro del presupuesto general.
- Entregar software que cumpla con las expectativas del cliente.
- Mantener un equipo de desarrollo óptimo y con buen funcionamiento.

_ La mayoría de los administradores, en alguna etapa, toman la responsabilidad de varias o todas las siguientes actividades:

Planeación del proyecto: los administradores de proyecto son responsables de la planeación, estimación y calendarización del desarrollo del proyecto, así como de la asignación de tareas a las personas. Supervisan el trabajo para verificar que se realice de acuerdo con los estándares requeridos y monitorizan el avance para comprobar que el desarrollo esté a tiempo y dentro del presupuesto.

Informes: los administradores de proyectos por lo común son responsables de informar del avance de un proyecto a los clientes y administradores de la compañía que desarrolla el software. Deben ser capaces de comunicarse en varios niveles, desde codificar información técnica detallada hasta elaborar resúmenes administrativos. Deben redactar documentos concisos y coherentes que sinteticen información crítica de reportes detallados del proyecto. Es necesario que esta información se presente durante las revisiones de avance.

Gestión del riesgo: los administradores de proyecto tienen que valorar los riesgos que pueden afectar un proyecto, monitorizar dichos riesgos y emprender acciones cuando surjan problemas.

Gestión de personal: los administradores de proyecto son responsables de administrar un equipo de personas. Deben elegir a los integrantes de sus equipos y establecer formas de trabajar que conduzcan a desempeño efectivo del equipo.

Redactar propuestas: la primera etapa en un proyecto de software puede implicar escribir una propuesta para obtener un contrato de trabajo. La propuesta describe los objetivos del proyecto y cómo se realizará. Por lo general, incluye estimaciones de costo y calendarización, además de justificar por qué el contrato del proyecto debería concederse a una organización o un equipo particular. La escritura de propuestas es una tarea esencial, pues la supervivencia de muchas compañías de software depende de contar con suficientes propuestas aceptadas y concesiones de contratos. Es posible que no haya lineamientos establecidos para esta tarea; la escritura de propuestas es una habilidad que se adquiere a través de práctica y experiencia.

Proyecto

Proyecto: es un esfuerzo temporal que se lleva a cabo para crear un producto, servicio o resultado único. Un proyecto es solamente eso, no podemos entender un proyecto como una actividad o una tarea ordinaria, un proyecto es un esfuerzo temporal, donde este tiene un inicio y un fin, o sea tiene un periodo determinado de tiempo en el cual tiene que ser realizado.

_ El proyecto tiene un objetivo único que es crear una cosa nueva, puede ser un producto, servicio, una definición de algo, es decir, hasta una cuestión teórica, pero tiene como meta la obtención de un resultado. Entonces como es temporal tiene un inicio y un final bien definido. El inicio es cuando decidimos comenzar con ese proyecto y el final en principio seria cuando se logran los objetivos, pero también podría haber un final del proyecto en el que se ve que los objetivos no se pueden cumplir porque se ve que no es factible llegar a cumplir esos objetivos o bien porque llega un momento en el cual ese proyecto ya no tiene más sentido o no sirve, entonces se abandona.

_ Todo proyecto implica la ejecución de un conjunto de actividades, y lo principal es que esas actividades tienen que estar coordinadas y controladas para que en vistas de lograr un nuevo objetivo, y que para lograr ese objetivo con esas actividades van a hacer falta una cierta cantidad de recursos, donde estos recursos van a ser personas que trabajan en el proyecto y además ese proyecto va a estar sujeto a restricciones que tienen que ver con tiempo, costos y los recursos que hagan falta, que pueden ser las personas que trabajan, la tecnología y los materiales que haga falta.

Gestión de proyectos

_ Es importante la gestión de proyectos porque un proyecto implica un conjunto de actividades, este conjunto hace uso de recursos que son limitados, tiene establecido un inicio y fin definido, tiene como objetivo obtener un resultado único, en donde este resultado tiene un conjunto de personas que son los stakeholders o los interesados que necesitan ese resultado, y para lograr eso deberíamos realizar una ejecución o elaboración gradual de una serie de actividades durante un determinado tiempo.

_ La gestión de los proyectos, sean de software o de otro tipo va a asegurar que todo esto sea posible, entonces el papel de quien hace la gestión del proyecto va más allá de la gestión del proceso de desarrollo, va a apuntar a la gestión del proyecto en sí, es decir, de qué manera se coordinan las distintas actividades o se tiene control sobre los recursos, tiempos o costos para lograr justamente que este esfuerzo temporal logre ese resultado único.

_ Todo proyecto requiere una administración o gestión, y la verdad es que todo proyecto puede tener como resultado un fracaso o éxito, el hecho de que haya una gestión o administración, no significa que necesariamente el resultado sea exitoso, pero la falta de administración o gestión seguramente llevara a un fracaso del proyecto que nos hemos planteado.

Éxito: entendemos por éxito cuando el proyecto se logra terminarlo logrando los objetivos, manteniendo el presupuesto dentro de lo que se había planeado y los recursos también dentro de lo que se había planeado, y además se realizó en el tiempo esperado.

Fracaso: y entendemos por fracaso cuando un proyecto no se termina, o bien cuando el presupuesto se va de control, cuando se van de control los recursos, cuando l tiempo se extiende, teniendo en cuenta también cuando no se logran los objetivos. Se dice también que fracaso también es cuando se logran los objetivos, pero no se logran dentro del presupuesto que estaba esperado, pero esto es relativo.

_ Quien hace la dirección del proyecto básicamente se supone que aplica ciertos conocimientos, habilidades, herramientas y técnicas para poder manejar las actividades necesarias y cumplir con los objetivos que tenga el proyecto. Cualquier cosa que no sea rutinaria es un proyecto, en cualquier actividad.

_ Se habla de la triple restricción, en donde estas son las que van a definir el fracaso o el éxito. Estas son:

- Alcance: lograr los objetivos.
- Costo: mantener los costos dentro de lo que estaba previsto.
- Tiempo: lograr el resultado dentro de los tiempos esperados.

_ Luego podemos considerar:

- Calidad
- Recursos
- Riesgos
- Satisfacción del cliente

_ Los criterios de éxito para la gestión del proyecto son los siguientes:

- Entregar el software al cliente en el tiempo acordado.
- Mantener costos dentro del presupuesto general.
- Entregar software que cumpla con las expectativas del cliente.
- Mantener un equipo de desarrollo óptimo y con buen funcionamiento.

_ Decíamos que un proyecto tiene un principio y un fin, y si lo graficamos en función del tiempo uno diría que empieza en algún momento y en algún momento termina, y por eso decimos que ese proyecto va a ir pasando por distintas fases (inicial, intermedia, final) y quien haga la gestión va a tener que definir en cada una de esas fases que tareas se hacen, como se hacen y quien las hace, y si pensamos en cual es el nivel de recurso, tanto materiales como de personas, en cada una de las fases va a haber distintas necesidades de recursos.

Acta de constitución: los recursos incrementan en la fase inicial porque el proyecto va a iniciar con una persona que es la que empieza el trabajo y hace la gestión, es decir, pensamos a ver si el proyecto es factible o no, y si lo es, vemos que recursos vamos a necesitar y a medida que se desarrolle seguro va a necesitar más recursos, y después

cuando se va concluyendo cada vez se va a ir necesitando menos recursos (materiales o humanos) hasta que el producto se concluya. Se formaliza el proyecto y se define su propósito.

Gestión del riesgo

_ La gestión del riesgo es una de las tareas más sustanciales para un administrador de proyecto. La gestión del riesgo implica anticipar riesgos que pudieran alterar el calendario del proyecto o la calidad del software a entregar, y posteriormente tomar acciones para evitar dichos riesgos. Podemos considerar un riesgo como algo que es preferible que no ocurra. Los riesgos pueden amenazar el proyecto, el software que se desarrolla o a la organización. Por lo tanto, existen tres categorías relacionadas de riesgo:

Riesgos del proyecto: son riesgos que alteran el calendario o los recursos del proyecto, por ejemplo, la renuncia de un diseñador experimentado en donde encontrar un diseñador de reemplazo con habilidades y experiencia adecuadas puede demorar mucho tiempo y, en consecuencia, el diseño del software tardará más tiempo en completarse, también la rotación de personal, cambios administrativos, indisponibilidad de software,

Riesgos del producto: afectan la calidad o el rendimiento del software a desarrollar, por ejemplo, la falla que presenta un componente que se adquirió al no desempeñarse como se esperaba, en donde esto puede afectar el rendimiento global del sistema, de modo que es más lento de lo previsto, también podemos considerar cambios de requerimientos, demoras en la especificación, subestimación de tamaño, etc, aunque estas últimas tres también pueden ser de proyecto.

Riesgos empresariales: afectan a la organización que desarrolla o adquiere el software, por ejemplo, un competidor que introduce un nuevo producto. Por ejemplo, cambios tecnológicos, competencia de productos, etc.

_ La gestión del riesgo es particularmente importante para los proyectos de software, debido a la incertidumbre inherente que enfrentan la mayoría de proyectos. Ésta se deriva de requerimientos vagamente definidos, cambios de requerimientos que obedecen a cambios en las necesidades del cliente, dificultades en estimar el tiempo y los recursos requeridos para el desarrollo de software, o bien, se deriva de diferencias en las habilidades individuales. Es necesario anticipar los riesgos; comprender el efecto de estos riesgos sobre el proyecto, el producto y la empresa; y dar los pasos adecuados para evitar dichos riesgos. Tal vez se necesite diseñar planes de contingencia de manera que, si ocurren los riesgos, se puedan tomar acciones inmediatas de recuperación

_ La gestión comprende varias etapas:

Identificación del riesgo: en donde hay que identificar posibles riesgos para el proyecto, el producto y la empresa. Algunos ejemplos son:

- Riesgos tecnológicos: se derivan de las tecnologías de software o hardware usadas para desarrollar el sistema.
- Riesgos personales: se asocian con las personas en el equipo de desarrollo. Vienen de las personas o el equipo de trabajo.
- Riesgos organizacionales: se derivan del entorno organizacional donde se desarrolla el software. Problemas económicos o de gerencia.
- Riesgos de herramientas: resultan de las herramientas de software y otro software de soporte que se usa para desarrollar el sistema.
- Riesgos de requerimientos: proceden de cambios a los requerimientos del cliente y del proceso de gestionarlos.
- Riesgos de estimación: surgen de las estimaciones administrativas de los recursos requeridos para construir el sistema.

Análisis de riesgos: se debe valorar la probabilidad y las consecuencias de dichos riesgos.

- La probabilidad del riesgo puede valorarse como muy baja (< 10%), baja (del 10 al 25%), moderada (del 25 al 50%), alta (del 50 al 75%) o muy alta (> 75%). Esto es una estimación.
- Los efectos del riesgo pueden estimarse como catastróficos (amenazan la supervivencia del proyecto), graves (causarían grandes demoras), tolerables (demoras dentro de la contingencia permitida) o insignificantes. Esto también es una estimación.

Planeación del riesgo: es indispensable elaborar planes para enfrentar el riesgo, evitarlo o minimizar sus efectos en el proyecto.

- Estrategias de evitación: reducir la probabilidad de que surja el riesgo. Significa hacer que ese riesgo directamente no aparezca.
- Estrategias de minimización: reducir el efecto del riesgo. La otra opción es mitigarlo, es decir, si el riesgo se presenta hacer que ese problema sea insignificante o reducirlo.
- Planes de contingencia: hacer frente y tener una estrategia para a ello. Cuando el problema se dio y no tenemos forma de minimizar su efecto, pensamos en como seguimos el paso para solucionarlo o salvar el proyecto.

Monitorización del riesgo: en donde hay que valorar regularmente el riesgo y los planes para atenuarlo, y revisarlos cuando se aprenda más sobre el riesgo. Consiste también en aplicar estrategias de mitigación cuando la posibilidad de los riesgos se vuelva más presente.

_ Para monitorizar los riesgos, quien hace la gestión de proyectos tiene que estar atento a determinadas señales de aviso o indicadores en el proyecto que indican que estamos frente a un problema. La persona tiene que estar muy atenta a esos indicadores de tal forma de estar atenta para aplicar las políticas de mitigación. Una vez que el riesgo ya se present va a ser difícil de controlar. Según el tipo de riesgo tenemos los siguientes riesgos posibles:

- Tecnológico: la base de datos que se usa en el sistema no puede procesar tantas transacciones por segundo como se esperaba, o los componentes de software de reutilización contienen defectos que hacen que no puedan reutilizarse como se planeó, etc.
- Personal: es imposible reclutar personal con las habilidades requeridas, el personal clave está enfermo e indispuesto en momentos críticos, no está disponible la capacitación requerida para el personal, etc.
- De organización: la organización se reestructura de modo que diferentes administraciones son responsables del proyecto, problemas financieros de la organización fuerzan reducciones en el presupuesto del proyecto, etc.
- Herramientas: el código elaborado por las herramientas de generación de código de software es ineficiente, las herramientas de software no pueden trabajar en una forma integrada.
- Requerimientos: se proponen cambios a los requerimientos que demandan mayor trabajo de rediseño, los clientes no entienden las repercusiones de los cambios a los requerimientos, etc.
- Estimación: se subestima el tiempo requerido para desarrollar el software, se subestima la tasa de reparación de defectos, se subestima el tamaño del software, etc.

Gestión de personal

_ La gestión de riesgo es una de las actividades que tiene que desarrollar quien hace la gestión del proyecto, pero también tenemos la gestión del personal, es decir, de la gente que trabaja en el proyecto. No hay duda que para un proyecto de software el principal actor que hay es la gente que trabaja, conocimiento y horas de trabajo de la gente. Entonces es importantísimo que en la gestión del proyecto de software la gestión del personal se haga de manera correcta porque de lo contrario va a haber un ambiente de trabajo complicado, la gente va a estar disconforme, entonces una cosa importante de quien hace la gestión de proyectos es estar muy pendientes de cuidar el personal y la gente que está trabajando, y para eso es importante que se sepan distinguir las habilidades de cada persona y asignarle a cada una responsabilidades y trabajos o actividades acordes a sus habilidades y si es posible a los que les es más útil.

_ Las personas son el principal capital de la empresa. Una incorrecta gestión del personal garantiza la falla del proyecto. Se debe signar a las personas responsabilidades acordes a sus habilidades. Hay cuatro factores que tenemos que tener en cuenta en la gestión del personal:

- Consistencia
- Respeto
- Inclusión
- Honestidad

_ Si el proyecto está fallando por algún motivo las personas involucradas en el mismo lo deben saber, y si está andando bien y se está llegando a tiempo, también, entonces

es importantísima la comunicación, por parte de quien hace la gestión hacia el resto de la gente.

Puntos clave

Gestión de proyectos de software: la buena gestión es esencial si los proyectos de ingeniería de software deben desarrollarse dentro del plazo y el presupuesto establecidos.

Gestión del software: es distinta de otras administraciones de ingeniería. El software es intangible. Los proyectos pueden ser novedosos o innovadores, así que no hay un conjunto de experiencias para orientar su gestión. Los procesos de software no son tan maduros como los procesos de ingeniería tradicionales.

Gestión del riesgo: se reconoce ahora como una de las tareas más importantes de la gestión de un proyecto.

_ La gestión del riesgo implica la identificación y valoración de los grandes riesgos del proyecto para establecer la probabilidad de que ocurran; también supone identificar y valorar las consecuencias para el proyecto si dicho riesgo surge. Debe hacer planes para evitar, gestionar o enfrentar los posibles riesgos.

_ Las personas se sienten motivadas por la interacción con otros individuos, el reconocimiento de la gestión y sus pares, y al recibir oportunidades de desarrollo personal.

Grupos de desarrollo de software: estos deber ser bastante pequeños y cohesivos. Los factores clave que influyen en la efectividad de un grupo son sus integrantes, la forma en que está organizado y la comunicación entre los miembros.

_ Las comunicaciones dentro de un grupo están influidas por factores como el estatus de los miembros del grupo, el tamaño del grupo, la composición por género del grupo, las personalidades y los canales de comunicación disponibles.

Calidad del software

Calidad

_ Cuando se habla de calidad, se habla por un lado de dos cosas:

Aseguramiento de calidad: definición de procesos y estándares que deben conducir a la obtención de productos de alta calidad. Esto viene del concepto de que independientemente de cuál sea el producto que uno tenga que generar y producir, la calidad del producto no se define por ver el producto resultante sino que hay todo un proceso para obtener el producto y es importante para lograr una calidad en el producto resultante que el proceso de construcción sea también un proceso de calidad, y por eso se habla de definir estándares (procesos estandarizados) que siguen conciertas normas de calidad con la idea de que si el proceso de construcción del

producto tiene calidad, es más factible que el producto obtenido sea un producto de calidad.

Control de calidad: aplicación de procesos de calidad para eliminar productos que no cuentan con el nivel de calidad requerido. Este se independiza del proceso anterior y va a mirar directamente al producto, y es lo que normalmente se hace en cualquier línea de producción.

_ Si nosotros tenemos un proceso bien estructurado y controlado en este proceso de producción del software, es más probable que el producto obtenido tenga una buena calidad, pero además de eso, una vez que tenemos el producto, lo tenemos que analizar para ver si cumple las normas de calidad que hemos establecido o no.

Manejo de la calidad del software

_ Entonces en general cuando se habla de software y calidad del mismo, tenemos que tener establecido cual es el nivel o estándar de calidad requerido para nuestro producto. Y para lograr esto tenemos que definir una serie de estándares que definen como son los procedimientos y aseguran que esos procedimientos sean cumplidos de esa manera. Entonces en definitiva lo que busca toda esta cuestión de pensar en la calidad es generar una cultura durante el proceso de desarrollo que tienda a garantizar de que el producto que se obtenga va a ser un producto con cierto nivel de calidad.

_ Se refiere a lograr un nivel de calidad requerido en el producto de software. Involucra a la definición de estándares de calidad apropiados y procedimientos que permitan asegurar que estos se cumplan. Debe llevar a desarrollar una cultura de calidad en donde la calidad es responsabilidad de todos.

Calidad de producto de software

_ Lo primero que uno puede pensar es que un producto de software es un producto de calidad si cumple con los requerimientos que hemos especificado, pero en realidad si el producto no cumple con los requerimientos no tiene un problema de calidad, sino que directamente está mal hecho y no sirve, entonces cuando hablamos de que cumpla con las especificaciones (funcionales), lo que define que sea de menor o mayor calidad es en realidad el cumplimiento de los requerimientos no funcionales.

_ Es difícil definir cuando un producto de software tiene la calidad esperada o no, si no cumple ningún requerimiento funcional no es un problema de calidad, pero si no cumple con los requerimientos no funcionales si tenemos un producto de calidad.

_ Calidad significa que un producto debe cumplir con sus especificaciones. Para sistemas de software:

- Algunos requerimientos de calidad son difíciles de especificar (eficiencia, mantenibilidad, reusabilidad, etc).
- Las especificaciones del Software son usualmente incompletas y a menudo inconsistentes.

El compromiso de calidad

_ No podemos esperar a que las especificaciones mejoren para poner atención al manejo de la calidad. Debe haber procedimientos que permitan mejorar la calidad, aunque las especificaciones no sean perfectas. El manejo de la calidad no solo se refiere a reducir defectos sino también a mejorar otras cualidades del producto

_ Nosotros tenemos que lograr un producto de calidad aun cuando las especificaciones no sean perfectas y por eso es que decimos de que el proceso o el compromiso con la calidad tiene que estar de un primer momento, y por eso se insiste tanto en la etapa de ingeniería de requerimientos de profundizar el conocimiento de los requerimientos, pensando justamente en la calidad del producto final, ya que mientras mejor sea esa especificación mejor definidos van a estar los requerimientos que nos van a permitir lograr un producto con la calidad que nos hace falta.

Actividades de manejo de calidad

_ En realidad, pensando en la calidad, independientemente del producto que uno vaya a desarrollar, uno debería preocuparse por tres cosas:

Aseguramiento de calidad: establecer procedimientos organizacionales y estándares para la calidad. Y esos procedimientos en definitiva tienen que decir cómo se hacen cada una de las cosas que haya que hacer. En general en una organización que tenga como objetivo certificarse en calidad, esa organización debe definir su procedimiento para todo y además hacerlos cumplir, pero no solamente eso sino también las cuestiones contables, contratación del personal, es decir, una empresa que se certifica en calidad tiene definidos sus procedimientos para todas las actividades que tenga.

Planeación de calidad: seleccionar procedimientos aplicables y estándares para un proyecto en particular y modificar estos como sean requeridos. Entonces para cada una de las actividades se va a tener que ver cuáles son los procedimientos que aplican para esa actividad en particular y cuales no aplican. El planear significa seleccionar de los procedimientos que están establecidos, cuales son aplicables y cuáles no.

Control de calidad: garantizar que procedimientos y estándares son seguidos por el equipo de desarrollo de software. De nada sirve tener especificados los procedimientos estándares para cada una de las actividades, si después en la actividad normal de la compañía en realidad se trabaja de otra manera.

_ El manejo de calidad debe ser separado del manejo del proyecto para asegurar independencia.

_ Tiene que haber alguien que controle que efectivamente todos los procesos estándares fueron aplicados o no de manera correcta.

Atributos de la calidad del software

_ Pensando en la calidad del producto de software, tendrían que haber algunos atributos que definan cual es la calidad del producto de software. Hay atributos que se ven y otros que son más difíciles que se vean:

- Protección
- Seguridad
- Fiabilidad
- Flexibilidad
- Robustez
- Comprensibilidad
- Comprobabilidad
- Adaptabilidad
- Modularidad
- Complejidad
- Portabilidad
- Usabilidad
- Reusabilidad
- Eficiencia
- Facilidad para que el usuario aprenda a utilizarlo

_ Muchos de estos tienen que ver con características internas del software como por ejemplo el tema de la seguridad, protección o robustez tienen que ver con cuestiones como la arquitectura que se ha elegido, la mantenibilidad y la comprensibilidad tienen que ver con la estructura del código, la usabilidad puede comprobarse con el testing, la eficiencia va a tener que ver con pruebas de estrés, etc.

_ Entonces, los atributos de calidad de un producto de software son bastantes difíciles de comprobar y no todos intervienen en el mismo producto, y por ende no los vamos a considerar para hacer un análisis de calidad. Cuando hablamos de calidad tenemos que poder determinar cuáles son los atributos importantes y después de que manera esos atributos van a ser evaluados.

_ Estas cosas que vemos nos van a permitir evaluar la calidad de un producto, pero como dijimos, la calidad no se puede centrar en ver la calidad del producto que obtenemos, sino que uno podría pensar que por un lado está la calidad del producto final y por el otro, está la calidad final del proceso que se desarrolla para obtener ese producto final

Calidad basada en procesos

_ Se habla de la calidad basada en procesos, y entonces decimos si un proceso es de calidad, es bastante probable que el producto obtenido sea de calidad. Si el proceso no está controlado y la calidad no interviene entonces podemos probar que el producto lo sea. Aplicar calidad al producto de software puede ser más complejo que aplicar procedimientos de calidad en otros tipos de producción. Cuando nosotros desarrollamos software, en general el desarrollo implica la aplicación de habilidades individuales y de experiencia y entonces es más difícil definir esos procesos, y por otro

lado hay además una falta por parte de los desarrolladores de ajustarse a estos procedimientos estandarizados de calidad. Entonces puede ser más difícil aplicar esta calidad basada en procesos, pero es importante que la organización trate de hacerlo, el problema es que esos procesos tienen que ser adecuados, que en general hay resistencia por parte de los desarrolladores a utilizar procesos estandarizados, muchas veces porque esos procesos no reflejan exactamente el proceso correcto de trabajo que se debería tener, es por eso que se tiene que tener especial cuidado en poner estándares inapropiados porque estos no van a ser utilizados.

_ Relación directa entre procesos y productos. Más complejo para software debido a:

- Se requiere la aplicación de habilidades individuales y experiencia, la cual es importante para el desarrollo de software.
- Factores externos en las que una aplicación es novedosa o la necesidad para acelerar el calendario de desarrollo puede empeorar la calidad del producto

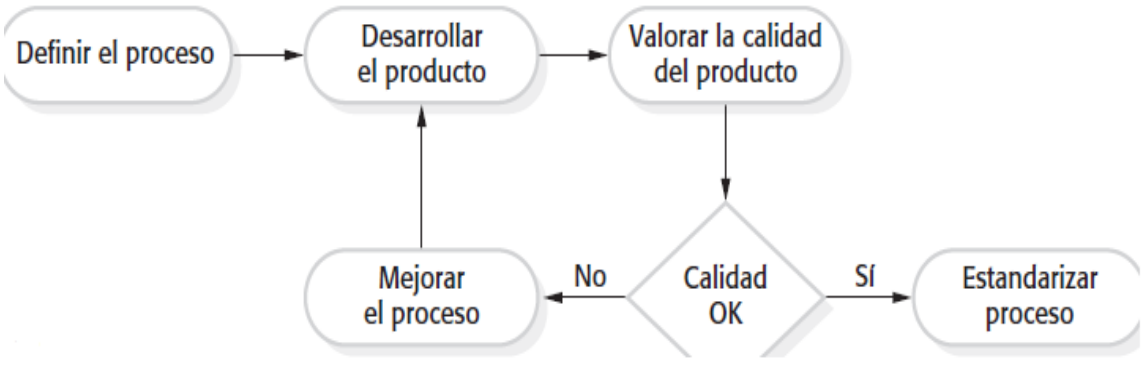
_ Debe tenerse especial cuidado de no imponer estándares inapropiados.

Calidad de procesos practica

_ Cuando trabajamos con la calidad en los procesos, en general, lo que se hace es:

- Definir procesos de estándares que indiquen como llevar a cabo las revisiones, la administración de la configuración, etc. Estos procesos son para todas las etapas del proceso de desarrollo.
- Monitorear el proceso de desarrollo para asegurar que se están siguiendo los estándares. De nada sirve haber definido los estándares si después los procesos no se realizan de esa manera, por eso tiene que haber alguien encargado de monitorear que efectivamente las actividades se hacen siguiendo esos estándares. Se tiene que monitorear que se apliquen y que sean apropiados.
- Reportar estos procesos a la administración del proyecto. No sirve de nada tener un conjunto de estándares y procedimientos que no son conocidos por las personas que los tienen que aplicar.

_ En general, la cuestión es de la siguiente manera, se define el proceso, se hace el desarrollo siguiendo ese proceso que se está definiendo, ahí aparecen problemas que se descubren con ese proceso, entonces se puede mejorar el proceso, luego se ve si el producto que se obtiene tiene una calidad buena, de lo contrario seguiremos mejorando el proceso hasta lograr que la calidad del producto mejore por haber mejorado la calidad del proceso de desarrollo, y cuando eso se pudo determinar, el proceso se puede estandarizar. Y esta es la forma en la cual sen establecen los distintos procesos de desarrollo de software.



Estándares de software

_ Estos estándares son importantes porque si se manejan bien van a permitir mejorar la calidad del producto, y es importante que existan sobre todo cuando tenemos una organización que puede generar proyectos a gran escala, por ejemplo internacional, donde es producto que vamos a obtener, en definitiva tiene que estar de acuerdo con ciertos estándares de calidad que a lo mejor no se aplican en algún lado pero si en otro, entonces es importante para que el producto sea aceptado y pueda ser comercializado, sea desarrollado siguiendo esos estándares que se utilizan en otros países.

_ Por ejemplo en la comunidad europea, y sobre todo los estándares de producción de software, requieren de un montón de condiciones y para que un producto pueda ser comercializado a ese nivel debería certificar que fue desarrollado siguiendo y respetando esos estándares, y también van a depender los estándares que haya que aplicar del producto que se esté trabajando ya que no es lo mismo desarrollar un producto de software de la gestión de un quiosco que un producto de software de hospital, etc. Entonces va a haber distintas exigencias en los estándares que nos exijan que cumplan nuestros procesos de desarrollo según el dominio para el cual estemos desarrollando.

_ Por otro lado en distintas organizaciones tiene que haber establecido los estándares del producto, es decir, nosotros producimos algo y esto tiene que cumplir con determinados estándares o normas establecidas, que también pueden ser propias de la organización o pueden respetar estándares o legislaciones nacionales y/o internacionales. Y eso va a definir estándares del producto en su conjunto, pero también podría exigir estándares de distintos componentes como por ejemplo un componente de seguridad con determinados estándares que debe cumplir.

_ Los estándares son muy importantes porque permiten estandarizar la calidad y definir las exigencias mínimas que vamos a tener ya sea para un proceso o un producto.

_ Son clave para un efectivo manejo de calidad. Puede ser un proyecto internacional, nacional, u organizacional. Los estándares de producto definen características que todos los componentes deberán tener.

Importancia de los estándares

_ Estándar significa algo que se utiliza siempre de la misma manera. La idea es que cuando uno establezca un estándar, se supone que se establece la mejor practica y ese estándar no debería tener errores, entonces sí sabemos que a una determinada actividad la realizamos de alguna manera y en realidad no era la forma correcta, obviamente eso no va a ser párate de un estándar, al contrario, el estándar va a definir que eso no se debe hacer de ninguna manera incorrecta. Y teniendo definido los estándares tenemos un marco de referencia, es decir, vamos a confrontar los interrogantes que se nos aparezcan, y además si establecemos esos estándares, cada persona que entre a la organización le decimos que tiene que estudiar todos los estándares porque es la forma en la que tiene que trabajar, por lo que es mucho más fácil que las personas que estén ingresando se puedan integrar y saber de qué manera trabaja la organización, y por eso dentro de una organización hay un manual de estándares.

- Reúne las mejores prácticas.
- Evita la repetición de errores pasados.
- Proporciona un marco para el análisis de calidad - involucra verificar la conformidad con estándares.
- Proporcionar continuidad. El personal nuevo puede entender a la organización entendiendo a los estándares aplicados.

Problemas con estándares

_ El hecho de que los estándares se establezcan no significa que se cumplan, y esto es un problema que hay que resolver, ya que no basta con establecer un manual donde están definidos todos los estándares de productos y de procesos y que después nadie lo haga, entonces hay que controlar que esos estándares se cumplen. Pero también hay que saber que hay problemas con los estándares y en general los problemas derivan de que la gente que está en el proceso de producción por ahí no los valora como importante, pero puede ser también que en el procedimiento sean demasiado complicados o que no se ajusten mucho a la realidad, entonces en ese caso los estándares tienen que ser revisados por la gente que los aplica para ver si son correctos o no. En general son bastante burocráticos porque esta todo de por medio, tiene que haber aprobaciones, alguien tiene que firmar y a veces tienen que ser varias personas para que pase de una etapa a otra, entonces como aplica toda esta burocracia puede que las personas se resistan a utilizarlos.

- No son vistos como relevantes ni se encuentran actualizados por los ingenieros de software.
- Involucra muchas formas burocráticas.
- No soportado por herramientas de software por lo que se requieren actividades para mantener los estándares.

Desarrollo de estándares

_ Entonces la idea es que los estándares existan y se deban aplicar, y para eso tendríamos que lograr:

- Involucra a los desarrolladores: los Ingenieros deberán entender la racionalidad bajo un estándar. Tienen que participar de la definición de los estándares con lo cual se logra que entienda porque hacen falta, para que sirven, que es una buena práctica y cuáles son los buenos resultados, y que además participen opinando y definiendo que sería lo más adecuado.
- Revisión de estándares y su uso regularmente: los estándares pueden rápidamente estar desactualizados lo cual reduce su credibilidad entre sus usuarios. Tienen que revisarse porque pueden desactualizarse, porque las herramientas del desarrollo cambian, etc.
- Los estándares detallados deberán tener asociado una herramienta de soporte: si necesitamos hacer un determinado proceso de aprobación, tendría que haber alguna herramienta de gestión que nos facilite el proceso y que la gente acceda a él para saber lo que tiene que hacer.

_ Excesivo trabajo manual es el más significante reclamo en contra de los estándares.

Normas ISO

_ Básicamente son estándares internacionales para el manejo de calidad, y se aplican a cualquier tipo de industrias y hoy por hoy cualquier empresa tiene como objetivo certificarse en calidad, porque se supone que si aplicamos estos estándares de calidad vamos a obtener mejores productos y porque la fama de una empresa que tiene una certificación ISO es distinta a aquella que no la tiene.

_ Conjunto de estándares internacionales para el manejo de calidad. Aplicable a un rango de organizaciones desde industrias de servicio a industrias de manufactura. Tenemos los siguientes dos:

- La ISO 9001 es aplicable a organizaciones del cual diseñan, desarrollan y mantienen productos.
- ISO 9001 es un modelo genérico del proceso de calidad. Está instanciado para cada organización.

Certificación ISO

_ Cuando uno quiere hacer una certificación con normas ISO, no es que la norma dice lo que se tiene que hacer, en realidad lo que dice es que como nosotros somos una organización debemos definir cuáles son los procesos que se tiene para cada una de las actividades de la organización.

- Los Estándares de calidad y procedimientos deberán ser documentados en un manual organizacional de calidad.
- Personal externo puede certificar que una organización conforma con los estándares ISO 9000/9001.
- Los clientes demandan cada vez más que sus desarrolladores tengan la certificación ISO 9000/9001.

Revisión

_ Para validar si estamos cumpliendo con la calidad en un proceso o en un producto, lo que hay que hacer es revisar, entonces habrá un grupo que se va a encargar de esta actividad de autoevaluación viendo la documentación y los problemas que puede haber en la misma, haciendo inspecciones, hacer mejoras y ver si se están aplicando correctamente o no.

_ El principal método de validación de la calidad de un proceso o de un producto Un grupo debe examinar parte o toda su documentación para buscar problemas potenciales. Hay diferentes tipos de revisiones con diferentes objetivos:

- Inspecciones para remover defectos (producto).
- Revisiones para estimación de progresos (procesos y producto).
- Revisiones de calidad (estándares y producto)

Procedimientos de revisión

_ Estas revisiones tienen que ver con:

Función de calidad: es parte del proceso general de administración de calidad.

Función de administración del proyecto: proveen información para los administradores del proyecto.

Funciones de comunicación y entretenimiento: paso de conocimientos entre miembros de desarrollo del equipo.

_ Entonces al hacer esta revisión en los distintos estándares de calidad, no solamente se va a revisar que se hayan definido los estándares para los procesos que estén bien, sino que también se va a revisar que estén bien comunicados, porque es necesario que la gente para que pueda aplicar un procedimiento estándar tiene que conocerlo y por ende tiene que haber comunicación y entrenamiento, y además tenemos en cuenta de qué manera se va a controlar que efectivamente eso se realice. Entonces dentro de un proyecto vemos cómo se va a administrar de manera adecuada la calidad.

Revisiones de calidad

_ Entonces la revisión de calidad va a ser un grupo de personas cuidadosamente examinará cada parte o todo un sistema de software y su documentación asociada

- Código, diseños, especificaciones, planes de prueba, estándares, etc. todo puede ser revisado.
- Documentos o Software puede ser “firmados” en cada revisión lo cual significa que la administración ha aprobado el progreso de la siguiente etapa del desarrollo.

_ Estas revisiones son revisiones internas que se hacen antes de pedir una certificación externa, y se deben hacer de manera periódica porque es importante que ese equipo de revisión sea algo constante y que la gente sepa que estos deben cumplirse y en algún momento se van a controlar.

- El objetivo es descubrir defectos en el sistema e inconsistencias.
- Cualquier documento producido en el proceso puede ser revisado.
- El equipo de revisión deberá ser relativamente pequeño y las revisiones deberán ser relativamente cortas.
- La revisión deberá ser grabada y almacenada

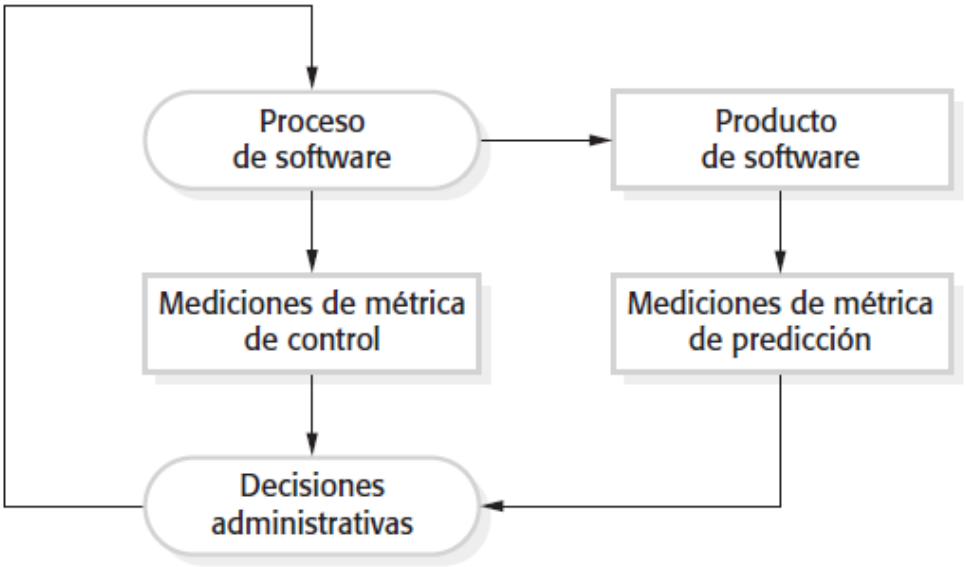
Tipos de revisiones

Review type	**Principal purpose**
Design or program inspections	To detect detailed errors in the design or code and to check whether standards have been followed. The review should be driven by a checklist of possible errors.
Progress reviews	To provide information for management about the overall progress of the project. This is both a process and a product review and is concerned with costs, plans and schedules.
Quality reviews	To carry out a technical analysis of product components or documentation to find faults or mismatches between the specification and the design, code or documentation. It may also be concerned with broader quality issues such as adherence to standards and other quality attributes.

Métricas de la calidad del producto

_ Una métrica de calidad deberá ser una forma de predicción de la calidad del producto. La mayoría de las métricas de calidad existentes son las métricas de la calidad del diseño las cuales se relacionan con la medición del acoplamiento o la complejidad del diseño.

Métricas de control y predicción



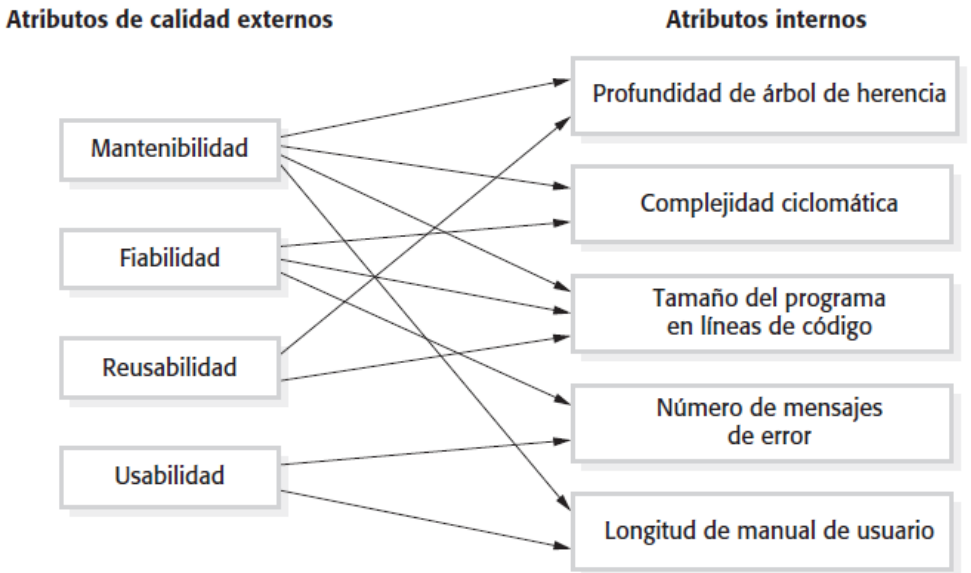
Suposición de métricas

_ Una propiedad del software puede ser medida. Existe una relación entre lo que se puede medir y que se quiere conocer. Esta relación ha sido formalizada y validada. Puede ser difícil relacionar que puede ser medido en cuanto a atributos deseables de calidad.

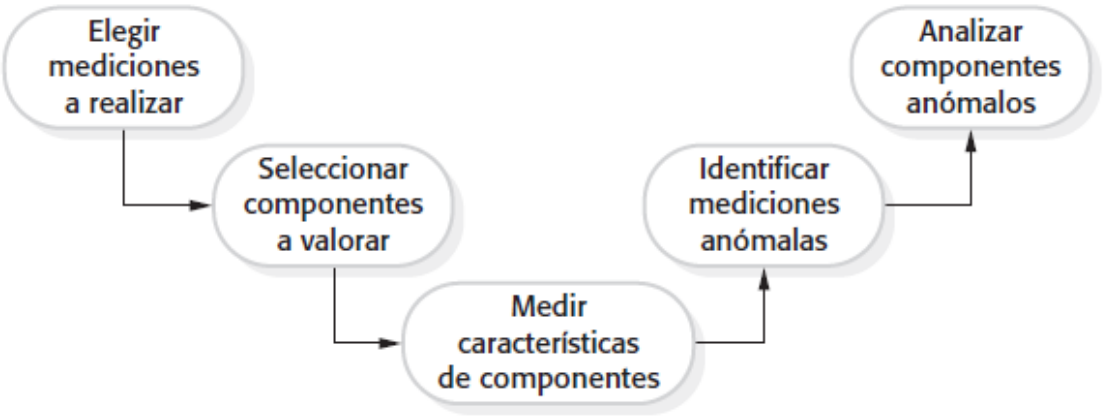
Atributos internos y externos

_ Desde el punto de vista del producto de software, una vez que tenemos el producto, se siguen todos los procesos con lo cual es de esperar que el producto que se obtiene tenga calidad porque estamos certificados y tenemos todos los procesos y los productos intermedios con un montón de condiciones y los cumplimos a todos. Ahora el producto de software que obtenemos puede ser un producto que tenga calidad o no, y para saber esto sacamos muestras de los productos que van apareciendo y a esas muestras les hacemos control de calidad, pero como el software es un producto intangible, básicamente las pruebas que hacemos serían las pruebas de usuario que definen si se acepta o no, pero esto por sí solo no va a definir la calidad del producto, ya que hay otros atributos que apuntan generalmente a cuestiones no generales y que van a ser los que van a definir que un producto, más allá de que respete todas sus cualidades funcionales, es de mala calidad, ya que no respeta algunos de los atributos de calidad externos.

_ Hay aspectos que son externos pero que en realidad se originan en cuestiones internas. La complejidad ciclomática por ejemplo mide la complejidad en ciclos, decisiones, etc, y mientras más alto es el número es porque es más complicado.



Proceso de medición del producto



Mantenibilidad del diseño

Cohesión: cuanto están relacionadas las partes forman un componente.

Acoplamiento: que tan independiente es un componente.

Entendibilidad: que tan fácil es entender las funciones de un componente.

Adaptabilidad: que tan fácil es cambiar un componente.

Métricas de acoplamiento

_ Asociado con la medida 'fan-in y fan-out':

Alta entrada (fan-in): implica un alto acoplamiento debido a las dependencias de los módulos.

Alta salida (fan-out): implica un alto acoplamiento debido a la complejidad del control.

Métricas de calidad en un programa

_ Las métricas de diseño son también aplicables a programas. Otras métricas incluyen

- Longitud: el tamaño del código fuente del programa.
- Complejidad ciclomática: la complejidad de control de un programa.
- Longitud de identificadores.
- Profundidad de condicionales anidados.

_ Los valores anómalos de las métricas sugieren que un componente presenta defectos o es difícil de entender.

Consideraciones para las métricas

_ La longitud del código es simple pero la experimentación ha sugerido que representa un buen predictor de problemas. La complejidad ciclomática puede ser engañosa. Nombres largos deberán incrementar la entendibilidad de un programa. Condicionales profundamente anidadas son difíciles de entender.

Métricas de calidad de la documentación

_ La legibilidad es importante en la documentación. El índice “Gunnings Fog” es una medida de la legibilidad. Basada en la longitud de las frases y el número de sílabas, en una palabra. Esto puede causar malas interpretaciones cuando se aplica a la documentación técnica.

Madures de las métricas

_ Las métricas todavía tienen un valor limitado y no ampliamente aceptado. Las relaciones entre que lo que se puede medir y lo que se quiere conocer no está bien comprendido aún. Hace falta poner de acuerdo a las organizaciones sobre las métricas necesarias en el proceso de software.

Puntos clave

Gestión de calidad del software: se ocupa de garantizar que el software tenga un número menor de defectos y que alcance los estándares requeridos de mantenibilidad, fiabilidad, portabilidad, etcétera. Incluye definir estándares para procesos y productos, y establecer procesos para comprobar que se siguieron dichos estándares.

Estándares de software: son importantes para el aseguramiento de la calidad, pues representan una identificación de las “mejores prácticas”. Al desarrollar el software, los estándares proporcionan un cimiento sólido para diseñar software de buena calidad.

_ Es necesario documentar un conjunto de procedimientos de aseguramiento de la calidad en un manual de calidad organizacional. Esto puede basarse en el modelo genérico para un manual de calidad sugerido en el estándar ISO 9001.

_ Las revisiones de los entregables del proceso de software incluyen a un equipo de personas que verifican que se siguieron los estándares de calidad. Las revisiones son la técnica usada más ampliamente para valorar la calidad.

_ En una inspección de programa o revisión de pares, un reducido equipo comprueba sistemáticamente el código. Ellos leen el código a detalle y buscan posibles errores y omisiones. Entonces los problemas detectados se discuten en una reunión de revisión del código. La medición del software puede usarse para recopilar datos cuantitativos tanto del software como del proceso de software. Se usan los valores de las métricas de software recopilados para hacer inferencias referentes a la calidad del producto y el proceso.

Métricas de calidad del producto: son particularmente útiles para resaltar los componentes anómalos que pudieran tener problemas de calidad. Dichos componentes deben entonces analizarse con más detalle.