

Ingeniería de software 2

Teórico

Alumno: Santiago Vietto

Docente: Natalia Mira

Institución: UCC

Año: 2022

Introducción

_ Las partes más importantes que va a tener la materia van a ser el proceso de desarrollo, el proceso de ingeniería de requerimientos, que ayuda a validar, en esta etapa de requerimientos, que justamente se haga lo que el cliente pidió y que este detallado de alguna manera, cumpliendo con los atributos de calidad que se espera del requerimiento. El requerimiento se plantea en lenguaje natural, donde uno anota lo que entendió de una entrevista, y muchas veces la interpretación que hacemos en esta etapa no es la adecuada, entonces para evitar esto trabajamos con las técnicas de modelado para tener distintas perspectivas y para siempre poder validar que lo que vayamos haciendo en el paso a paso de todo el ciclo de vida del sistema realmente sea correcto y esté de acuerdo a las especificaciones que dio el cliente. Por último, tenemos los patrones de análisis de diseño.

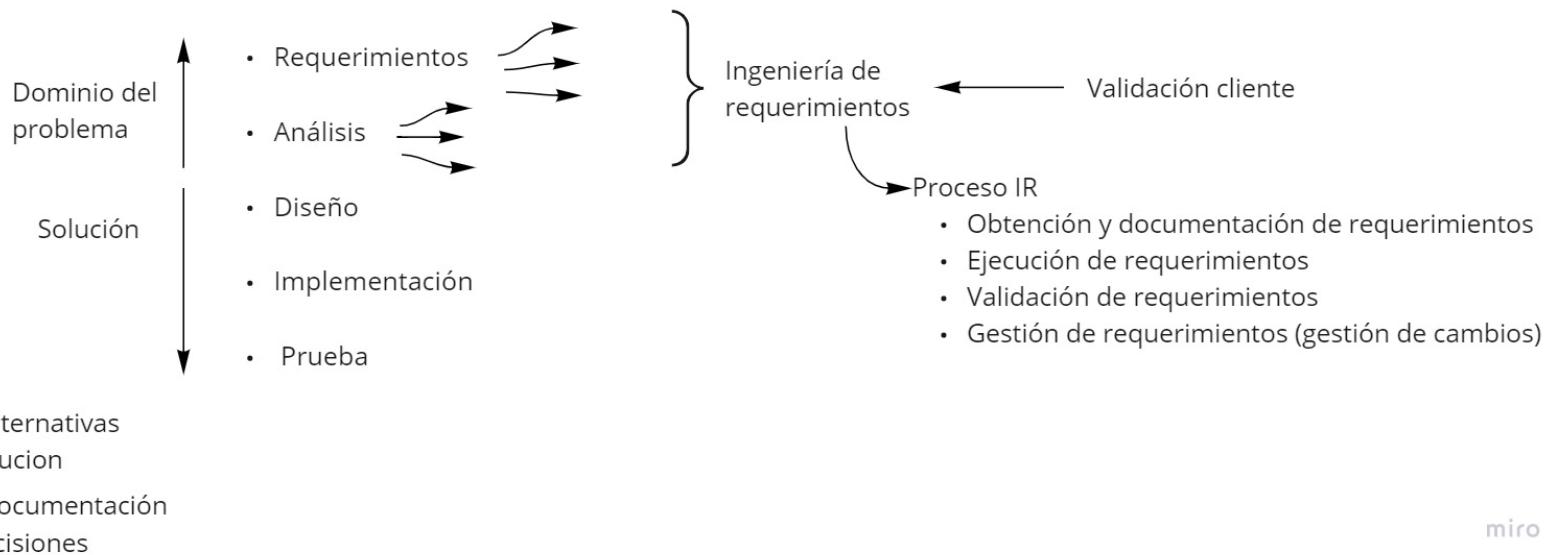
Conceptos

Ingeniería de software: surge, dentro de la ingeniería de sistemas, como una disciplina que viene a formalizar la construcción de software que en sus comienzos es solo código, donde antes solo lo entendía solo el que programaba, pero luego estas prácticas fueron cambiando gracias a las actividades que fueron surgiendo en la ingeniería de software.

Proceso de desarrollo: marco de trabajo que nosotros necesitamos organizar tomando algún proceso existente o desarrollando uno propio, pero para poder construir software.

Ciclo de vida del software: vamos a plantear las siguientes etapas que se muestran el diagrama. El proceso de desarrollo nos va a dotar de actividades que se tienen que hacer en cada una de las etapas, y el orden en el cual se van a hacer lo va a determinar la metodología que se use. Desde la etapa de requerimientos, hasta la etapa de análisis estamos en el dominio del problema, y es importante iterar varias veces los ciclos que tienen las actividades en su conjunto de cada etapa e independientemente de la metodología para lograr entender bien el problema antes de sentarme a diseñar o construir la propuesta de solución. La parte de diseño e implementación representan la solución que se propone.

- Requerimientos
- Análisis
- Diseño
- Implementación
- Prueba



_ Los requerimientos no son negociables, ya que no somos expertos en el dominio y no lo conocemos, por eso debemos escuchar al experto del dominio y trabajar mucho en sincronía con el cliente, con la problemática, con las distintas alternativas que surgen en esta etapa. Entonces, las actividades de requerimientos y de análisis de requerimientos, se complementan y trabajamos con la validación del cliente. A partir de ahí empezamos a construir una solución que tiene que ver con las dos etapas mencionadas, diseño e implementación, en donde en el diseño tenemos que plantear la arquitectura que tiene que tener decidido en que se va a implementar, por ejemplo, motor de base de datos, servicio web, librerías, lenguaje de programación, o sea todo lo que tiene que ver con la parte más de tecnología de los sistemas software, entonces a partir de ahí el diseño tiene una serie de características puntuales que lo definen e incluso que impactan en él sus clases, para esto vamos a utilizar patrones de diseño.

_ Hasta ahora con un modelo así, el ciclo de vida y el proceso de desarrollo, están muy relacionados. En la etapa de requerimientos y análisis tenemos que entender el problema, y en la de diseño tenemos que plantear alternativas de solución y también documentar decisiones de diseño que vamos a tomar.

Actividades que se realizan en la etapa de análisis de requerimientos: analizar requerimientos es seguir analizando lo que nos dijo el cliente para ver si lo interpretamos adecuadamente. Hay técnicas de prototipado que se usan para visualizar el sistema para que el cliente lo valide, pero todavía estamos en una etapa muy temprana para construir software. Entonces, en esta etapa, iteramos las veces que haga falta para validar los requerimientos con prácticas de la ingeniería de requerimientos. La ingeniería de requerimientos está formada por el siguiente proceso que permite validar que los requerimientos estén conectados desde su relevamiento hasta su implementación para poder gestionar cambios:

- Obtención y descubrimiento de requerimientos.
- Especificación de requerimientos.
- Validación de requerimientos.
- Gestión y administración de requerimientos.

Atributos de calidad de un buen diseño: cuando hablamos de usabilidad, los atributos o criterios de calidad de un buen diseño, desde el punto de vista del usuario serían, por ejemplo, que el software sea intuitivo, resistente a fallos, donde nos podamos enterar si se terminó una transacción o no, que se recupere si hay algún error en el medio, seguimiento por parte del desarrollador, notificaciones, para el caso de los juegos tenemos el rendimiento, la optimización, etc. Es muy importante el tipo de software que estamos desarrollando para determinar los atributos de calidad.

- Requerimientos no funcionales: hacen referencia a lo no funcional del sistema que estemos usando, como, por ejemplo, tiempos de respuesta, usabilidad, robustez (que tiene que ver con la arquitectura, la resistencia a fallos, usos de recursos de memoria) como el software se muestra. Los requerimientos no funcionales hacen que el usuario acepte o no el software que se está usando. Estos requerimientos están relacionados con las características de un buen diseño.

_ Ahora como desarrolladores, por ejemplo, el código del programa tiene que estar bien documentado o identado, se deben aplicar los principios de programación (buenos nombres a variables, documentación y formato del código, refinamiento y modularidad, es decir, dividir un problema en otros más pequeños en donde cada porción de programa debe hacer solo una cosa y ocultar dicha funcionalidad), se debe poder reutilizar código para minimizar el trabajo en el desarrollo y así lograr calidad en cuanto a tiempos.

_ Dentro del área de desarrollo tenemos un área que es la de planificación del proyecto, que tiene que ver con la planificación de tiempos y de costos. Esta etapa no es menor, ya que tenemos que decidir rápidamente en las primeras etapas, y tal vez antes de poder lograr detalle sobre los requerimientos, para poder hacer una estimación más certera. Lo primero que tenemos que hacer es planificar el proceso de desarrollo, y aparte nos sirve para quemar N etapas tempranas poniendo un tiempo de entrega y un costo de todo ese trabajo que vamos a hacer, y esto también hace a la calidad del diseño.

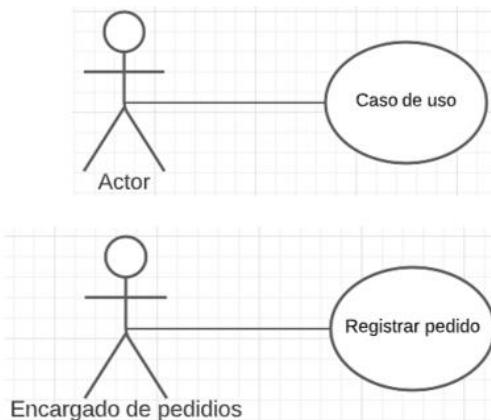
Diagrama de actividad: representan una descripción de lo que hace un caso de uso o un requerimiento. No es un diagrama de flujo, pero tenemos una secuencia de actividades paso a paso que sigue una funcionalidad.

Diseñar software: cuando diseñamos software hablamos de una primera versión o visión de una solución ya considerando el software, la tecnología, y como vamos a llevar todos los requerimientos en la especificación del dominio a un sistema de software. A veces hay

restricciones en cuanto al hardware, dispositivos, procesador, al momento de implementar con una tecnología determinada.

Modelar: cuando hablamos de modelar hablamos de una abstracción gráfica, y es gráfica porque a veces un diagrama dice más que un texto en lenguaje natural y a su vez se entiende más rápido, y decimos abstracción, ya que representamos una parte en particular y nos abstraemos de otras que no nos interesan o información que no es relevante para el tipo de diagrama que estamos haciendo, de ese modelo.

Diagrama de casos de uso: dentro de los diagramas útiles para la etapa de requerimiento y para la etapa de diseño tenemos los diagramas de casos de uso. A estos los usamos puntualmente para modelar un comportamiento del sistema. Nos enfocamos en el objetivo que tiene el diagrama. A continuación, vemos un ejemplo, donde el actor es quien tiene el permiso para ejecutar la acción y la acción misma puede escribirse con un verbo en infinitivo y detalla el comportamiento de la acción misma.



_ Como podemos observar, el actor es el único autorizado para registrar pedidos, entonces cuando modelamos el diseño de la solución, tomamos el caso de uso como un entregable a evaluar dentro de la documentación de la parte del problema, y manejamos los aspectos de seguridad en donde le daríamos acceso solamente al encargado de pedidos, ya que es el que va a poder registrar los pedidos.

_ Este diagrama, además de mostrar el comportamiento, nos da más información relevante a la hora de construir la solución.

Abstracción: es tomar una perspectiva del sistema abstrandónos de otras que no nos interesan en pos de cumplir con el objetivo del diagrama que estamos modelando o del modelo que estamos realizando.

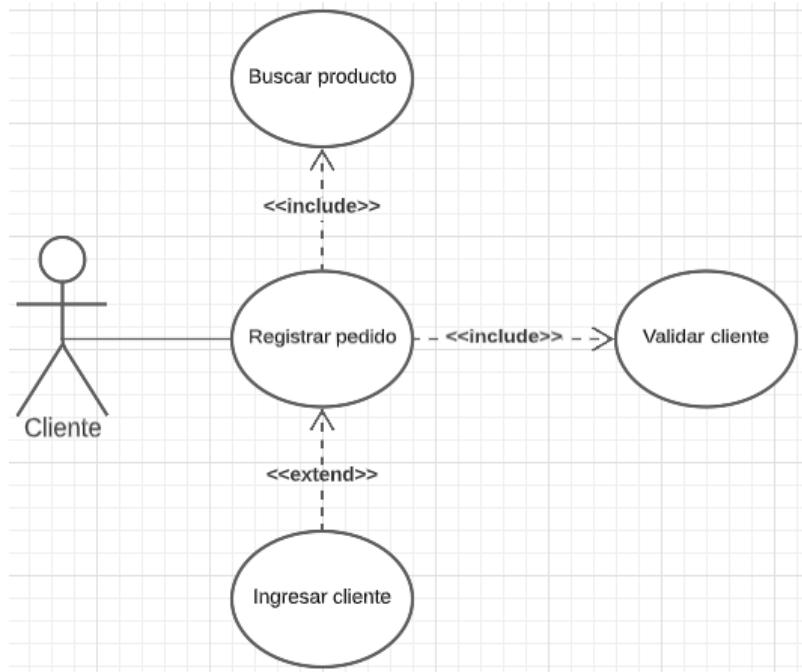
- Niveles de abstracción: los vamos a ir logrando usando perspectivas o técnicas para lograr diagramas de calidad.

Objetos de dominio: son clases conceptuales que surgen en una especificación, pero que en las etapas de requerimientos se llaman objetos del dominio.

Clases de diseño: representan la estructura del software en el ámbito de la solución del sistema. Tienen atributos y métodos, y estas están más cerca del componente de software.

Importancia de los requerimientos: estos son importantes porque minimizan la presencia de errores justamente en el producto final que es el software, y también reflejan un entendimiento del dominio.

Modularidad: es un atributo de calidad de un software. Hacer un código modular permite el tema del reuso donde se pueden agrupar requerimientos en algún módulo o subsistema que nos permita por ejemplo hacer una división de todo el sistema. Por ejemplo, modelamos de la siguiente manera:



_ En este caso tenemos un cliente que trata de registrar un pedido e incluye un caso de uso llamado “buscar producto”. “Buscar producto” es un caso de uso que a lo mejor se accede desde distintos actores en otros procesos y no solo desde “registrar pedidos”, ya que suponemos que es una empresa que genera productos y arma pedidos para enviarlos. “Buscar producto” quizás se pueda acceder desde otro módulo o sector, entonces al modelarlo como caso de uso propiciamos el reuso de esta funcionalidad para que pueda ser accedido desde otros módulos. Si tenemos un modelo de caso de uso que tiene asociaciones como este, estamos priorizando el reuso de esos casos de uso, donde vamos a agrupar funcionalidades que van a ser accedidas por distintos propósitos.

_ La modularidad no solo está presente en todo el desarrollo, sino que también los está en la parte de requerimientos. Mientras más asociaciones de funcionalidades de casos de uso tengamos, en relación a inclusión o extensión. Las relaciones de inclusión son obligatorias

y se dan siempre en el caso de uso principal, en este caso cuando se accede al caso de uso “registrar pedido”, se llama al caso de uso “buscar producto” para asociarlo y se llama al caso de uso “validar cliente” para ver los datos del mismo, y si todo está en orden el caso de uso general quizás se pueda dar de alta al mismo.

_ Para modular, hacemos toda la lista de requerimientos y después los agrupamos en módulos o al revés, identificamos los subsistemas y después le damos detalle a los mismos.

Usabilidad: significa lograr satisfacer las necesidades del usuario. Esta es una disciplina que está enfocada en la visión del sistema del lado del usuario.

Confiabilidad: está asociado con la seguridad del software, en donde hablamos de seguridad de acceso y la seguridad de que no quede vulnerable a ciertos ataques cibernéticos.

Robustez: hace referencia a que el sistema se pueda componer ante fallas o un mal comportamiento.

Complejidad: las actividades dentro del proceso de desarrollo, código o entregables, generan una cierta complejidad que de alguna manera uno tiene que tratar de minimizar y evitar arrastrar a través de técnicas. Esta complejidad está en todas las etapas de desarrollo del software. La comunicación entre el cliente y el desarrollador ya genera complejidades en su comunicación, entonces a partir de ahí ya tenemos complejidad. También tenemos la complejidad en los cambios cuando aparece un error para poder lidiar con el mismo y que ese cambio se implemente en todo el proceso de desarrollo, entre otros.

Arquitectura de software: está del lado de la solución y se bosqueja en el diseño. Es la estructura del sistema que de alguna manera mezcla todo, tecnología, servicios, lenguajes de programación, etc. La arquitectura se define como una columna vertebral que le da soporte estructural al software, y garantiza su funcionamiento cumpliendo con los requerimientos de calidad de software. Los atributos de calidad de software se reflejan en un buen diseño de la arquitectura.

Reusabilidad: la idea es trabajar con componentes reusables.

Portabilidad: hace referencia a que el software se pueda utilizar en distintas plataformas, celulares, tablets, etc.

Mantenibilidad: está asociado a mantener el software una vez que se implementa.

Trazabilidad: ayuda a mantener actualizada toda la documentación que se genera por los cambios. Permite analizar qué tan complejo es implementar un cambio.

Modelado

Introducción

Modelo

_ Podemos definir a un modelo como:

- Una simplificación de la realidad.
- Construimos modelos para poder comprender mejor el sistema que estamos desarrollando.
- Es una representación a bajo costo de la realidad.
- Abstracción de un sistema. La abstracción permite ocuparnos de detalles relevantes para un propósito.
- Utilidad que permite abordar sistemas complejos.

Modelado

_ El lenguaje empleado para modelar el software es el lenguaje visual, como un diagrama de clases del dominio o caso de uso, ya que es fácil de interpretar y procesar, a diferencia de un código fuente o un lenguaje natural.

Objetivos de modelar:

- Nos ayudan a visualizar un sistema como es, o como queremos que sea.
- Nos permiten especificar la estructura o el comportamiento de un sistema.
- Nos dan una plantilla que nos guía en la construcción del sistema.
- Documentan las decisiones que hemos tomado.
- Nos facilitan la comunicación con el cliente.

Principios de modelado:

- La elección de qué modelos crear tiene una profunda influencia en cómo atacamos un problema y cómo delineamos una solución.
 - Diferentes enfoques (analista estructurado, analista orientado a objeto o desarrollador de base de datos).
- Cada modelo debe poder expresarse a diferentes niveles de precisión.
- Los mejores modelos tienen una conexión con la realidad.
- Un único modelo no es suficiente. Cualquier sistema es enfocado mejor con un conjunto de modelos independientes, pero relacionados.

Aportes del modelado:

- Se facilita la comunicación entre el equipo al existir un lenguaje común.
- Se dispone de documentación que trasciende al proyecto.
- Hay estructuras que trascienden lo representable en un lenguaje de programación

Conceptos del modelado:

- Sistema software: descripto por un conjunto de modelos.
- Modelo: simplificación para comprender mejor un sistema.
- Diagrama: representación gráfica de un modelo (UML usa grafos).
- Vista: subconjunto de diagramas de un modelo que analiza un aspecto concreto. En el contexto del software hay cinco vistas complementarias que son las más importantes para visualizar, especificar, construir y documentar una arquitectura software:
 - Vista de casos de uso: funcionalidades del sistema.
 - Vista de diseño: clases, interfaces, requerimientos funcionales.
 - Vista de interacción: flujo de datos, requerimientos no funcionales.
 - Vista de implementación: archivos, código fuente.
 - Vista de despliegue: instalación y ejecución del sistema.

Cada una de estas vistas involucra modelado estructural y modelado de comportamiento.

Ingeniería dirigida por modelos (MDE): es un enfoque para el desarrollo de software donde los modelos en lugar de los programas son los principales resultados del proceso de desarrollo. Los programas que se ejecutan en una plataforma de hardware/software se generan automáticamente a partir de los modelos.

Importancia de modelar

Trabajo de investigación

_ Tenemos que determinar:

- Problemática
- Propuesta de solución
- Resultados

_ En el caso de los resultados, no solamente son los esperados. A veces la propuesta de solución genera resultados que de alguna manera validan que la hipótesis de esa técnica para desarrollar, por ejemplo, un software no sería validada. Esto sirve para trabajos de investigación, porque la investigación misma se llevó a cabo. Tenemos que planear la problemática con esa hipótesis y a partir de ahí que se pueda probar una solución en función a una demostración de la misma.

Licitación de requerimientos: es una de las técnicas que se utiliza en la etapa de ingeniería de requerimientos para descubrir justamente requerimientos en dominios que no son tan intuitivos, y que no dependen tanto del uso sino más de procedimientos de negocio, control, etc.

- El paper X propone una técnica para descubrir requerimientos no funcionales, donde los aspectos tienen que ver con la calidad del sistema, por ejemplo, que sostengan un sistema de manera robusta, que el sistema cumpla con condiciones de seguridad, que accesos o vulnerabilidades puede llegar a tener, es decir, que sea de alguna manera consistente gracias a esos requerimientos no funcionales basados en la gestión de conocimiento de los stakeholders. Entonces, a estos requerimientos no funcionales se los va a plantear desde la interacción con los stakeholders o los usuarios, y a partir de ahí se desarrolla una serie de técnicas y una propuesta de solución con un modelo, y por último la conclusión y los resultados.

_ En el área de ingeniería de requerimientos cualquier técnica ayuda a mejorar la comunicación con el cliente, el modelado ayuda a especificar mejor los requerimientos, y los papers ayudan a ser más rigurosos cuando estamos generando un proceso de desarrollo.

Buenas prácticas en el modelado

- Tener conocimiento de la situación: realizar investigaciones y lectura del caso.
- Organización.
- Elegimos un proceso de desarrollo, una guía de las actividades a seguir.
- Complementamos con nuestra experiencia: en ese proceso comenzamos definiendo los objetos o los requerimientos.
- Modelamos: donde vemos que vamos a modelar, que subsistemas vamos a tener, cuáles son los procesos funcionales, los objetos, los casos de uso, que diagramas usamos (diagrama de contexto, diagrama de flujo, diagrama de actividad).

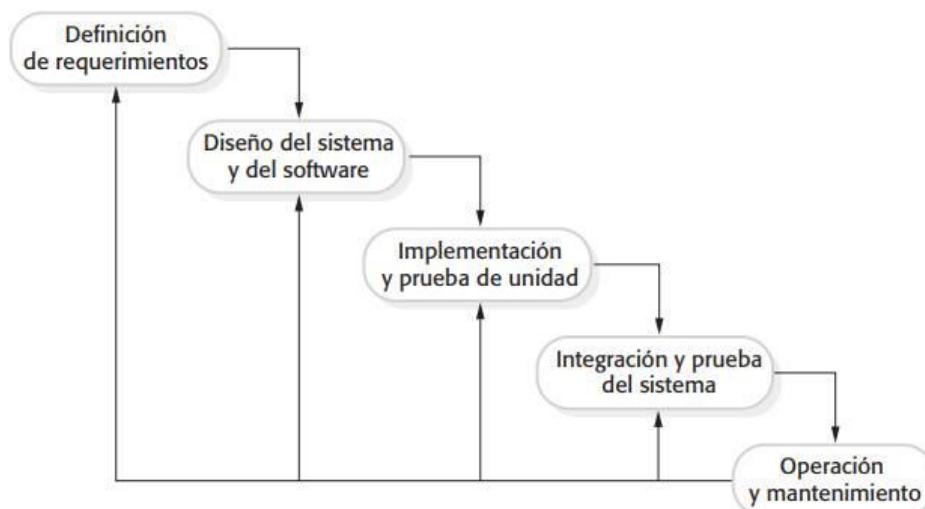
Proceso de desarrollo

_ Para realizar el proyecto tenemos que tener en cuenta el modelado, que diagramas nos sirven para entender cada vez más el dominio (mundo de problemas y mundo de la solución), y el proceso de desarrollo. El proceso de desarrollo va a estar condicionado por el tipo de sistema o proyecto que vayamos a abordar.

Modelos de procesos: en toda organización tenemos tres modelos de procesos distintos:

- Cascada: a continuación, vemos como se organizan las etapas del ciclo de vida del proceso de desarrollo de software en este modelo, que a su vez es iterativo e

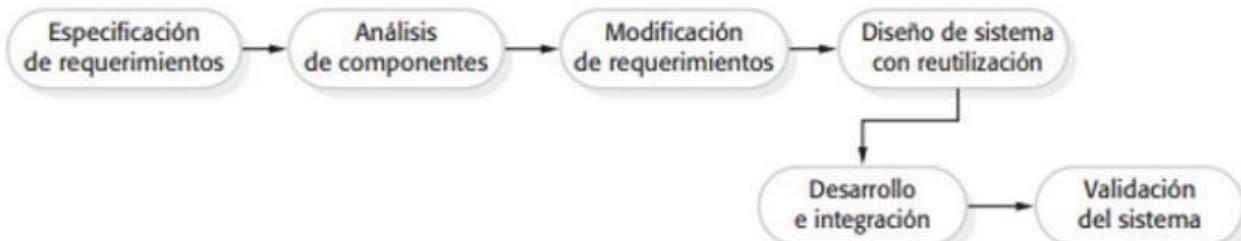
incremental. En este, las etapas están bien definidas y cuando termina una pasa a la siguiente.



- Desarrollo evolutivo: está dentro de la categoría incremental, y está basado en un bosquejo de requerimientos.



- Integración por componentes (reutilización): se usa mucho cuando reusamos lo que ya tenemos, o a lo mejor desarrollamos sistemas similares. Reusamos componentes y tratamos de ir adaptando los requerimientos nuevos a los que ya tenemos y eso con tal de achicar tiempos de desarrollo.



_ Cualquier metodología que adoptemos tenemos que ver si cumple con las características o que características del modelo forma parte de esa categoría. Cada tipo de sistema que desarrollamos nos condiciona a elegir algún proceso con ciertas características, entonces para considerar un proceso de desarrollo tenemos que considerar o tener en cuenta que tipo sistema estamos hablando.

_ Al momento de elegir un modelo para el proceso de desarrollo, recordamos que el modelado, que viene a ser un poco como estructuramos todo, arranca con un objetivo, después vamos a un subsistema en donde identificamos requerimientos en texto plano o en un lenguaje natural que describe los procedimientos funcionales correspondientes a un dominio. A todo esto, lo tenemos que estructurar de alguna manera para sistematizarlo y posteriormente se convierta en un producto de software operativo, y para esto tenemos que ver cuál es la mejor técnica que nos haga entender mejor lo que estamos modelando, ya sea en el mundo del problema o ubicado ya cuando estemos construyendo sistemas en el mundo de la solución. Entonces, para lograr esto primero tenemos que plantear un alcance, luego una lista de objetos del dominio, después hacemos casos de uso, especificamos los requerimientos, hacemos un diagrama de actividad , en donde todo este orden en el proceso, que vamos sistematizando para modelar y para mejorar este entendimiento del dominio que justamente estamos tratando de entender, son lo que hacemos cuando modelamos. Mientras más diagramas de casos de uso, más perspectivas planteamos cuando modelamos. Pero a la hora de elegir un proceso, también tenemos que considerar el objetivo del sistema que son necesarios y que se adapte mejor al desarrollo que elegimos.

Criterios a considerar

_ A continuación tenemos los criterios que consideramos para elegir un proceso de desarrollo:

Criterio 1: nivel de flexibilidad de los requerimientos:

- Cuando hablamos de desarrollo web, hablamos de una página simple que quizás pueda tener una base de datos por detrás u otro servicio, y que tiene tanta flexibilidad en los requerimientos porque el usuario, a medida que vayan saliendo las primeras versiones de la página web, siempre la va a estar modificando y agregando cosas. Entonces, no podemos tener una especificación de requerimientos estricta, sino que la tenemos que bosquejar y pasarle primeras versiones al cliente para que la valide rápidamente y a partir de ahí retroalimentar y mejorar. No podemos trabajar con mucha rigurosidad en la etapa de requerimientos porque seguramente, al ser un desarrollo web, predomina más la estética, la dinámica y la usabilidad, por lo que necesita ser validadas en etapas muy tempranas. Por ende, cuando hablamos de un desarrollo web, hablamos de

algo ágil y rápido como un desarrollo evolutivo, obteniendo una versión inicial rápida para ser mejorada.

- En un contexto de cambios permanentes que deben ser introducidos frecuentemente, volviendo con la página web, hacemos referencia a los cambios solicitados por el cliente y por ende al desarrollo evolutivo.
- Cuando hablamos de desarrollo de aplicaciones clásicas, un ambiente estable, donde no hay cambios en el dominio, los equipos de trabajo son estratificados (tenemos un gran equipo de desarrollo y estos están bien distribuidos y cada uno hace su parte o función), las fases de desarrollo son estables y se respetan todas las etapas planificadas, un entorno predecible, etc, estos ambientes estables en los que no implican grandes cambios se basan en procedimientos bien definidos y se trabaja con un desarrollo cascada.

Criterio 2: considerar la escala (tamaño) y el alcance del desarrollo:

- En proyectos grandes, con mucha cantidad de integrantes, hay que realizar planes de gestión bien elaborados. Por ende, consideramos que lo mejor sería utilizar un proceso de desarrollo en cascada.

Criterio 3: que conviene más teniendo en cuenta lo que exige el cliente:

- Desarrollar software a través de un sprint facilita el lanzamiento de sistemas realmente complejos y genera una impresión de progresión rápida. Esto es útil para clientes exigentes que quieren ver avances o que no confían en lo que se está desarrollando, también para clientes que no se hacen una idea del sistema que necesitan y tienen que ver funcionando algo para tener una visión de lo que realmente quieren. Por ende, para estos casos, aplicamos desarrollo evolutivo.
- Si contamos con períodos de desarrollos a largo plazo, con fechas de entrega que no se aproximan rápidamente, en este caso aplicamos un proceso de desarrollo cascada.

Criterio 4: definir los usuarios finales claramente:

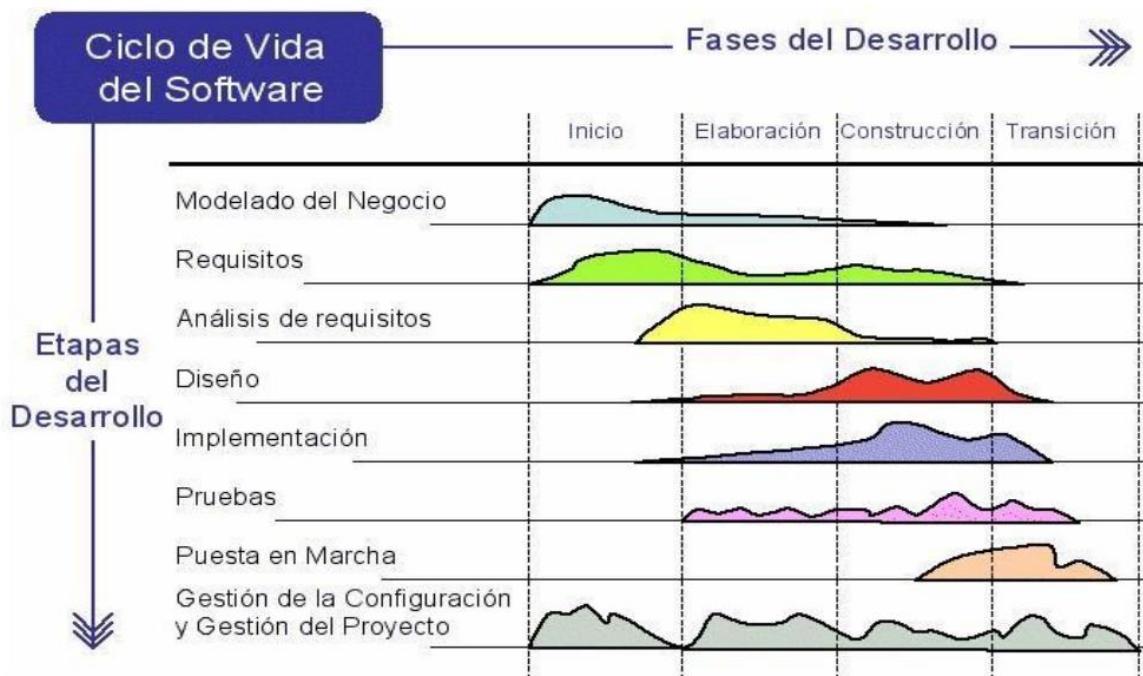
- Un grupo controlado de usuarios finales probablemente tenga un conjunto de requerimientos fijos con los que trabajar, lo que hace que el método de cascada sea ideal. Pero si los objetivos finales están dispersos, habrá que encargarse del feedback después del lanzamiento de la aplicación pidiendo la inclusión de nuevas funciones.

Criterio 5: considerar donde se ubica el equipo de desarrollo:

- Si el equipo de desarrolladores está disperso en todo el mundo, hará falta un mayor nivel de coordinación, y responsabilidad. En este caso, un régimen de gestión de proyecto más rígido es la mejor opción, y este es el caso donde el

método de la cascada reuce más. El método requiere un contacto más frecuente y equipos más unidos, pues un equipo de desarrolladores disperso podría trabajar con mucha confusión y errores en el proceso de desarrollo si el modelo elegido es el ágil.

Presentación del proceso unificado de desarrollo de software (PUDS)



Modelado de negocio

- La perspectiva de cómo se mueven los procesos en lugar en donde se construirá una solución de software (institución, empresa, dominio).
- Los diagramas de procesos del negocio, son representaciones de los procesos que se dan en la organización y que son de interés para el sistema.
- Se trata de representar la relación existente entre un proceso y las unidades funcionales (como pueden ser los departamentos o personas) responsables del proceso.
- Es un diagrama de flujo extendido, en donde además, las operaciones se detallan las personas que intervienen en un proceso.

Objetivo

_ Los objetivos deben estar planteados desde la perspectiva del negocio:

- Deben definir porque se construye el sistema para tal fin.
- Que beneficios obtiene el negocio en la construcción del sistema.
- No debe describir funcionalidades.

_ Esto nos permite ubicarnos desde la perspectiva del cliente y del usuario y no solamente como desarrollador. Por ejemplo, un sistema contra incendios y detección de intrusos para un edificio de oficinas, el objetivo incorrecto sería construir un sistema de alarma contra incendios e intrusos para el edificio que proporcione avisos de fuego y de avisos de accesos no autorizadas tanto internas como externas, ahora, el objetivo correcto sería asegurar que el funcionamiento normal de los trabajos realizados en el edificio no se interrumpa por eventos como el fuego y accesos no autorizados.

Proceso Unificado de Desarrollo de Software

Concepto

_ El proceso unificado es un proceso de desarrollo de software, y lo podemos entender como un conjunto de actividades necesarias para transformar los requisitos del usuario en un sistema software. Tiene las siguientes características:

- Es un marco genérico que puede especializarse para una variedad de tipos de sistemas, diferentes áreas de aplicación, tipos de organizaciones, niveles de aptitud y diferentes tamaños de proyectos.
- Está basado en componentes. El software está formado por componentes software interconectados a través de interfaces.
- Es un proceso orientado a objetos.
- Está dirigido por casos de uso, centrado en la arquitectura, y es iterativo e incremental.

Dirigido por casos de uso

Caso de uso: es un fragmento de funcionalidad del sistema que proporciona un resultado de valor a un usuario. Los casos de uso modelan los requerimientos funcionales del sistema.

- Todos los casos de uso juntos constituyen el modelo de casos de uso.
- Los casos de uso también guían el proceso de desarrollo (diseño, implementación, y prueba). Basándose en los casos de uso los desarrolladores crean una serie de modelos de diseño e implementación que llevan a cabo los casos de uso. De este modo los casos de uso no solo inician el proceso de desarrollo, sino que le proporcionan un hilo conductor, avanza a través de una serie de flujos de trabajo que parten de los casos de uso.

Centrado en la Arquitectura

Arquitectura: conjunto de decisiones significativas acerca de la organización de un sistema software, la selección de los elementos estructurales a partir de los cuales se compone el sistema, las interfaces entre ellos, su comportamiento, sus colaboraciones, y su composición.

_ La arquitectura de un sistema software se describe mediante diferentes vistas del sistema en construcción. El concepto de arquitectura software incluye los aspectos estáticos y dinámicos más significativos del sistema. La arquitectura es una vista del diseño completo con las características más importantes resaltadas, dejando los detalles de lado.

_ Los casos de uso y la arquitectura están profundamente relacionados. Los casos de uso deben encajar en la arquitectura, y a su vez la arquitectura debe permitir el desarrollo de todos los casos de uso requeridos, actualmente y a futuro.

_ El arquitecto desarrolla la forma o arquitectura a partir de la comprensión de un conjunto reducido de casos de uso fundamentales o críticos (usualmente no más del 10 % del total). En forma resumida, podemos decir que el arquitecto:

- Crea un esquema en borrador de la arquitectura comenzando por la parte no específica de los casos de uso (por ejemplo, la plataforma) pero con una comprensión general de los casos de uso fundamentales.
- A continuación, trabaja con un conjunto de casos de uso claves o fundamentales. Cada caso de uso es especificado en detalle y realizado en términos de subsistemas, clases, y componentes.
- A medida que los casos de uso se especifican y maduran, se descubre más de la arquitectura, y esto a su vez lleva a la maduración de más casos de uso.

_ Este proceso continúa hasta que se considere que la arquitectura es estable.

Iterativo e Incremental

_ Es práctico dividir el esfuerzo de desarrollo de un proyecto de software en partes más pequeñas o mini proyectos. Cada mini proyecto es una iteración que resulta en un incremento. Las iteraciones hacen referencia a pasos en el flujo de trabajo, y los incrementos a crecimientos en el producto.

_ Las iteraciones deben estar controladas. Esto significa que deben seleccionarse y ejecutarse de una forma planificada. Los desarrolladores basan la selección de lo que implementarán en cada iteración en dos cosas, el conjunto de casos de uso que amplían la funcionalidad, y en los riesgos más importantes que deben mitigarse. En cada iteración los desarrolladores identifican y especifican los casos de uso relevantes, crean un diseño utilizando la arquitectura seleccionada como guía, para implementar dichos casos de uso.

Si la iteración cumple sus objetivos, se continúa con la próxima. Si no deben revisarse las decisiones previas y probar un nuevo enfoque.

Beneficios del enfoque iterativo:

- La iteración controlada reduce el riesgo a los costes de un solo incremento.
- Reduce el riesgo de retrasos en el calendario atacando los riesgos más importantes primero.
- Acelera el desarrollo. Los trabajadores trabajan de manera más eficiente al obtener resultados a corto plazo.
- Tiene un enfoque más realista al reconocer que los requisitos no pueden definirse completamente al principio.

Ciclo de vida

— El proceso unificado se repite a lo largo de una serie de ciclos que constituyen la vida de un sistema. Cada ciclo constituye una versión del sistema.

Fases: cada ciclo consta de cuatro fases:



— Cada fase se subdivide en iteraciones. En cada iteración se desarrolla en secuencia un conjunto de disciplinas o flujos de trabajos.

Disciplinas: cada disciplina es un conjunto de actividades relacionadas (flujos de trabajo) vinculadas a un área específica dentro del proyecto total. Las más importantes son:

- Captura de requisitos: identificar requisitos del sistema, construir un modelo del mismo, modelo de casos de uso, modelo del dominio (o negocio).

- Análisis: especificar requisitos, construir modelo del análisis.
- Diseño: encontrar la forma del sistema (solución), construir modelo del diseño.
- Codificación (Implementación): codificar el diseño (solución), construir modelo de implementación.
- Pruebas: verificar la implementación, construir modelo de pruebas.

_ El agrupamiento de actividades en disciplinas es principalmente una ayuda para comprender el proyecto desde la visión tradicional en cascada.



_ Cada disciplina está asociada con un conjunto de modelos que se desarrollan. Estos modelos están compuestos por artefactos. Los artefactos más importantes son los modelos que cada disciplina realiza:

- Modelo de casos de uso
- Modelo de diseño
- Modelo de implementación
- Modelo de prueba

_ El proceso unificado consiste en una serie de disciplinas o flujos de trabajo que van desde los requisitos hasta las pruebas. Los flujos de trabajo desarrollan modelos desde el modelo de casos de uso hasta el modelo de pruebas.

Disciplina	Modelos
Requisitos	Modelo de Casos de Uso
Análisis	Modelo de Análisis
Diseño	Modelo de Diseño - Modelo de Despliegue
Implementación	Modelo de Implementación
Prueba	Modelo de Prueba

Hitos: cada fase finaliza con un hito.

- Cada hito se determina por la disponibilidad de un conjunto de artefactos, es decir un conjunto de modelos o documentos que han sido desarrollados hasta alcanzar un estado predefinido.
- Los hitos tienen muchos objetivos, en donde el más crítico es que los directores deben tomar ciertas decisiones antes de que el trabajo continúe con la siguiente fase.
- Los hitos también permiten controlar la dirección y progreso del trabajo.
- Al final se obtiene un conjunto de datos a partir del seguimiento del tiempo y esfuerzo consumidos en cada fase. Estos datos son útiles para las estimaciones en futuros proyectos.

Producto del proceso unificado: no es sólo código ejecutable, son los modelos o representación del software, y debe ajustarse a todas las personas implicadas.

Fase de inicio

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

- ¿Cuáles son las principales funciones del sistema para los usuarios más importantes?
- ¿Cómo podría ser la mejor arquitectura del sistema?
- ¿Cuál es el plan del proyecto y cuánto costará desarrollar el producto?

_ Por otro lado, se identifican y priorizan los riesgos más importantes.

_ El objetivo de esta fase es ayudar al equipo de proyecto a decidir cuáles son los verdaderos objetivos del proyecto. Las iteraciones exploran diferentes soluciones posibles, y diferentes arquitecturas posibles. Puede que todo el trabajo físico realizado en esta fase sea descartado. Lo único que normalmente sobrevive a la fase de inicio es el incremento del conocimiento en el equipo. Los artefactos que típicamente sobreviven a esta fase son:

- Un enunciado de los mayores requerimientos planteados generalmente como casos de uso.
- Un boceto inicial de la arquitectura.
- Una descripción de los objetivos del proyecto.
- Una versión muy preliminar del plan del proyecto.
- Un modelo del negocio.

_ La fase de inicio finaliza con el Hito de Objetivos del Ciclo de Vida. Este hito es alcanzado cuando el equipo de proyectos y los stakeholders llegan a un acuerdo sobre:

- Cuál es el conjunto de necesidades del negocio, y que conjunto de funciones satisfacen estas necesidades.
- Una planificación preliminar de iteraciones.
- Una arquitectura preliminar.

_ Debe poder responderse las siguientes cuestiones:

- ¿Se ha determinado con claridad el ámbito del sistema?
- ¿Se ha llegado a un acuerdo con todas las personas involucradas (stakeholders) sobre los requisitos funcionales del sistema?
- ¿Se identifican los riesgos críticos?
- ¿El uso del producto justifica la relación costo-beneficio?
- ¿Es factible para su organización llevar adelante el proyecto?
- ¿Están los inversores de acuerdo con los objetivos?

Fase de elaboración

_ Durante la fase de elaboración se especifican en detalle la mayoría de los casos de uso del producto y se diseña la arquitectura. Las iteraciones en la fase de elaboración:

- Establecen una firme comprensión del problema a solucionar.
- Establece la fundación arquitectural para el software.
- Establece un plan detallado para las siguientes iteraciones.
- Elimina los mayores riesgos.

_ El resultado de esta fase es la línea base de la arquitectura. En esta fase se construyen típicamente los siguientes artefactos:

- El cuerpo básico del software en la forma de un prototipo arquitectural.
- Casos de prueba.
- Un plan detallado para las siguientes iteraciones.

_ La fase de elaboración finaliza con el hito de la Arquitectura del Ciclo de Vida. Este hito se alcanza cuando el equipo de desarrollo y los stakeholders llegan a un acuerdo sobre:

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

_ Al alcanzar este hito debe poder responderse a preguntas como:

- ¿Se ha creado una línea base de la arquitectura? ¿Es adaptable y robusta? ¿Puede evolucionar?
- ¿Se han identificado y mitigado los riesgos más graves?

- ¿Proporciona el proyecto, una adecuada recuperación de la inversión?
- ¿Se ha obtenido la aprobación de los inversores?

Fase de construcción

_ Durante la fase de construcción se crea el producto. La línea base de la arquitectura crece hasta convertirse en el sistema completo. Al final de esta fase, el producto contiene todos los casos de uso implementados, sin embargo, puede que no esté libre de defectos. Los artefactos producidos durante esta fase son:

- El sistema software.
- Los casos de prueba.
- Los manuales de usuario.

_ La fase de construcción finaliza con el hito de Capacidad Operativa Inicial. Este hito se alcanza cuando el equipo de desarrollo y los stakeholders llegan a un acuerdo sobre:

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

Fase de transición

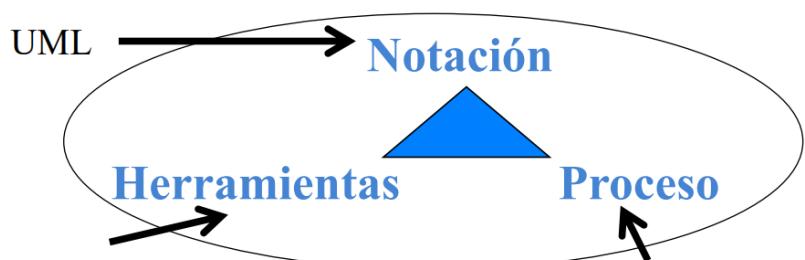
_ La fase de transición cubre el período durante el cual el producto se convierte en la versión beta. Las iteraciones en esta fase continúan agregando características al software. Sin embargo, las características se agregan a un sistema que el usuario se encuentra utilizando activamente. Los artefactos construidos en esta fase son los mismos que en la fase de construcción. El equipo se encuentra ocupado fundamentalmente en corregir y extender la funcionalidad del sistema desarrollado en la fase anterior.

_ La fase de transición finaliza con el hito de Lanzamiento del Producto. Este hito se alcanza cuando el equipo de desarrollo y los stakeholders llegan a un acuerdo sobre:

- Se han alcanzado los objetivos fijados en la fase de Inicio.
- El usuario está satisfecho.

Unified Modeling Language (UML)

_ El proceso unificado de desarrollo usa UML.



- RATIONAL ROSE
- VISIO

PROCESO UNIFICADO DE
DESARROLLO DE RATIONAL

_ UML es un lenguaje de modelado, de propósito general, usado para la visualización, especificación, construcción y documentación de sistemas Orientados a Objeto. UML combina notaciones provenientes desde:

- Modelado orientado a objetos.
- Modelado de datos.
- Modelado de componentes.
- Modelado de flujos de trabajo (Workflows).

_ UML es un lenguaje ya que un lenguaje provee un vocabulario y reglas para combinar los elementos del vocabulario con el propósito de comunicar. En un lenguaje de modelado ese vocabulario y reglas se focalizan en representaciones conceptuales y físicas de un sistema.



_ UML no es un lenguaje de programación visual, o la especificación de una herramienta o un repositorio, y tampoco es una metodología, método o proceso.

Bloques de construcción de UML

Diagrama de casos de uso: muestra un conjunto de casos de uso, actores y sus relaciones. Permite establecer el comportamiento deseado del sistema. Especifica los requerimientos funcionales del Software.

Especificación de casos de uso (Trazo Fino): son los lineamientos para el contenido del flujo de eventos. Esto quiere decir que:

- Describe cómo inicia y termina el Caso de Uso.
- Describe qué datos se intercambian entre el actor y el Caso de Uso.
- No describe detalles de la interfaz del usuario, a menos que sea necesario para entender el comportamiento del sistema.
- Describe solo los eventos que pertenecen a ese Caso de Uso, y no lo que pasa en otros Casos de Uso o fuera del sistema.

_ Una pre-condición es una restricción sobre cuando un Caso de Uso puede empezar, pero no es el evento que inicia el Caso de Uso. Una post-condición para un Caso de Uso debe ser verdadera, independientemente de cuál flujo sea ejecutado, si algo puede fallar, debería cubrirse en la post condición diciendo: “ La acción se ha completado o si algo ha fallado, la acción no se ha realizado”, en lugar de decir “La acción se ha completado”

Requerimientos

Introducción

Concepto

_ Un requerimiento de software define las funciones, capacidades o atributos de cualquier sistema de software. Los requerimientos representan:

- Factores de calidad del sistema, que permitirán evaluar su utilidad a un cliente o usuario.
- Los datos de entrada al proceso de desarrollo de software y lo que se quiere implementar.
- Una descripción del comportamiento del sistema, describe información del dominio de la aplicación, restricciones de la operación del sistema y especifica atributos o propiedades del sistema.
- Un problema por resolver.

Características de los requerimientos:

- Están por escritos
- Verificables
- Viables
- Necesarios
- Completos
- Priorizables
- No ambiguos
- Correctos
- Precisos

Tipos de requerimientos:

- Funcionales: describen servicios o funciones.
- No funcionales: son un límite en el sistema o en el proceso de desarrollo.
- De Dominio: se obtienen del dominio de la aplicación del sistema y reflejan características.

Redacción de requerimientos

_ Los requerimientos son vitales para el entendimiento del problema. Una buena redacción facilita una buena interpretación y también facilita la validación. Todos los requerimientos deben tener el mismo estilo de redacción. A continuación, vemos las partes de la reacción:

- Lugar, tiempo, evento u objeto.
- Quien o el sujeto con el rol o perfil adecuado para hacer ese requerimiento.
- Debe, deberá, no debe, no deberá. La mayoría de las veces tratamos de redactar los requerimientos desde lo positivo, es decir, lo que permite hacer y no lo que no permite hacer.
- Acción, verbo o sentencia de la funcionalidad puntual que el sistema hace.

_ Tenemos que entender estas cuatro partes y hacer los requerimientos los más completos posibles. A continuación, vemos algunos ejemplos:

1)_ “El sistema en el módulo de ventas debe ofrecer una opción mediante la cual el usuario pueda ver una comparación de lo presupuestado versus la venta real en un rango de fechas”.

_ Para este caso, tenemos primero como se menciona en que etapa o en que parte del sistema ocurre, luego cual es la acción o funcionalidad que deberá hacer el sujeto, quien es la persona que en este caso es un usuario genérico. Luego, para ver la relación del requerimiento con los casos de usos, determinamos en este ejemplo que el caso de uso sería “comparar presupuesto con la venta real”, entonces esta sería la acción concreta que sale de este requerimiento.

2)_ “Cuando el tiempo de conexión excede el valor predeterminado como máximo, el sistema debe cancelar la selección informándole al usuario que el tiempo se agotó y por lo tanto será desconectado”.

_ Tenemos el lugar y tiempo en donde sucede la funcionalidad, luego se le menciona al actor que es el usuario la funcionalidad que realizará. Este ejemplo sucede en los casos de inicio de sesión. En este caso tenemos el problema de la ambigüedad cuando hace referencia al tiempo ya que al momento de validar no vamos a saber justamente cuánto es ese tiempo permitido, por lo tanto, debemos cuantificar cuando hablamos de valores y es importante que se describa mejor el negocio o la situación en el mismo requerimiento.

3)_ “En el reporte de mercados objetivos, al aplicarle filtros por asesor y zona, muestra la información en desorden, se debe corregir de tal forma que salga ordenado por número de cédula del asesor y agrupado por zona.”

_ Primero tenemos el lugar o el módulo, luego la condición o que hace el sistema y después se menciona a el sujeto en este caso es el mismo reporte.

Consejos para escribir requerimientos

- Mantener sentencias y párrafos cortos.
- El tema de la cohesión es importante, ya que el requerimiento debe escribir una sola función.
- Deben tener una apropiada gramática, ortografía y puntuación.

- Se puede hacer un glosario para definir palabras de ser necesario para evitar la ambigüedad.
- Se puede utilizar términos consistentes definidos en el glosario.
- Cuando tenemos que “el sistema debe, deberá...” estas deben estar seguidas de una acción. Estas palabras se pueden repetir las veces que sea necesario.
- Para evitar ambigüedades no se recomienda usar palabras como amigable, fácil, simple, rápido, fuerte, superior, aceptable, robusto.
- Si nos encontramos con “y”, “o”, “etc”, esto puede representar varios requerimientos.

_ A continuación vemos una lista de términos ambiguos:

Termino	Ejemplo de uso	Como mejorarlo
Aceptable, adecuado	“Las interfaces deben ser aceptables.” “El tiempo de espera en el reporte de ventas mensuales deben ser adecuado.”	Definir qué significa aceptable, por ejemplo, que tal cosa sea aceptable según quién.
Tanto como sea posible	“Permitir tantos usuarios conectados como sea posible.”	No dejar a los desarrolladores determinar lo que es posible, prepare un set de datos que evalúen la factibilidad.
Al menos..., en un mínimo.., no más que..., que no exceda...	“El tiempo de ejecución del reporte mensual no supere 10.”	Especificar el valor mínimo y máximo.
Entre	“El porcentaje de error tolerable en el proceso de facturación está planteado entre 1 y 10 cada 10.000 facturas generadas.”	Definan si los puntos límites están incluidos.
Depende de	“La ejecución del proceso de venta depende de que se ejecute con éxito el sistema de facturación.”	Existen entradas concretas, servicios activos, etc, que se deben evaluar de la interacción con otros sistemas que hay que especificar.
Eficiente	“El sistema debe tener un comportamiento eficiente.”	Definir qué significa eficiente, por ejemplo, si esta referido a la rapidez de ejecución de un proceso o a la facilidad de uso para el usuario.

Rápido	“Los reportes estadísticos deben ser rápidos.”	Especificar el valor mínimo aceptable de velocidad cuando se ejecuta una acción del sistema.
Flexible	“El proceso de generación de contratos laborales debe ser flexible.”	Describir la forma en la que el sistema debe adaptarse a los cambios de las condiciones o las necesidades del negocio.
Mejorando, mejor, más rápido, superior	“Requerimiento de cambio: mejorar todos los tiempos de respuesta de los reportes estadísticos.”	Cuantificar cuanto mejor o más rápido constituye una mejoría adecuada en un área funcional específica.
Incluyendo..., así sucesivamente, etc, tal como...	“El sistema debe permitir cargar los datos personales del cliente como nombre, apellido, etc.”	La lista de los items debe incluir todas las posibilidades, de lo contrario no será útil para el diseño o el testing.
Maximizar, minimizar, optimizar	“Optimizar el proceso de facturación.”	Especificar con valor el máximo, el mínimo y el óptimo.
Normalmente, idealmente	“Idealmente debería captar los códigos de error de la BD y mostrarlo con un mensaje más amigable.”	Describir específicamente el funcionamiento del sistema ante una condición anormal.
Opcionalmente	“Opcionalmente en los parámetros de fecha debería mostrar un calendario.”	Aclarar si es opcional para el sistema, para el usuario o para el desarrollador.
Razonablemente, cuando sea necesario, donde sea apropiado	“Cuando sea necesario proveer una ayuda online.”	Explicar que significa ese juicio.
Robusto	“El sistema debe ser robusto.”	Definir como el sistema maneja las excepciones de la BD y como responde a condiciones inesperadas operativas.
Elegante, prolíjo, sin costura, transparente	“Las demoras en los procesos de ejecución en background se debe mostrar elegantemente.”	Traducir las expectativas del usuario en características del producto específicas.

Varios	"Cuando se detecten varios errores en el proceso de pedido se deben solicitar los cambios al área de desarrollo."	Definir cuantos específicamente, límites mínimos y máximos.
No debe	"El sistema no debe controlar el input del subsistema A."	Definir requerimiento desde lo positivo, o sea lo que el sistema debe hacer.
Suficiente	"Cuando las pruebas de integración fueron suficientes, los componentes pasan a las pruebas de sistema." "Se deben abrir las puertas del ascensor y deben permanecer así por un tiempo suficiente como para que el usuario pueda ingresar."	Especificar el valor de suficiente.
Soporten, habiliten	"El sistema debe soportar conexión remota." "El sistema debe habilitar servicios para acceso remoto."	Definir exactamente las funciones del sistema que hay que soportar o habilitar.

Identificación de requerimientos

_ Tareas que facilitan la identificación de requerimientos:

- Analizar los documentos "fuentes" (resultados de las reuniones con los usuarios).
- Extraer las sentencias que contienen las palabras: DEBERÁ, DEBE.
- Identificar sentencias que impliquen requerimientos.
- Identificar sentencias de tipo VERBOS, ACCIONES y un RESULTADO.
- Hacer una lista de requerimientos FUNCIONALES y NO FUNCIONALES.

Problemas con los requerimientos

_ El proceso de requerimientos no es fácil, no es simplemente "tomar nota" de las necesidades del cliente. Es un proceso de comunicación, por lo que es necesario entender que es lo que quiere el cliente. Además, es un proceso de negociación, es necesario determinar qué cosas podemos comprometernos a hacer. Dado esto, puede que aparezcan los siguientes problemas:

- No hay acuerdos en los requerimientos: puede ocurrir cuando no se hizo ningún tipo de negociación y no hay acuerdos reales entre los usuarios y desarrolladores, o porque no se ha realizado ningún tipo de verificación que posibilite determinar si los desarrolladores han comprendido las exigencias de los usuarios.
- Los requerimientos no se han priorizado: cuando no se ha hecho ningún estudio costo-beneficio, ni de ningún otro tipo que posibilite la definición de prioridades y/o cronogramas basados en los requerimientos.
- Requerimientos incompletos: el listado de los requerimientos no incluye cosas que son necesarias para que el software funcione.
- Requerimientos contradictorios: cuando unos requerimientos parecen contradecir a otros requerimientos. Si el software cumple a cabalidad con algunos requerimientos, no puede cumplir con los otros.
- Requerimientos ambiguos: cuando existen múltiples interpretaciones para el requerimiento. Cada usuario y/o desarrollador puede entender algo distinto (y puede no existir ningún acuerdo).
- Requerimientos de desarrollador: el desarrollador introduce requerimientos que no fueron solicitados por el cliente, y que hacen difícil satisfacer los requerimientos reales (presunciones de diseño).

_ Para solucionar estos problemas, debemos llevar un proceso disciplinado que busque determinar y solucionar los diferentes problemas de los requerimientos, y un sistema de especificaciones que posibilite comunicar y negociar los requerimientos eficientemente con los usuarios.

Proceso de ingeniería de requerimiento

Ingeniería de requerimientos

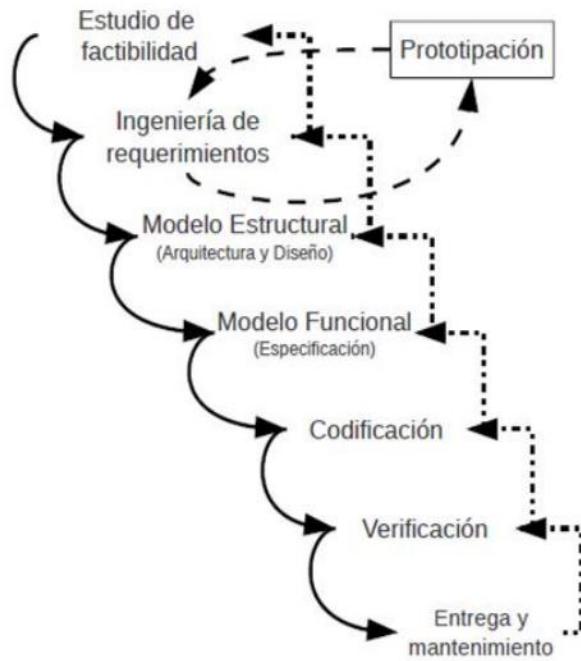
Concepto

Ingeniería de requerimientos: es el proceso para establecer los servicios que el sistema deberá proveer (requerimientos funcionales) y las restricciones bajo las cuales deberá operar y ser desarrollado (requerimientos no funcionales). Es un proceso cooperativo e iterativo en el cual se analiza el problema, documenta observaciones resultantes de una variedad de formatos de representación y chequea la precisión de la comprensión obtenida.

_ La ingeniería de requerimientos como tal, en este gráfico, está explotado en el ciclo de vida de cualquier proceso de desarrollo de software. En primer lugar, tenemos el estudio de factibilidad, que analiza qué tan factible es el proyecto tanto operativamente, como económico, etc. Después, empezamos con la etapa de ingeniería de requerimientos, la cual le agregó más velocidad, justamente, a la etapa de requerimientos. Antes, solo

teníamos la etapa de requerimientos, seguido del modelo estructural con una arquitectura y diseño, y el modelo funcional, codificación, verificación, entrega y mantenimiento. Esto se representa hasta la entrega y tiene relación entre las etapas mediante la iteración. Ahora, con la creación de la ingeniería de requerimientos, el área o etapa de requerimientos se convirtió en lo más clave a la hora de determinar errores en el desarrollo del software.

Figura 1 Esquema del ciclo de vida de cascada. La Ingeniería de Requerimientos es la segunda fase.



_ La ingeniería de requerimientos es la segunda fase estipulada en el ciclo de vida de cascada. Según este modelo de ciclo de vida, es necesario contar con los requerimientos para poder definir el modelo estructural del sistema, es decir, la arquitectura y el diseño. Sin embargo, usualmente es imposible contar con todos los requerimientos del sistema en un tiempo razonable, por lo que en la mayoría de los desarrollos se comienza a definir el modelo estructural teniendo en cuenta solo algunos requerimientos.

Requerimientos

Concepto

_ Un requerimiento es una característica del sistema o una descripción de algo que el sistema es capaz de hacer con el objeto de satisfacer el propósito del sistema, lo que ha sido apropiadamente documentado y validado por el solicitante. Los requerimientos tratan exclusivamente sobre los fenómenos del dominio de aplicación y no sobre la máquina que los implementa. No son descripciones de componentes de software, no son acciones concretas de los componentes de software del sistema, sino que plantean lenguaje del dominio.

_ Es importante resaltar algunos puntos de la definición anterior:

- No es lo mismo un pedido o deseo de un usuario o cliente que un requerimiento. No todos los pedidos o deseos de un usuario o cliente se convierten necesariamente en requerimientos, pero si todos los requerimientos se originan en un pedido o deseo de un usuario o cliente. Es decir, el usuario puede plantear requerimientos genéricos y somos nosotros quienes los tenemos que representar en la documentación formal de una manera en la que después la pueda interpretar el programador, arquitecto, etc.
- Para que un pedido o deseo de un usuario o cliente se convierta en requerimiento, este debe ser documentado apropiadamente y el solicitante debe validarlos. El solicitante después interpreta y valida lo que escribimos en la redacción técnica.
- Los ingenieros de software no originan los requerimientos; su función es convertir pedidos de los usuarios o clientes en requerimientos. Luego deben proveer un sistema que los implemente.
- Los requerimientos siempre están en el entorno del sistema que se va a desarrollar, nunca dentro de él. Por consiguiente, palabras tales como tabla, XML, clase, subrutina, etc, no pueden formar parte de un requerimiento.

Atributos de los requerimientos:

- Fecha en la que fue creado
- Número de versión
- Autor
- Prioridad
- Estado
- Origen
- Número de iteración

Requerimientos funcionales

_ Un requerimiento funcional describe una interacción entre el sistema y su ambiente. Los requerimientos funcionales describen como debe comportarse el sistema ante un estímulo. A continuación, vemos algunos ejemplos:

Sistema de control de ascensores:

1. Una persona que desee utilizar el ascensor para ir del piso n al n + m (con m > 0) debe pulsar el botón “Arriba” en el piso n, y en cuyo caso el ascensor eventualmente deberá detenerse en dicho piso.
2. Se deben abrir las puertas del ascensor y deben permanecer así por un tiempo suficiente como para que el usuario pueda ingresar.
3. Las puertas nunca deben abrirse a menos que el ascensor este detenido en un piso.

_ En este ejemplo tenemos muchos requerimientos, por ende, podríamos desglosarlo en más de uno. En el primer punto tenemos dos requerimientos, uno sobre la acción de apretar el botón para subir, y otro sobre cómo se debe desplazar el ascensor. En el segundo punto tenemos un solo requerimiento, y en el tercer punto también. La redacción está planteada en un lenguaje muy natural ya que son las funciones planteadas por el usuario.

Sistema de caja de ahorro de un banco:

1. Cada caja de ahorro se identifica por un número de cuenta.
2. Cualquier persona puede efectuar un depósito de una cantidad positiva de dinero en cualquier caja de ahorro.
3. Solo el titular o los cotitulares de una caja de ahorro pueden extraer una cantidad positiva de dinero de ella.

_ En este caso, los requerimientos están bastante claros y atómicos, y hay una conexión en el periodo de tiempo, pero de todas maneras se le pueda dar más precisión y detalle para que sean más complejos.

_ Como vemos se puede dar descripción de un requerimiento.

_ Tenemos que tener en cuenta que los requerimientos funcionales incluyen:

- Chequeos de validación de la entrada: validaciones de tipos de datos, validación de que se ingrese un tipo de dato concreto, etc.
- Secuencia exacta de las operaciones: esa secuencia por pasos como el trazo fino o el diagrama de casos de uso ayudan mucho y están basadas en las técnicas de escenarios, en donde identificamos escenarios que nos permitan guiar a los usuarios en cómo se va a ir llevando a cabo el proceso.

- Respuestas a situaciones anormales, incluyendo:
 - Desborde
 - Comunicaciones
 - Manejo de errores y recuperación
- Efecto de los parámetros del sistema.
- Relación entre las salidas y las entradas, incluyendo:
 - Secuencias de entrada/salida.
 - Fórmulas para convertir entrada en salida.

_ Tenemos la siguiente clasificación:

Requerimientos del usuario: deben describir los requerimientos funcionales y no funcionales de tal forma que sean comprensibles por los usuarios del sistema sin conocimientos técnicos detallados. Especifican el comportamiento externo del sistema y deben evitar las características de diseño del sistema.

Requerimientos del sistema: son versiones extendidas de los requerimientos del usuario y son utilizados por los ingenieros de software como punto de partida para el diseño de sistema; deben describir el comportamiento externo del sistema y sus restricciones operativas.

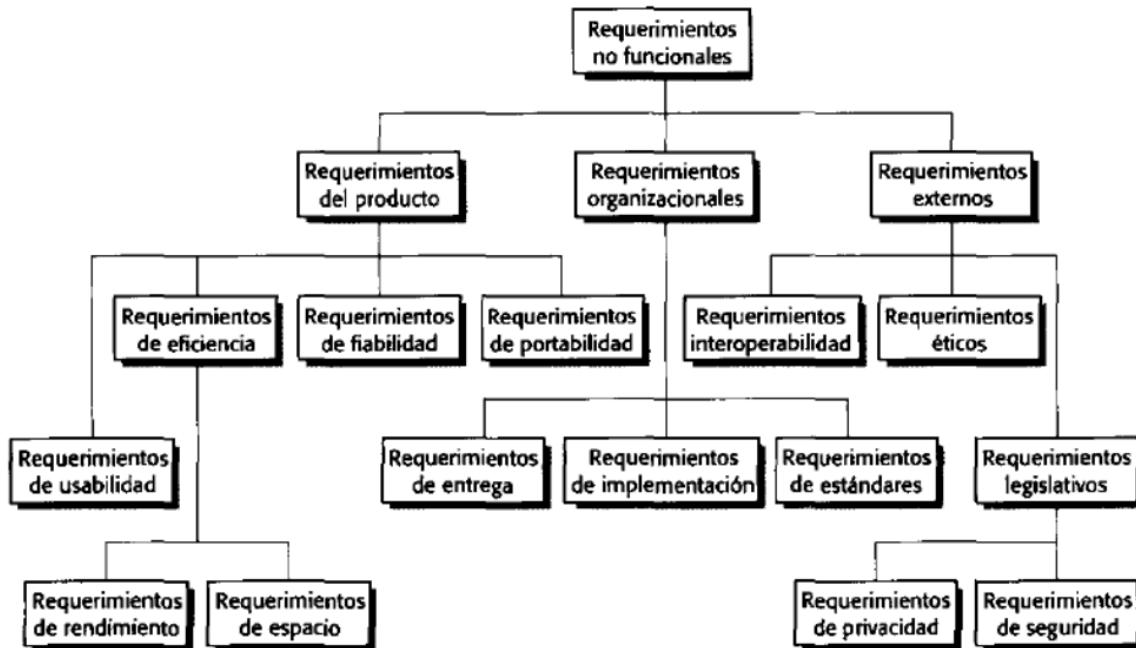
Requerimientos no funcionales

_ Los requerimientos no funcionales tienen que ver con representar atributos de calidad o cualidades del sistema. Un requerimiento no funcional es una restricción sobre el sistema o su proceso de producción. Si bien los requerimientos funcionales, o la función o funcionalidad del sistema, son esenciales para poder construir el sistema correcto, existen ciertas cualidades o atributos que los usuarios esperan del sistema, que no tienen una relación simple con la funcionalidad que desean. A estas cualidades o atributos se los llama requerimientos no funcionales. Estos no hacen a la generación del reporte, pero si tienen como respuesta a la tolerancia del otro lado de un requerimiento funcional.

_ Para redactar requerimientos no funcionales, la idea es que sean descripciones que estén asociadas a un requerimiento funcional. Algunos ejemplos son:

- El sistema debe ejecutar sobre Linux Ubuntu 10.4 LTS 10.04 (Lucid Lynx) y Windows 10.
- El sistema debe impedir que usuarios no autorizados accedan a información crítica.
- El sistema debe estar en funcionamiento 24/7.
- El sistema debe poder procesar 100.000 transacciones por hora.
- El sistema debe ser implementado en Java.

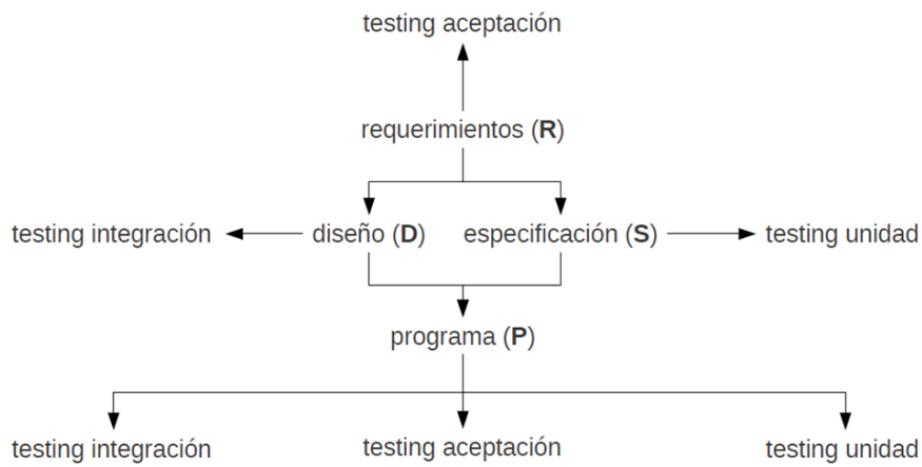
_ Tenemos la siguiente subclasificación:



Relación entre los requerimientos y el resto de las etapas del proceso

_ Teniendo en cuenta el siguiente grafico:

Figura 2 Relación entre requerimientos, diseño, especificación, programa y testing.



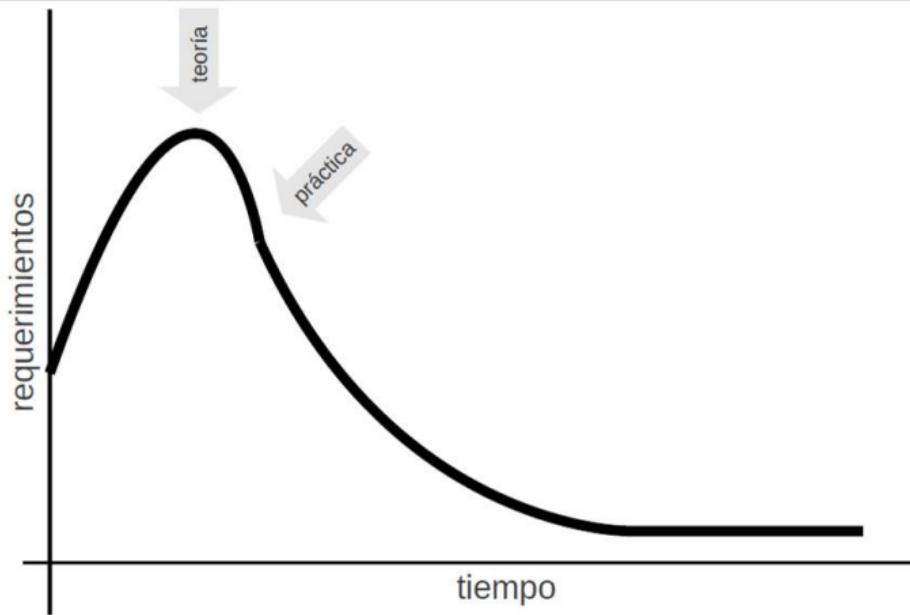
- El diseño y la especificación del sistema se obtienen a partir de los requerimientos.
- El diseño y la especificación se utilizan para implementar el programa.
- Finalmente, los requerimientos son el documento esencial para llevar adelante el testing de aceptación.

_ De estas relaciones surge claramente que la calidad de los requerimientos tiene una influencia enorme sobre el éxito o fracaso del proyecto, puesto que de ellos dependen los documentos claves del desarrollo. En consecuencia, cuanto mayor calidad tengan los requerimientos mayores son las posibilidades de que el proyecto sea un éxito.

_ Según el modelo de desarrollo de cascada, en el primer diagrama, a la fase de ingeniería de Requerimientos le sigue la de Arquitectura y Diseño. Cabe preguntarse, entonces, en qué momento de la ejecución de la primera es conveniente iniciar la segunda. Por ende, se podrá iniciar la fase de Arquitectura y Diseño (o la etapa de Definición de requerimientos) cuando se desacelera la cantidad de requerimientos validados que pide el cliente. Esto ocurre cuando el cociente entre la aparición de requerimientos y el tiempo es bajo durante un periodo de tiempo razonable. Cuando no tenemos más inestabilidad en cuanto al cambio de requerimientos podemos decir que estamos listos para prototipar, por lo que la prototipación pasa a ser parte de la ingeniería de requerimientos.

_ Tenemos la curva típica en la ingeniería de requerimientos, que representa la aparición de los requerimientos. Nosotros tenemos una maduración desde el momento que empezamos el proyecto, y en toda esta etapa vamos descubriendo requerimientos de forma iterativa con el cliente, y en algún momento los requerimientos ya no cambian y nos estamos familiarizando con el dominio y a partir de ahí, deberíamos buscar en esa línea requerimientos que deban ser consensuados con el cliente y luego plantear la parte de solución.

Figura 3 Curva típica que representa la aparición de requerimientos con respecto al tiempo en el período que media hasta la primera entrega.



Proceso de la ingeniería de requerimientos

_ Los procesos son fundamentales para las actividades del hombre. Un proceso es un conjunto de actividades organizadas que convierte entradas en salidas, y puede estar definido con diferentes niveles de detalle. Algunos ejemplos de procesos son:

- Un manual de instrucciones de un lavavajillas.
 - Un libro de recetas de cocina.
 - El manual de procedimientos de un banco.
 - Un manual de calidad de desarrollo de software.

_ En este caso las actividades que forman parte del proceso de la ingeniería de requerimientos son:

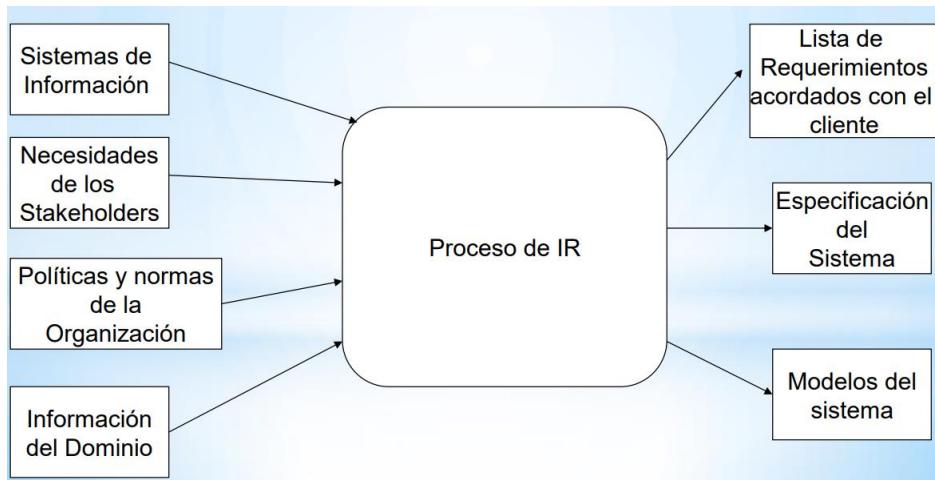
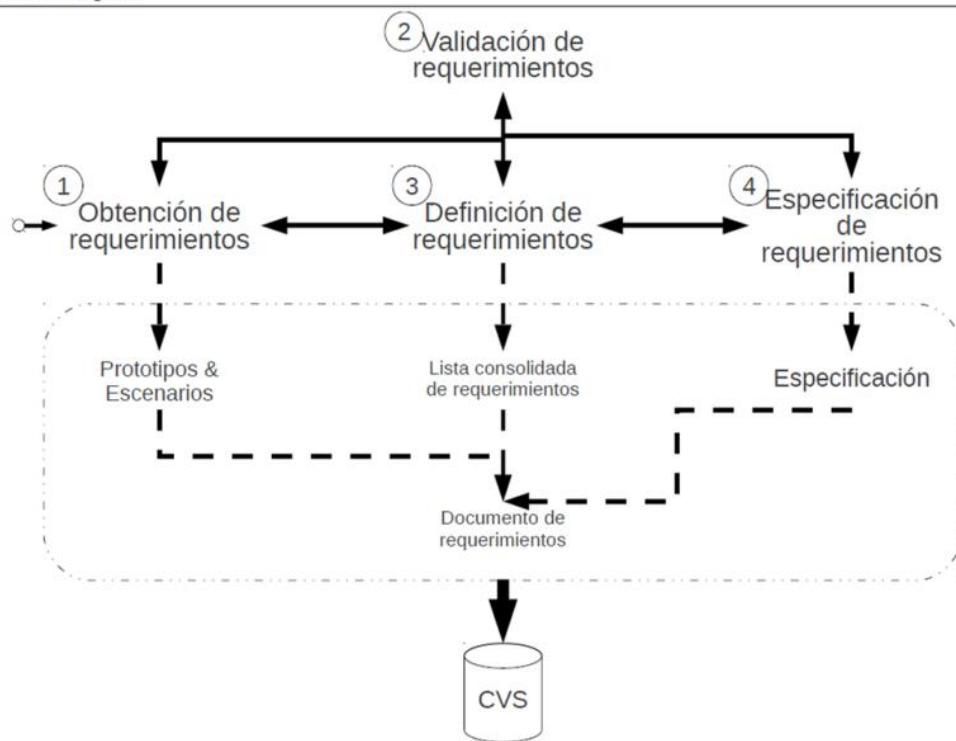


Figura 4 Proceso y productos de la Ingeniería de Requerimientos. Los números indican la secuencia más común entre las etapas.



_ Las siguientes etapas se van haciendo en paralelo con iteraciones y control de versiones. El proceso comienza por la obtención de requerimientos y finaliza con la redacción del documento de requerimientos. En todos los casos el término "requerimientos" refiere tanto a requerimientos funcionales como no funcionales.

_ Analizamos el grafico, en donde por un lado estamos trabajando con la técnica de casos de uso que nos permite, a medida que vamos incorporando más diagramas, entrar más en detalle, y por otro lado tenemos las técnicas que vienen de las disciplinas más administrativas como por ejemplo las entrevistas, cuestionarios, etc. A partir de esos requerimientos comenzamos una breve definición o representación de estos requerimientos para ayudar a la validación de estos que sería la segunda etapa del proceso de requerimientos. Todas las técnicas de diagramas de casos de uso y los diagramas de UML nos permiten determinar y validar si los requerimientos son consistentes y demás.

_ Antes de validar requerimientos, ya tenemos que tener un bosquejo y tenemos que haber aplicado todas las técnicas de obtención de requerimientos para tener todo el detalle posible del diseño. Tenemos un documento o una primera versión de una documentación de requerimientos. Entonces, un poco el objetivo es verificar que los requerimientos realmente definen el sistema que el cliente desea en el mismo. Todavía estamos en una etapa central en el desarrollo, de requerimientos y análisis, entonces no tenemos muy en claro el "cómo", sino que estamos tratando de que el cliente vea, en esta etapa de relevamiento, un modelo o bosquejo de lo que entendimos del planteo de esos requerimientos de funcionalidades o módulos. La importancia de esta etapa radica en evitar costos descubriendo errores en los requerimientos y no durante el desarrollo o después de que el sistema este en uso. Todas las actividades de la ingeniería de software son actividades de soporte, es decir, pensamos como podemos hacer para evitar que se presente algún error por desconocimiento del dominio y para eso trabajamos con ese nivel de rigurosidad y somos metódicos en todas las técnicas posibles para poder validar los requerimientos en esta etapa temprana.

Obtención de los requerimientos

_ En esta etapa se obtienen los requerimientos del sistema a través de la observación de sistemas existentes y del entorno donde se instalará el sistema mediante reuniones con los interesados, creación de lenguaje de modelados, prototipos y escenarios, y todo aquello que resuelva o ayude a descubrir más requerimientos. Estas técnicas deben aplicarse una y otra vez hasta que finalice la etapa.

_ Se trabaja con los clientes y los usuarios finales del sistema para determinar el dominio de la aplicación, servicios del sistema, rendimiento requerido del sistema, restricciones de hardware, etc. También debemos identificar e incluir a todos los "stakeholders".

_ Obtener y comprender los requerimientos de los stakeholders es un proceso difícil por varias razones:

- Los stakeholders a menudo no conocen lo que desean obtener del sistema informático excepto en términos muy generales; puede resultarles difícil expresar lo que quieren que haga el sistema o pueden hacer demandas irreales debido a que no conocen el coste de sus peticiones.
- Los stakeholders expresan los requerimientos con sus propios términos de forma natural y con un conocimiento implícito de su propio trabajo. Los ingenieros de requerimientos, sin experiencia en el dominio del cliente, deben comprender estos requerimientos.
- Diferentes stakeholders tienen requerimientos distintos, que pueden expresar de varias formas. Los ingenieros de requerimientos tienen que considerar todas las fuentes potenciales de requerimientos y descubrir las concordancias y los conflictos.
- Los factores políticos pueden influir en los requerimientos del sistema. Por ejemplo, los directivos pueden solicitar requerimientos específicos del sistema que incrementarán su influencia en la organización.
- El entorno económico y de negocios en el que se lleva a cabo el análisis es dinámico. Inevitablemente, cambia durante el proceso de análisis. Por lo tanto, la importancia de ciertos requerimientos puede cambiar. Puedenemerger nuevos requerimientos de nuevos stakeholders que no habían sido consultados previamente.

_ De esta manera obtenemos los modelos del sistema.

_ Tenemos muchas técnicas de requerimientos. La idea es saber cuál debemos adoptar y con cual nos sentimos más cómodos teniendo en cuenta el objetivo del uso de esa técnica. A continuación, tenemos:

- Informales
- Semiformales
- Formales
- Estructuradas
- Orientadas a objeto
- Algebraicas
- Lógicas,
- Procedurales
- Declarativas

_ Si estamos tratando de entender algún dominio complejo, a lo mejor deberíamos empezar por una más semiformal y lógica para plantear los escenarios.

Texto informal: en estos siempre trabajamos con lenguaje natural, y requiere revisión ya que se presenta mucha ambigüedad, por ende, hay términos que no se deben poner por más que estén en lenguaje natural. Se escribe mucho, no aprovechan que una imagen vale más que 1000 palabras. Proveen estructura y cierto grado de completitud. Se pueden complementar con notaciones gráficas.

Descubrimiento de requerimientos: es el proceso de recoger información sobre el sistema propuesto y extraer los requerimientos del usuario y del sistema. Las técnicas propuestas se siguen planteando en lenguaje natural, pero a la hora de la documentación la idea es que se trabaje con distintos niveles de detalle, por ejemplo, a veces necesitamos grabar la entrevista y después pasamos en limpio. Todas estas técnicas juntas nos van a ayudar a hacer una documentación de requerimientos:

- Entrevistas
- Observación
- Escenarios
- Prototipos

Técnicas de identificación de requerimientos: pasamos por las siguientes etapas, que son válidas a la hora de identificar requerimientos:

- Extracción de información: consiste en obtener de los usuarios “lo que ellos saben y nosotros no”. Se supone que las metodologías ágiles involucran mucho al cliente en el equipo de desarrollo, entonces sumamos al cliente al equipo de desarrollo y en esa convivencia nos enteramos mucho más del dominio, pero el cliente no tiene en realidad esa disponibilidad para trabajar con el equipo de desarrollo por lo que debemos ser precisos a la hora de seleccionar las técnicas puntuales para poder captar todo el conocimiento que tiene el cliente, por ende, esto es un proceso de captura donde todo lo que se obtiene se registra y de manera gráfica también. Es un proceso de comunicación e interacción y es importante observarlos en su “hábitat natural”.
- Identificación de stakeholders: es decir, cualquier persona o grupo que se verá afectado por el sistema directa o indirectamente. Desde los usuarios finales que interactúan con el sistema, y todos aquellos que se pueden ver afectados por su instalación. Personas que tienen influencia suficiente como para cambiar los alcances del sistema, definir requerimientos no funcionales, etc. A continuación, tenemos ejemplos de lo que sería un stakeholder:
 - Ingenieros que desarrollan sistema relacionados, gerentes del negocio, expertos del dominio del sistema, usuarios finales, inversores, etc.
- Identificación de usuarios, identificar sectores perjudicados: un punto clave es categorizar los usuarios, ya que el usuario y el cliente en general, no son el mismo. El usuario es el que va a usar el sistema y el cliente es el que tiene la motivación de

obtener el éxito del proyecto. También, tenemos clientes de la organización, entidades externas como entes reguladores, y también pueden ser usuarios perjudicados, nuevos usuarios, analistas, desarrolladores, programadores, etc.

- Preguntas del proceso libres del contexto.
- Meta preguntas, preguntar sobre esas mismas preguntas.
- Brainstorm o “tormenta” de ideas.
- Reuniones de relevamientos.

Técnicas de relevamiento:

- Entrevistas: proporcionan una visión general de lo que hacen los stakeholders, como podrían interactuar con el sistema y los problemas con los que se enfrentan los sistemas actuales. Tenemos que saber estructurar esa entrevista para poder determinar que queremos obtener de la misma. Hay un proceso de desarrollo de entrevistas:
 - Definición de objetivos: tenemos que definir que queremos entender y no hacerla más de una hora.
 - Planificación: tiene que ver con elegir el entrevistado en función del conocimiento para lograr empatizar.
 - Concreción: hacemos referencia que a la entrevista la preparamos unos días antes y le avisamos de antemano al entrevistado.
 - Realización: tenemos que respetar el tiempo de duración en respeto a la otra persona que estamos entrevistando.
 - Evaluación: resumimos la información abordada en la entrevista para mostrársela al entrevistado para que sugiera cambios.
- Escenarios: son ejemplos de una situación de la vida real. Agregan detalle a la lista de requerimientos describiendo sesiones de interacción. En los trazos finos, tenemos que el actor hace y el sistema responde, en donde el requerimiento es una caja negra y a partir de ahí hay una interacción con el usuario y esa caja negra, y vamos visualizando como va a ser la dinámica entre el usuario y el sistema. El escenario va un poco más allá, pero también plantea una interacción, es decir, el paso a paso de una actividad. Un escenario incluye:
 - Una descripción de lo que esperan el sistema y los usuarios cuando el escenario comienza.
 - Una descripción del flujo normal de eventos en el escenario.
 - Una descripción de lo que puede ir mal y cómo manejarlo, lo que serían los flujos alternativos.
 - Información de otras actividades que se podrían llevar a cabo al mismo tiempo.
 - Una descripción del estado del sistema cuando el escenario termina.
- Casos de uso: los casos de uso identifican las interacciones particulares con el sistema.

- Etnografía: es una técnica de observación que se puede utilizar para entender los requerimientos sociales y organizacionales. El analista se introduce por sí solo en el entorno laboral donde se utilizará el sistema y descubre requerimientos implícitos que reflejan los procesos reales. Son buenas para analizar el comportamiento de un usuario con un sistema, por ejemplo, una página web en donde puede surgir el tema de que la gente mayor tiene dificultad para ver, y tenemos que pensar cómo resolver este problema. Esta técnica está asociada a como analizar el comportamiento del uso del sistema. Se descubren requerimientos que surgen de la relación de un trabajo con otro. Esta técnica es efectiva para descubrir dos tipos de requerimientos:
 - Los que se derivan de la forma en la que la gente trabaja realmente.
 - Los requerimientos que se derivan de la cooperación y conocimientos de las actividades de la gente.
- Prototipos: es una versión inicial del sistema que se realiza en etapas tempranas del proceso de desarrollo para inferir requerimientos omitidos.
- Estudio de documentación: varios tipos de documentación, como manuales y reportes, pueden proporcionar al analista información valiosa con respecto a las organizaciones y a sus operaciones. No siempre tenemos documentación, pero si hay bienvenida sea. La documentación difícilmente refleja la forma en que realmente se desarrollan las actividades, o donde se encuentra el poder de la toma de decisiones. Sin embargo, puede ser de gran importancia para introducir al analista al dominio de operación y el vocabulario que utiliza.
- Léxico extendido del lenguaje (LEL): es un conjunto de términos provenientes del lenguaje de la aplicación, donde se identifica la semántica de cada término. Está formado por un conjunto de símbolos, y los símbolos son, en general, las palabras o frases utilizadas por el usuario y que repite con más frecuencia. O salen del dominio o están repetidas en las entrevistas por varios usuarios, entonces a uno le llaman la atención. Podría ser como un glosario, pero tiene un poco más de detalle y se usa para dominios complejos. Por ejemplo, un nombre puede llegar a ser una acción a llevar a cabo y puede ser que en la descripción del dominio no se tenga en cuenta. Cada símbolo tiene:
 - Un nombre que lo identifica.
 - Una noción que indica que es el símbolo
 - Un impacto que indica como repercute en el sistema.
- Puntos de vista: la técnica de POV consiste en enfocarse en las necesidades y los deseos de los usuarios para organizarlas y encontrarles soluciones innovadoras. Cada usuario tiene un punto de vista distinto, usa el sistema de una forma distinta o tiene necesidades distintas con el sistema, entonces es importante considerarlas. Cualquier sistema de software no trivial debe satisfacer las necesidades de un grupo diverso de interesados (stakeholders). Cada uno de estos puede tener intereses diferentes en el sistema de software, y por lo tanto sus necesidades

pueden generar requerimientos que tengan conflicto entre sí, o incluso se contradicen. Los métodos orientados a puntos de vista toman en consideración estas perspectivas diferentes y las utilizan para estructurar y organizar tanto el proceso de obtención, como los requerimientos mismos. Los elementos de dicha técnica son:

- Usuario: persona que disfruta de un servicio o del empleo de un producto.
- Necesidad: puntos a satisfacer por parte de un producto.
- Un estado de carencia percibida.
- Insight: motivación.

Por ejemplo, las personas ancianas (usuario) necesitan teléfonos grandes (necesidad) porque tienen dificultades para distinguir las teclas (insight).

- JAD (Joint Application Development): una alternativa a las entrevistas individuales y se desarrolla a lo largo de un conjunto de reuniones en grupo durante un periodo de 2 a 4 días. En estas reuniones se ayuda a los clientes y usuarios a formular problemas y explorar posibles soluciones, involucrándolos y haciéndolos sentirse partícipes del desarrollo. Esta técnica se base en cuatro principios [Raghavan]:
 - Dinámica de grupo.
 - Uso de ayudas visuales para mejorar la comunicación (diagramas, multimedia, etc).
 - Mantener un proceso organizado y racional.
 - Una filosofía de documentación WYSIWYG (What You See Is What You Get, lo que se ve es lo que se obtiene), por la que durante las reuniones se trabaja directamente sobre los documentos a generar.

Estrategia sugerida: aprender todo lo que se pueda de los documentos, formularios, informes y archivos existentes. Es sorprendente lo que se puede aprender de un sistema sin necesidad de quitarle tiempo a la gente. A veces sucede que no podemos lograr la entrevista con el experto, por ende, tenemos que ver cuáles son los otros elementos para no perder tiempo.

_ De ser posible, se observará el sistema en acción. No se plantearán preguntas. Tan sólo se observará y se tomarán notas o dibujos. Conviene asegurarse de que las personas observadas saben que no se les está evaluando. En caso contrario, harán su trabajo de manera más eficaz que lo normal. Todo esto sucede cuando a veces al tener que desarrollar un sistema que tiene que ver con el reemplazo de algo existente, por ejemplo, cuando tenemos una nueva versión de algo o cambia algo haciendo que tenga más complejidad en su proceso, donde muchas veces se observa cómo se trabaja de más.

_ Diseñar y distribuir cuestionarios para aclarar aspectos que no se comprenden bien. Será también buen momento para solicitar opiniones sobre los problemas y las limitaciones. Los cuestionarios requieren que los usuarios inviertan una parte de su tiempo. Pero son ellos los que pueden elegir cuándo les viene mejor hacerlo.

_ Realizar entrevistas (o sesiones de trabajo en grupo, como JAD). Como ya se ha recogido una base de requerimientos iniciales en los pasos anteriores, se pueden utilizar las entrevistas para verificar y aclarar las cuestiones y los problemas de mayor dificultad. En este punto se pueden llegar a aplicar algunas de las otras técnicas como Escenarios, Tormenta de ideas, Puntos de Vista, ETHICS y Desarrollo de Prototipos.

_ Se verifican los requerimientos a través del uso de técnicas como entrevistas, observación y orientados a puntos de vista.

Análisis de requerimiento y negociación:

_ Todas estas técnicas son actividades que se incluyen en el proceso de IR (ingeniería de requerimientos), para descubrir problemas en la etapa de elicitación para luego pasar a la etapa de documentación formal de los requerimientos. La etapa previa a la documentación es la negociación. Por otro lado, usar un checklist de esa documentación de requerimientos que me permita validar los requerimientos en sus aspectos características de calidad.

- Checklist: se realizan una lista de preguntas para verificar la calidad de los requerimientos.

_ Tenemos una serie de preguntas para validar esos requerimientos y ver si hace falta agregarle algún tipo de información. Entonces para saber si es necesario se hacen preguntas como si tiene toda la información necesaria, si es claro, si es preciso, si tiene una única implementación, etc. Todas estas técnicas que usamos para validar requerimientos o para saber si estos cumplen con los atributos de calidad, se llevan a una herramienta como un Excel para poder aplicar la herramienta de evaluación.

_ Muchas veces hay requerimientos que están relacionados o no, algunos parecen tener caminos distintos, pero son los mismos, en donde todo este descubrimiento que se hace en esta etapa de análisis de requerimientos se logra muchas veces con una:

- Matriz de iteración: un objetivo importante es tratar de relacionar los requerimientos para determinar si hay algún tipo de conflicto, solapamiento o independencia.

_ Todos los sistemas grandes y complejos tienen muchos stakeholders y con ellos hay que negociar. Se organiza una reunión con los stakeholders para lograr la negociación. Existen tres etapas:

- Etapa en donde el analista expone los problemas asociados con los requerimientos: orden de prioridad, requerimientos innecesarios, requerimientos ambiguos, etc.
- Etapa de discusión donde los stakeholder se ponen de acuerdo como resolver los problemas.

- Etapa de resolución donde se especifica que se hace con los requerimientos que generaron problemas, cuales se eliminan, cuales se modifican y como.

_ Tenemos que tener en cuenta los cambios en la gestión de requerimientos. De todo esto nos llevamos las técnicas que se utilizan en la obtención y el análisis de requerimientos, en esta etapa de priorización y negociación para que podamos valorar cuales son los requerimientos de peso que seguramente van a garantizar el éxito del proyecto. Tenemos que destacar la extracción de información, la importancia de los stakeholders, la importancia de identificar a los usuarios, reuniones de relevamientos, técnicas de relevamientos, estrategia sugerida y la negociación de requerimientos.

Validación de requerimientos

_ Ubicándonos en las cuatro actividades del proceso de desarrollo, una vez que obtenemos los requerimientos y tenemos un bosquejo con algunos modelos, podemos empezar a validarlos. Los requerimientos solo pueden ser validados por el cliente. Por lo tanto, una vez que se ha obtenido un requerimiento (preguntando al cliente) se le debe preguntar si ese es el requerimiento que expreso. Esos requerimientos están asociados con la validación de los mismos, donde a esa validación la podemos hacer nosotros con el cliente y podemos ir viendo si estamos logrando la comprensión de esos requerimientos.

_ La validación de requerimientos trata de mostrar que los requerimientos definen realmente el sistema que el cliente desea. También, durante el proceso de validación de requerimientos, se deben llevar a cabo verificaciones sobre requerimientos en el documento de requerimientos. Por ende, obtenemos el documento de requerimientos validado.

_ Verifica que los requerimientos realmente definen el sistema que el cliente desea. La importancia de esta etapa radica en evitar costos descubriendo errores en los requerimientos y no durante el desarrollo o después de que el sistema este en uso.

Proceso de validación de requerimientos:

- Verificaciones de validez: aquí surgen funciones adicionales o diferentes a las planteadas. Entonces, nosotros estamos validando o leyendo algún requerimiento y surgen algunas situaciones excepcionales como, por ejemplo, utilizando la técnica del trazo fino de los casos de uso me permite identificar algunos flujos alternativos de acuerdo a los casos excepcionales que planteemos. Hacer estas verificaciones de validez implica entrar en detalle y poder hacer que el sistema tenga un requerimiento previo que valide previamente a hacer tal cosa.
- Verificaciones de consistencia: en el documento de requerimiento no debe haber restricciones o descripciones contradictorias de la misma función del sistema. Muchas veces lo que hacemos en la escritura de un requerimiento lo contradecimos en el siguiente, por ahí el proceso es complejo y muestra que al

relevamiento lo hicieron dos personas distintas del equipo de trabajo donde uno hizo una cosa y el otro otra, o claramente hay una ambigüedad o algo que no se termina de entender, por lo cual hemos escrito requerimientos inconsistentes. La inconsistencia en los requerimientos se suele dar cuando uno los empieza a redactar en un proceso largo.

- Verificaciones de completitud: el documento de requerimiento debe incluir requerimientos que definan todas las funciones y restricciones propuestas por el usuario. Nosotros definimos una estructura de redacción en donde comenzamos con que el sistema "debe" luego redactamos el sujeto, el objeto y cuál sería su función y en base a eso ya tenemos una estructura para asegurarnos la completitud del documento o del requerimiento, la idea es que estén planteadas y contempladas las restricciones.
- Verificaciones de realismo: de acuerdo a la tecnología existente, los requerimientos deben verificarse para asegurar que se pueden implementar, también se debe tener en cuenta el presupuesto y la agenda. Esta verificación esta atravesada por cuestiones de gestión.
- Verificabilidad: esto se logra si por cada requerimiento se puede escribir un conjunto de pruebas que demuestren que el sistema a entregar cumple con los requerimientos especificados. La verificabilidad de un requerimiento redactado escrito se valida mucho con los prototipos porque el cliente de alguna manera visualiza de una forma más rápida la interacción que hace con el sistema y aquellos requerimientos que están cuantificados, como por ejemplo los requerimientos no funcionales, son los requerimientos que se pueden verificar o que tienen esta propiedad de verificabilidad. Esta propiedad nos ayuda para hacer los casos de prueba, donde nos basamos en ese requerimiento bien redactado para poder determinar cómo hacer la prueba que me valide o certifique que ese requerimiento cumple con lo que el cliente está esperando.

Técnicas de validación de requerimientos: para llevar a cabo las verificaciones anteriores se utilizan técnicas, como en todas las prácticas de ingeniería de software, y son las siguientes:

- Revisión de requerimientos: los requerimientos son analizados sistemáticamente por un conjunto de revisores. La revisión es un proceso manual que involucra a varios lectores, tanto del cliente como del contratista. Puede ser formal o informal. En la revisión formal el equipo de desarrollo "conduce" al cliente a través de los requerimientos, explicándole las implicaciones de cada uno. Los conflictos, contradicciones, errores y omisiones deben señalarse durante la revisión y registrarse formalmente. Los revisores comprueban:
 - Consistencia
 - Integridad
 - Verificabilidad

- Comprendibilidad
- Rastreabilidad
- Construcción de prototipos: los usuarios pueden experimentar con este modelo para ver si cumple con las necesidades reales. Algunas actividades son:
 - Elegir las personas que probaran el prototipo.
 - Desarrollar los escenarios a probar.
 - Ejecutar los escenarios.
 - Documentar los problemas.
- Generación de casos de prueba: los requerimientos deben poder probarse. Se diseñan las pruebas antes de la codificación de los requerimientos. Algunas actividades son:
 - Planificar las pruebas.
 - Diseñar los procedimientos de prueba.
 - Diseñar los casos de prueba.
 - Ejecutar las pruebas.
 - Documentar resultados.

Definición de requerimientos

_ En el momento que logramos poder escribir algo o definir requerimientos, los vamos validando. En esta etapa la información colectada en la etapa anterior se vuelca en un documento consolidado, organizado y estructurado. Las técnicas aplicadas para descubrir requerimientos generan varios prototipos y escenarios, la información recabada tiende a estar dispersa y desorganizada, por lo que es necesario consolidarla y organizarla.

Priorización de requerimiento:

- Requerimientos que deben ser absolutamente satisfechos.
- Requerimientos que son muy deseables, pero no indispensables.
- Requerimientos que son posibles, pero que podrán eliminarse.

Calidad de los requerimientos: la calidad de un requerimiento se refiere en general a que tan perdurable en el tiempo será tal y como esta descripto en este momento. Un requerimiento es de mala calidad si el cliente no puede expresarlo claramente o directamente duda o desconoce qué es lo que realmente necesita o quiere. La calidad de un requerimiento está directamente relacionada con el grado de validación que aún necesita.

Atributo	Comentario
Cohesivo	El requerimiento habla de una única cosa.
Completo	El requerimiento está enunciado en un único lugar y no falta información.
Consistente	El requerimiento no contradice ningún otro requerimiento y es consistente con cualquier otra información relevante.
Correcto	El requerimiento apunta a soportar el negocio detrás del sistema como fue enunciado por el cliente.
Actualizado	El requerimiento no se volvió obsoleto por el paso del tiempo.
Observable externamente	El requerimiento habla de una característica del sistema que es observable o experimentada por el usuario. Los “requerimientos” que hablan de la arquitectura, diseño o implementación no son, en realidad, requerimientos.
Factible	El requerimiento puede implementarse dentro de los límites del proyecto.
No ambiguo	El requerimiento está enunciado de forma clara y concisa utilizando el vocabulario del dominio de aplicación, habiéndose definido con precisión todos los términos técnicos y siglas. El enunciado tiene una única interpretación. Sustantivos, adjetivos preposiciones, verbos y frases subjetivas que resulten vagas o poco claras deben ser evitadas. Se prohíbe el uso de oraciones negativas o compuestas.
Validable	La implementación del requerimiento puede validarse a través de inspección, análisis, demostración o prueba (test).

- El requerimiento no necesita más validación.
- El requerimiento debe ser validado por al menos un interesado más.
- El requerimiento debe ser validado por todos los interesados.

Especificación de requerimientos

_ Es el planteamiento formal de las etapas anteriores. En pocas palabras en esta etapa la lista consolidada de requerimientos se escribe desde la perspectiva del desarrollador. Esta etapa puede incluirse en las fases denominadas modelo estructural o modelo funcional en el modelo de ciclo de vida de cascada. En esta etapa se debe lograr una documentación de los requerimientos con un apropiado nivel de detalle. Obtenemos la lista de requerimientos del usuario del sistema.

Productos del proceso de la ingeniería de requerimientos

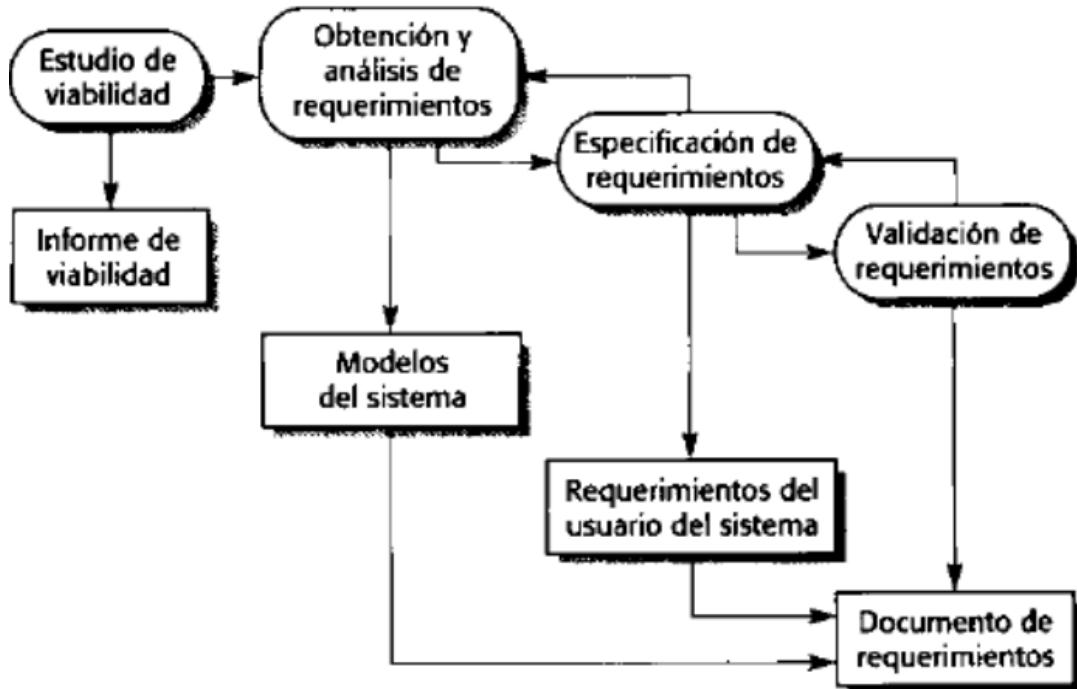
Prototipos: debe ser la forma de interacción casi excluyente entre los ingenieros de requerimientos y el cliente, durante esta fase. Nos referimos concretamente a prototipos desecharables. Un prototipo desecharable es un programa que imita al menos algunos de los requerimientos funcionales pero que luego de que los requerimientos se han validado, el programa se descarta.

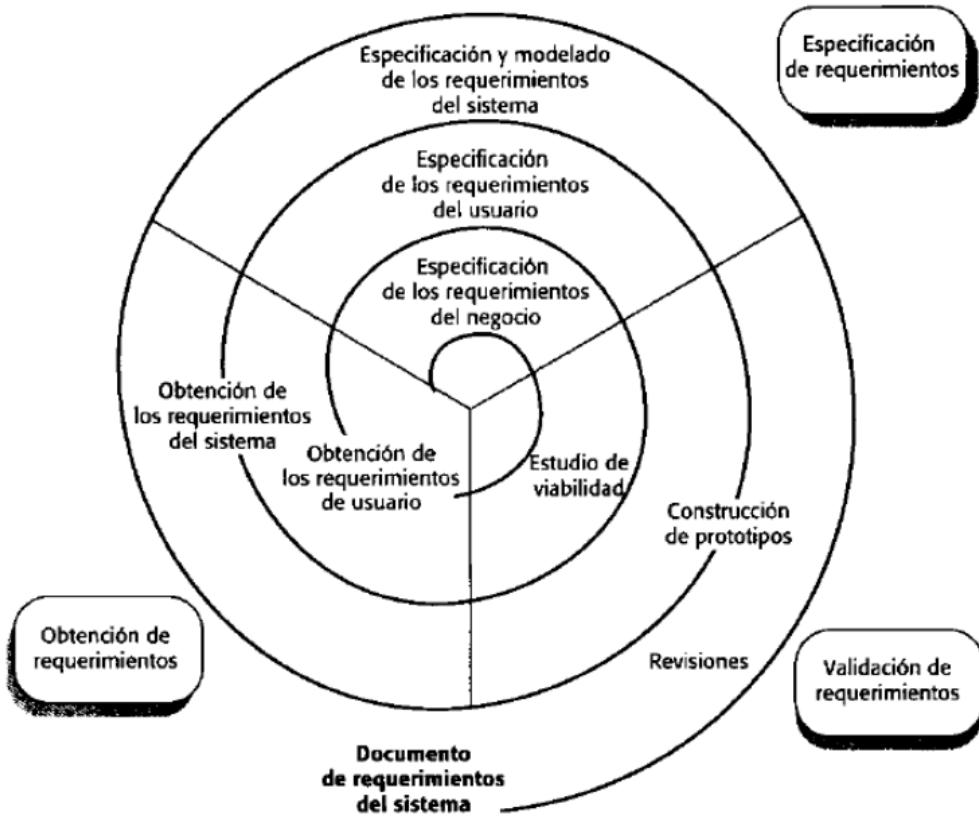
Escenarios: es muy complejo o costoso, y muchas veces imposible, prototipar los atributos de calidad de un sistema (tales como disponibilidad, seguridad, testeabilidad, etc). En consecuencia, en lugar de prototipos, se utilizan escenarios de atributos de calidad como técnica efectiva para determinar con suficiente precisión cual es el significado que el usuario les asigna a las cualidades del sistema.

Lista consolidada de requerimientos: es un documento que consolida, organiza y estructura la información recolectada durante la etapa de obtención de requerimientos.

- El vocabulario y las notaciones que se utilicen para escribir esta lista deben ser estrictamente las del cliente o las del dominio de aplicación no deberán incluir términos tales como tabla, registro, campo, lista enlazada, función, proceso o procedimiento (en su acepción informática), variable, base de datos, archivo, directorio, sistema operativo, modulo, objeto, clase, hilo de control, servidor, HTTP, web service, etc.
- Frases tales como el “sistema deberá implementar un módulo...” o un web service tomara los datos de la tabla personas y los enviara vía SOAP al servidor Web...” están totalmente fuera del alcance de este documento.
- Frases tales como “el sistema deberá implementarse en C++...” o “el sistema deberá procesar 10.000 transacciones por minuto...” pueden estar dentro del alcance de este documento, como requerimientos no funcionales.

Proceso de ingeniería de requerimientos / Modelo en espiral





Elicitacion de requerimientos: significa descubrir requerimientos del sistema utilizando:

- Entrevistas con los stakeholders
- Documentación del sistema
- Conocimiento del dominio
- Estudio de mercado

Análisis y negociación: los requerimientos son analizados en detalle y los stakeholders se ponen de acuerdo para determinar los requerimientos válidos.

Documentación de requerimientos: se documentan los requerimientos acordados utilizando lenguaje natural y diagramas que faciliten la comprensión de todos los stakeholders.

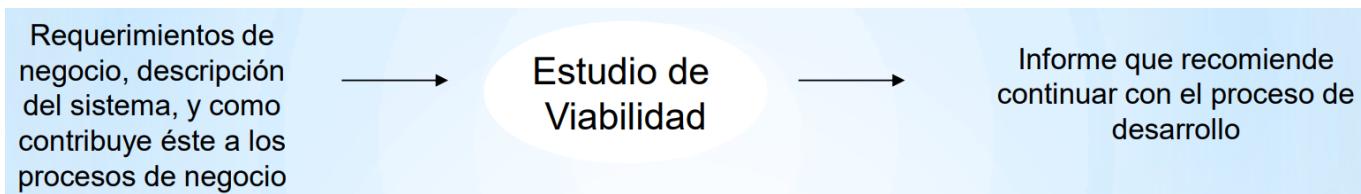
Actores en el proceso de ingeniería de requerimientos

Actores → Roles

_ Esta identificación de actores en base al role que asumen en el proceso es útil cuando entramos en el detalle de cada proceso. Algunos ejemplos:

- Expertos del dominio
- Usuarios finales del sistema
- Ingenieros de requerimientos
- Ingenieros de software
- Líderes de proyecto

Estudio de viabilidad



_ Hay que resolver varias cuestiones:

- Contribución del sistema a los objetivos generales de la organización.
- Si se puede implementar el sistema utilizando la tecnología actual y dentro de las restricciones coste tiempo.
- Si el sistema se puede integrar con otros sistemas ya existentes en la organización.

Entrada: recopilación y evaluación de información.

_ Preguntas para la evaluación de entrada:

- ¿Cómo se arreglaría la organización si no se implantara el sistema?
- ¿Cuáles son los problemas con los procesos actuales y cómo ayudaría a aliviarlos el nuevo sistema?
- ¿Cuál es la contribución directa que hará el sistema a los objetivos y requerimientos del negocio?

Actores: jefes de departamentos donde se utilizará el sistema, pares que conozcan el tipo de sistema propuesto, expertos en tecnología, usuarios finales.

Tiempo estimado: 2 o 3 semanas.

Salida: informe del estudio de viabilidad.

- Recomendar si se debe continuar o no con el desarrollo del sistema.
- Proponer cambios en el alcance, el presupuesto y confección de la agenda.
- Sugerir requerimientos adicionales.

Administración de requerimientos

_ La gestión o administración de requerimientos es el proceso de comprender y controlar los cambios en los requerimientos. Hay que establecer un proceso formal para implementar las propuestas de cambios y vincular estos a los requerimientos del sistema. El proceso de gestión de requerimiento debería empezar en cuanto esté disponible una versión preliminar del documento de requerimientos.

Razones para administrar requerimientos: a continuación, vemos las razones por las que debemos gestionar o administrar los requerimientos:

- Los requerimientos para sistemas de software son siempre cambiantes. En el tránsito de la vida del sistema los requerimientos pueden cambiar, pueden modificarse, pueden generar modificaciones que dependan de las reglas del negocio y que tienen que ser presentadas en el sistema, entonces es importante que identifiquemos cuáles son esos requerimientos que tienen más posibilidades de modificarse durante el proceso de desarrollo y tener una trazabilidad de los mismos.
- Debido a que el problema no puede definirse completamente, los requerimientos de software son incompletos. La comprensión o el conocimiento del dominio, que a medida que van pasando las entrevistas y a medida que vamos modelando las diferentes soluciones con diferentes perspectivas, vamos entendiendo un poco más lo que queremos representar y también eso va generando nuevas versiones del requerimiento. La idea es quedarnos con la versión más completa y que cumpla con todas las condiciones y atributos de calidad que mencionamos anteriormente.
- La comprensión del problema por parte de los stakeholders está cambiando constantemente a lo largo del proceso de desarrollo. La comunicación con los expertos del dominio tiene que ser constante, por eso es un proceso de relevamiento, descubrimiento y validación que se da permanentemente para poder lograr que el requerimiento quede lo más completo posible, y en ese discurso los stakeholders van modificando estos requerimientos.
- Cuando los usuarios finales tienen experiencia con el sistema descubren nuevas necesidades y prioridades.
- Los clientes del sistema imponen requerimientos debido a las restricciones organizacionales y de presupuesto; y estos entrar en conflicto con los requerimientos de los usuarios finales.

Requerimientos duraderos y volátiles:

_ Todos los requerimientos deben tener un seguimiento y una trazabilidad de los modelos que generan, por lo tanto, hay que identificar estos requerimientos que si o si van a cambiar o que es muy probable que cambien, porque están ligados a las políticas gubernamentales, alguna ley de turno, a aspectos que tengan que ver con lo no funcional

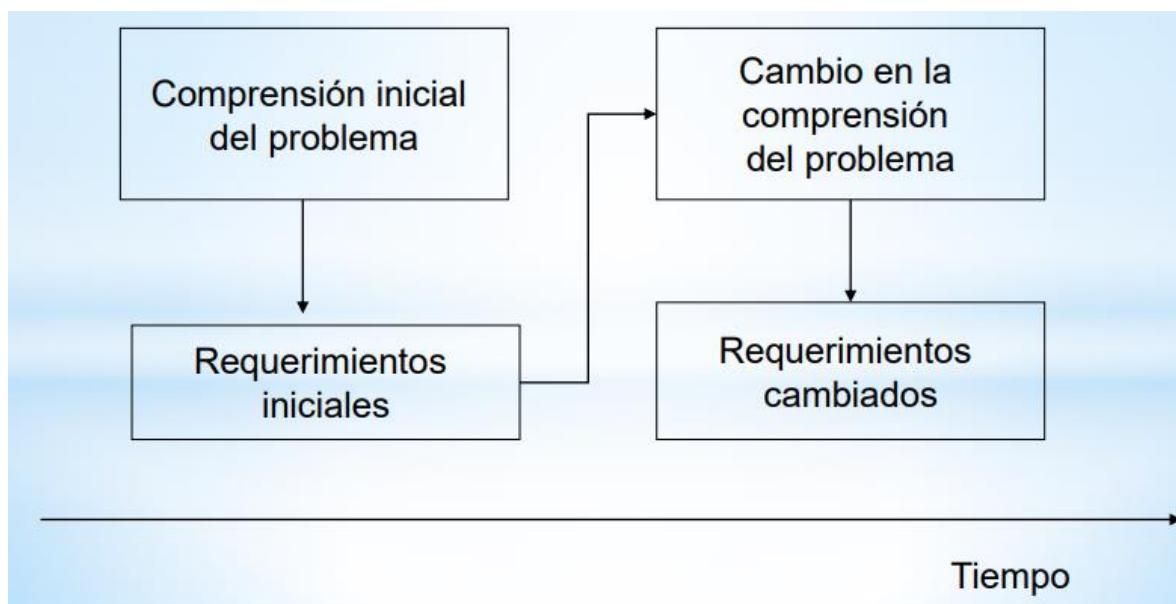
asociado a los procedimientos del negocio, también asociadas a alguna tecnología o servicio que pudiera dejar de estar disponible, etc. Entonces, estos requerimientos que sabemos que pueden llegar a cambiar con más veracidad que aquellos que son parte del sistema, son sobre los cuales tenemos que hacer énfasis, porque la idea es poder modificarlos de manera ágil. Desde una perspectiva evolutiva, los requerimientos se dividen en dos clases:

- Requerimientos duraderos: son relativamente estables que se derivan de la actividad principal de la organización. Por ejemplo, en un hospital requerimientos que se refieran a los pacientes, médicos, enfermeras y tratamientos. Estos requerimientos se pueden derivar del modelo de dominio.
- Requerimientos volátiles: estos probablemente cambian durante el proceso de desarrollo del sistema o después de que este se haya puesto en funcionamiento. Por ejemplo, requerimientos resultantes de las políticas gubernamentales sobre sanidad. Estos requerimientos se clasifican en:
 - Requerimientos cambiantes: los requerimientos cambian debido a los cambios en el entorno en el que opera la organización. Por ejemplo, alguna política del negocio o política organizacional que se sostenía durante un largo tiempo y luego cambia (sistemas de monotributo).
 - Requerimientos emergentes: los requerimientos que emergen al incrementarse la comprensión del cliente en el desarrollo del sistema. El proceso de diseño puede revelar requerimientos emergentes nuevos. Estos requerimientos son los que surgen cuando teníamos una versión del requerimiento más o menos entendido por el cliente, ya que este muchas veces no tiene muy en claro cómo se va a llevar a cabo, y que luego una vez que entendió la perspectiva y le mostramos las validaciones e hicimos un prototipo, el cliente cambia de parecer y nos dice que ahora que el sistema hace tal cosa debería también hacer otra, es decir, son requerimientos que van surgiendo, pero parten del desconocimiento. Esto pasa en etapas tempranas.
 - Requerimientos consecuentes: los requerimientos que son el resultado de la introducción del sistema informático, esto puede cambiar los procesos de negocio, las formas de trabajo que generan nuevos requerimientos. Por ejemplo, el sistema antes permitía cargar los productos por teclado y ahora se implementó un lector óptico, pero resulta que este lector tiene algunas fallas y por ende puede que tengamos que generar un requerimiento consecuente para validar el numero cargado, puede que surjan otros requerimientos o no.
 - Requerimientos de compatibilidad: estos requerimientos dependen de sistemas particulares o procesos de negocios dentro de la organización. Tenemos que ver que los requerimientos que generamos, para validar la compatibilidad que se hace en un momento que tiene un tiempo establecido, no contradigan a algún otro requerimiento. Siempre existen situaciones en

donde el cliente quiere por ejemplo durante un periodo de tiempo sacar alguna promoción, entonces a esos requerimientos los tenemos que modelar como promociones vigentes y así el sistema sabrá o generamos una volatilidad en los requerimientos para que sean flexibles y cuando el cliente quiera generar una promoción, este la carga y el sistema la valida dejándola compatible con otra promoción vigente.

Planificación de la gestión de requerimientos:

_ La planificación es la etapa inicial en el proceso de gestión de requerimientos.



_ Primero tenemos que comprender el problema inicial, después tenemos la base de los requerimientos iniciales, se produce el cambio en la comprensión del problema y después se plantean los requerimientos cambiados. Este proceso se va dando a lo largo del tiempo.

Gestión de requerimientos:

- Identificación de requerimientos: identificar cada requerimiento de forma única para permitir su rastreo.
- Un proceso de gestión del cambio: al contar con un proceso formal todos los cambios propuestos son tratados de forma consciente y controlada. Muchas veces el cliente pide cambios todo el tiempo y no se compromete con lo que realmente dijo, por ende, hay que formalizar esto para que sepa que no todos los cambios que se plantean se pueden llevar a cabo o genera un retrabajo. El cambio hay que tratarlo una vez implementado el sistema, porque en la etapa de desarrollo perdemos noción de la relación de un requerimiento con el resto del sistema, y al agregar tal cosa empiezan a surgir errores.

- Para evitar el problema anterior se definen políticas de rastreo, y para esto es importante que tengamos una matriz de trazabilidad que nos permita ver las:
 - Relaciones entre requerimientos, esto nos sirve para entender el impacto de un cambio.
 - Relaciones entre los requerimientos y el diseño del sistema que se debe registrar (elementos y diagramas que se desprenden en todas las etapas del ciclo de vida del desarrollo).
 - Y de qué manera estos registros se deben mantener.

Existen muchas relaciones entre requerimientos y entre estos y el diseño del sistema. Existen vínculos entre los requerimientos y las razones fundamentales por las que estos se propusieron. Cuando se proponen cambios, se debe rastrear el impacto de estos cambios en los otros requerimientos y en el diseño del sistema. Existen tres tipos de información de rastreo que pueden ser mantenidas:

- Información de rastreo de la fuente: vincula los requerimientos con los stakeholders que propusieron los requerimientos y la razón de éstos. Cuando se propone un cambio, esta información se utiliza para encontrar y consultar a los stakeholders sobre el cambio.
- Información de rastreo de los requerimientos: vincula los requerimientos dependientes en el documento de requerimientos. Esta información se utiliza para evaluar cómo es probable que muchos requerimientos se vean afectados por un cambio propuesto y la magnitud de los cambios consecuentes en los requerimientos.
- Información de rastreo del diseño: vincula los requerimientos a los módulos del diseño en los cuales son implementados. Esta información se utiliza para evaluar el impacto de los cambios de los requerimientos propuestos en el diseño e implementación del sistema.

Matriz de rastreo (trazabilidad): para lo mencionado anteriormente se utiliza la matriz de rastreo. A menudo, la información de rastreo se representa utilizando matrices de rastreo, las cuales relacionan los requerimientos con los stakeholders, con los módulos del diseño los requerimientos entre ellos. En una matriz de rastreo de requerimientos, cada requerimiento se representa en una fila y en una columna de la matriz. Cuando existen dependencias entre diferentes requerimientos, éstas se registran en la celda en la intersección fila/columna.

_ El siguiente ejemplo es de requerimientos con requerimientos. Esta matriz nos ayuda a analizar el impacto de los cambios, por ejemplo, el requerimiento 2.1 se relaciona con el 1.3, que estén relacionados no quiere decir que su relación sea muy estrecha o que se modifique uno si o si por el otro, pero lo tenemos que analizar y no lo podemos dejar de lado. Una “D” en la intersección fila/columna ilustra que el requerimiento en la fila depende del requerimiento señalado en la columna; una “R” significa que existe alguna

otra relación más débil entre los requerimientos. En el caso del requerimiento 2.1 si queremos modificarlo tenemos que analizar si o si el 1.3, el 2.2 y el 3.2 porque hay una relación.

Req.	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2
1.1		D	R					
1.2			D		R		D	
1.3	R			R				
2.1			R		D			D
2.2							D	
2.3		R		D				
3.1							R	
3.2						R		

_ Ventajas y desventajas:

- Las matrices de rastreo pueden utilizarse cuando se tiene que gestionar un número pequeño de requerimientos, pero son difíciles de manejar y caras de mantener para sistemas grandes con muchos requerimientos.
- Estas herramientas son super útiles para incorporarlas en todo el desarrollo o en distintas etapas del mismo, sobre todo con los requerimientos que son la parte más sensible, pero siempre con criterio. Entonces la idea es definir un módulo que puede ser critico o es un módulo que puede llegar a tener cambios y ahí se le aplica una matriz para visualizar los cambios.
- Para estos sistemas se debería captar la información de rastreo en una base de datos de requerimientos en la que cada requerimiento esté explícitamente vinculado a los requerimientos relacionados. De esta forma, se puede evaluar el impacto de los cambios utilizando las facilidades de exploración de la base de datos. Se pueden generar automáticamente matrices de rastreo de la base de datos.

Herramientas CASE: existen herramientas CASE (Computer Aided Software Engineering o Ingeniería de Software Asistida por Computadora) para:

- Almacenar requerimientos: los requerimientos deben mantenerse en un almacén de datos seguro y administrado que sea accesible a todos los que estén implicados en el proceso de ingeniería de requerimientos. Los requerimientos son información sensible por lo que no cualquiera debería acceder a la base de datos de los requerimientos.

- Gestionar el cambio: este proceso se simplifica si está disponible una herramienta de ayuda.
- Gestionar el rastreo: como se indicó anteriormente, las herramientas de ayuda para el rastreo permiten que se descubran requerimientos relacionados. Algunas herramientas utilizan técnicas de procesamiento del lenguaje natural para ayudarle a descubrir posibles relaciones entre los requerimientos.

_ El proceso de gestión de requerimientos puede llevarse a cabo utilizando los recursos disponibles en los procesadores de texto, hojas de cálculo y bases de datos. Sin embargo, para sistemas grandes, se requieren herramientas de ayuda más especializadas, como por ejemplo Requisite Pro.

Gestión del cambio

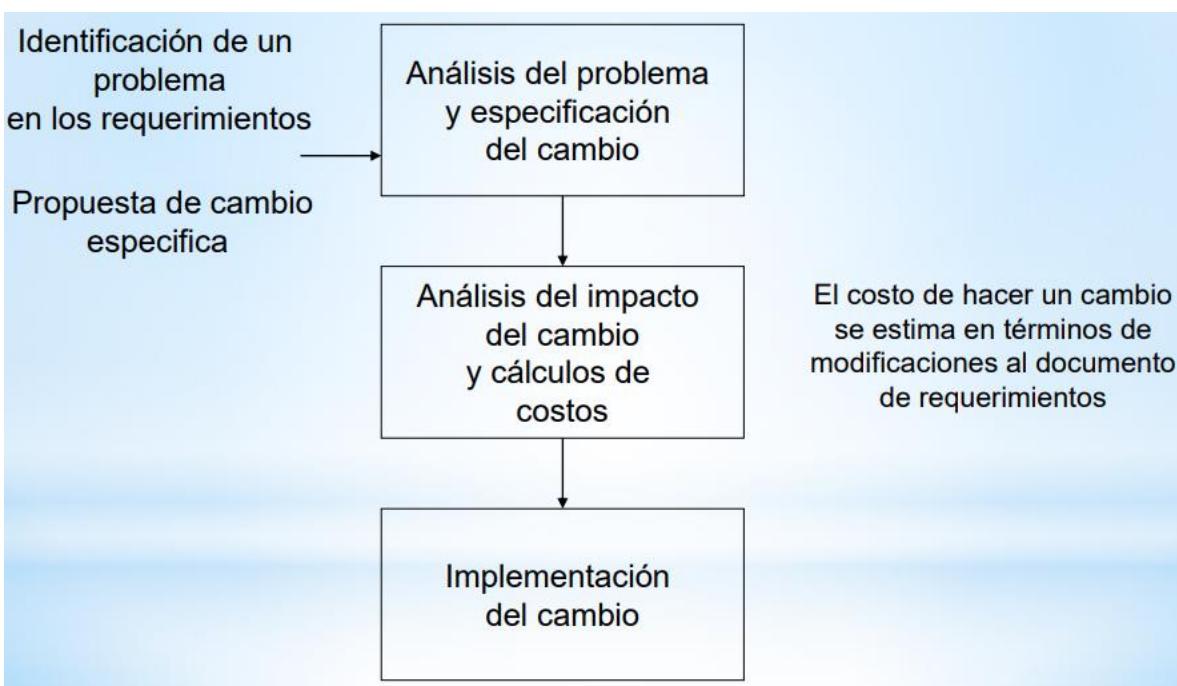
_ Se debe aplicar a todos los cambios propuestos en los requerimientos. Todos los requerimientos tienen que estar en dicha base de datos, y a los que se les requiera cambio, deberíamos identificarlos en la matriz. La ventaja de utilizar un proceso formal para gestionar el cambio es que:

- Todos los cambios propuestos son tratados de forma consistente.
- Los cambios en el documento de requerimientos se hacen de forma controlada.

_ Existen tres etapas principales en un proceso de gestión de cambio:

1. Análisis del problema y especificación del cambio: el proceso empieza con la identificación de un problema en los requerimientos o algunas veces con una propuesta de cambio específica. Durante esta etapa, el problema o la propuesta de cambio se analiza para verificar que ésta es válida. Los resultados del análisis se pasan al solicitante del cambio, y algunas veces se hace una propuesta de cambio de requerimientos más específica.
2. Análisis del cambio y cálculo de costes: el efecto de un cambio propuesto se valora utilizando la información de rastreo y el conocimiento general de los requerimientos del sistema. El coste de hacer un cambio se estima en términos de modificaciones al documento de requerimientos, y si es apropiado al diseño e implementación del sistema. Una vez que este análisis se completa, se toma una decisión sobre si se continúa con el cambio de requerimientos.
3. Implementación del cambio: se modifica el documento de requerimientos y, en su caso, el diseño e implementación del sistema. Debe organizar el documento de requerimientos de modo que pueda hacer cambios en él sin tener que hacer grandes reorganizaciones o redactar nuevamente gran cantidad del mismo. Como sucede con los programas, los cambios en los documentos se llevan a cabo minimizando las referencias externas y haciendo las secciones del documento tan

modulares como sea posible. De esta manera, se pueden cambiar y reemplazar secciones individuales sin afectar a otras partes del documento.



_ Los procesos de desarrollo iterativo, como la programación extrema, se han diseñado para hacer frente a los requerimientos que cambian durante el proceso de desarrollo. En estos procesos, cuando un usuario propone un cambio en los requerimientos, no se hace a través de un proceso formal de gestión del cambio. Más bien, el usuario tiene que establecer la prioridad del cambio y, si es de alta prioridad, decidir qué característica del sistema que fue planificada para la siguiente iteración debería abandonarse.

Control de versiones: el control de versiones debe comenzar cuando se tiene un borrador de los requerimientos de esta forma se puede mantener la historia de los cambios. Se aplica a un requerimiento y a un conjunto de requerimientos representado en la forma de documentos. Toda versión de los requerimientos debe ser únicamente identificada y todo miembro del equipo debe tener acceso a la versión actual de los requerimientos.

- Los cambios se deben documentar claramente y se deben comunicar a todos los afectados.
- Para evitar errores en la comunicación solo se permite que ciertos miembros del equipo actualicen los requerimientos.
- Para cada versión de un documento de requerimientos se debería incluir en una historia de revisión para identificar los cambios realizados. Se registra la fecha del cambio, los individuos que lo realizaron y la razón del cambio.

Diseño de software

Principios que guían el diseño

Principios fundamentales

Principios que guían el proceso:

- Ser ágil.
- En cada etapa centrarse en la calidad.
- Estar listo para adaptar.
- Formar un equipo eficaz.
- Establecer mecanismos para la comunicación y coordinación.
- Administrar el cambio.
- Evaluar el riesgo.

Principios que guían la práctica:

- Divide y vencerás.
- Entender el uso de la abstracción.
- Buscar la coherencia.
- Centrarse en la transferencia de información.
- Construir software que tenga modularidad eficaz.
- Buscar patrones.

Principios que guían toda actividad estructural

Principios de comunicación:

- Escuchar.
- Antes de comunicarse, prepararse.
- Alguien debe facilitar la actividad.
- Es mejor la comunicación cara a cara.
- Tomar notas y documentar las decisiones.
- Perseguir la colaboración.

Principios de planeación:

- Entender el alcance del proyecto.
- Involucrar en la actividad de planeación a los participantes del software.
- Reconocer que la planeación es iterativa.
- Estimar con base en lo que se sabe.
- Al definir el plan, tomar en cuenta los riesgos.

- Ser realista.

Principios de modelado:

- Principios genéricos:
 - El equipo de software tiene como objetivo principal elaborar software y no crear modelos.
 - Viajar ligero, no crear más modelos de los necesarios.
 - Tratar de producir el modelo más sencillo que describa al problema o al software.
 - Construir modelos susceptibles al cambio.
- Principios del modelado de requerimientos:
 - Debe representarse y entenderse el dominio de información de un problema.
 - Deben definirse las funciones que realizará el software.
 - Debe representarse el comportamiento del software (como consecuencias de eventos externos).
- Principios de modelado de diseño:
 - El diseño debe poder rastrearse hasta el modelo de requerimientos.
 - Siempre tomar en cuenta la arquitectura del sistema que se va a construir.
 - El diseño debe desarrollarse en forma iterativa. El diseñador debe buscar más sencillez en cada iteración.

Principios de la construcción:

- Principios de codificación:
 - Principios de preparación:
 - i. Entender el problema que se trata de resolver.
 - ii. Comprender los principios y conceptos básicos del diseño.
 - iii. Seleccionar un ambiente de programación que disponga de herramientas que hagan más fácil su trabajo.
 - Principios de programación:
 - i. Restringir sus algoritmos por medio del uso de programación estructurada.
 - ii. Seleccionar estructuras de datos que satisfagan las necesidades del diseño.
 - iii. Entender la arquitectura del software y crear interfaces que son congruentes con ella.
 - Principios de validación:
 - i. Realizar el recorrido del código cuando sea apropiado.
 - ii. Llevar a cabo pruebas unitarias y corregir los errores que se detecten.
 - iii. Rediseñar el código.

- Principios de la prueba:
 - La prueba es el proceso que ejecuta un programa con objeto de encontrar un error.
 - Un buen caso de prueba es el que tiene alta probabilidad de encontrar un error que no se ha detectado hasta el momento.
 - Una prueba exitosa es la que descubre un error no detectado hasta el momento.

Principios de despliegue:

- Deben manejarse las expectativas de los clientes.
- Debe ensamblarse y probarse el paquete completo que se entregará.
- Se deben proporcionar a los usuarios finales materiales de aprendizaje apropiados.
- El software defectuoso debe corregirse primero y después entregarse.

Diseño

Concepto

_ El diseño de software agrupa el conjunto de principios, conceptos y prácticas que llevan al desarrollo de un sistema o producto de alta calidad. El diseño es una representación significativa de ingeniería de algo que vamos a construir.

_ Marco Vitruvio, romano crítico de arquitectura, afirmaba que los edificios bien diseñados eran aquellos que tenían resistencia, funcionalidad y belleza. Lo mismo se aplica al buen software:

- Resistencia: un programa no debe tener ningún error que impida su funcionamiento.
- Funcionalidad: un programa debe ser apropiado para los fines que persigue.
- Belleza: la experiencia de usar el programa debe ser placentera.

Roger Pressman: el diseño del software comienza cuando termina la primera iteración de la ingeniería de requerimientos. El objetivo del diseño del software es aplicar un conjunto de principios, conceptos y prácticas que llevan al desarrollo de un sistema o producto de alta calidad. La meta del diseño es crear un modelo de software que implantará correctamente todos los requerimientos del usuario y causará satisfacción a quienes lo utilicen. Los diseñadores del software deben elegir entre muchas alternativas de diseño y llegar a la solución que mejor se adapte a las necesidades de los participantes en el proyecto.

Diseñar

- _ El objetivo del diseño es producir un modelo o representación que tenga resistencia, funcionalidad y belleza. Para lograrlo, debe practicarse la diversificación y luego la convergencia.
- _ El diseño arquitectónico es la primera etapa en el proceso de diseño y representa un enlace crítico entre los procesos de ingeniería de diseño y de requerimientos.
- _ Al diseño lo realizan los ingenieros del software, ingenieros en sistemas, y todos aquellos relacionados. Y es importante ya que necesitamos un plano. El diseño comienza con el modelo de requerimientos. Se trabaja para transformar este modelo y obtener cuatro niveles de detalle de diseño:
 - Estructura de datos
 - Arquitectura del sistema
 - Representación del interfaz
 - Detalles a nivel de componentes
- _ Se produce una especificación del diseño que se compone de los modelos del diseño que describen, los datos, la arquitectura, interfaces y componentes. Tenemos que considerar el tipo de sistemas que estamos desarrollando para saber si el enfoque estará puesto en el usuario, la accesibilidad, la información que circula, la performance, la seguridad, etc. Ahora bien, en cada etapa, para saber si lo hecho es correcto, se revisan los productos del diseño del software, en cuanto a calidad, corrección, finalización y consistencia.
- _ El diseño del software, al igual que los enfoques de diseño de ingeniería en otras disciplinas, va cambiando continuamente a medida que se desarrollan métodos nuevos, análisis mejores y se amplía el conocimiento.

El diseño del software y la ingeniería del software

- _ El diseño del software se encuentra en el núcleo técnico de la IS y se aplica independientemente del modelo de diseño de software que se utilice. Una vez que se analizan y especifican los requisitos del software, el diseño del software es la primera de las tres actividades técnicas que se requieren para construir y verificar el software:

- Diseño
- Generación de código
- Pruebas

_ Ventajas de diseñar según Bass:

- Comunicación stakeholders.
- Análisis del sistema.
- Reutilización a gran escala.

_ Ventajas de diseñar según Hofmeister:

- Herramienta para negociar los requerimientos del sistema.
- Forma de estructurar las discusiones con los clientes, desarrolladores y gestores.
- Herramienta esencial para la gestión de la complejidad.

_ La arquitectura del sistema afecta al rendimiento, solidez, grado de distribución y mantenibilidad de un sistema.

1. Rendimiento. → Subsistemas
2. Protección. → Capas
3. Seguridad. → Único Subsistema
4. Disponibilidad. → Componentes Redundantes
5. Mantenibilidad. → Componentes Independientes

Lineamientos para el diseño:

- Debe tener una arquitectura que:
 - 1) Se haya creado con el empleo de estilos o patrones arquitectónicos reconocibles.
 - 2) Esté compuesta de componentes con buenas características de diseño.
 - 3) Se implementen en forma evolutiva, de modo que faciliten la implementación y las pruebas.
 - Debe ser modular, es decir, el software debe estar dividido de manera lógica en elementos o subsistemas.
 - Debe contener distintas representaciones de datos, arquitectura, interfaces y componentes.
 - Debe conducir a estructuras de datos apropiadas para las clases que se van a implementar y que surjan de patrones reconocibles de datos.
 - Debe llevar a componentes que tengan características funcionales independientes.
- Lineamientos para el diseño.
- Debe conducir a interfaces que reduzcan la complejidad de las conexiones entre los componentes y el ambiente externo.
 - Debe obtenerse con el empleo de un método repetible motivado por la información obtenida durante el análisis de los requerimientos del software.
 - Debe representarse con una notación que comunique con eficacia su significado.

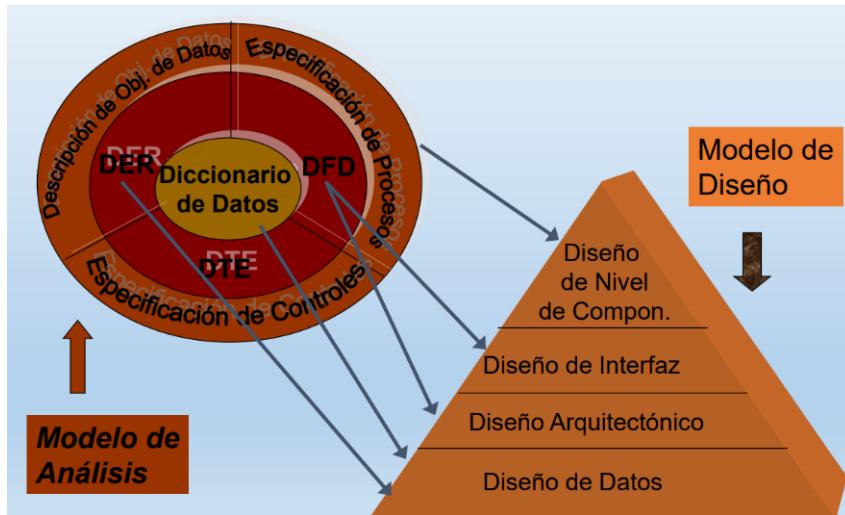
_ Estos lineamientos se consiguen con la aplicación de los principios de diseño fundamentales, una metodología sistemática y con revisión.

Elementos del modelo de análisis

_ El modelo de análisis debe lograr tres objetivos:

1. Describir lo que quiere el cliente.
2. Establecer una base para el diseño de software.
3. Definir un conjunto de requerimientos que pueda validar una vez que se construye el software.

_ Cada parte del modelo de análisis proporciona la información necesaria para crear los cuatro modelos de diseño. El modelo de análisis extraído de un diseño estructurado es el siguiente:



- Diseño de datos (estructuras de datos): transforma el modelo del dominio de información que se crea durante el análisis en las estructuras de datos que se necesitarán para implementar el software. Los objetos de datos y las relaciones definidas en el DER (diagrama relación entidad) y el contenido de datos detallado que se representa en el diccionario de datos proporcionan la base de la actividad del diseño de datos.
 - Modelo de análisis: descripción de objetos de datos y diccionario de datos.
- Diseño arquitectónico: define la relación entre los elementos estructurales principales del software, los patrones de diseño que se pueden utilizar para lograr los requisitos que se han definido para el sistema, y las restricciones que afectan a la manera en que se pueden aplicar los patrones de diseño arquitectónicos.
 - Modelo de análisis: diagrama de flujo de datos (DFD).
- Diseño de la interfaz: describe la manera de comunicarse el software dentro de sí mismo, con sistemas que interoperan dentro de él y con las personas que lo utilizan.

- Modelo de análisis: diagrama de flujo de datos (DFD).
- Diseño a nivel de componentes: transforma los elementos estructurales de la arquitectura del software en una descripción procedural de los componentes del software.

_ La importancia del diseño del software se puede describir con una sola palabra "calidad". Entonces, el diseño es el lugar en donde se fomentará la calidad en la ingeniería del software. El diseño proporciona las representaciones del software que se pueden evaluar en cuanto a calidad. Sin un diseño, corremos el riesgo de construir un sistema inestable.

Proceso de diseño

_ El diseño del software es un proceso iterativo mediante el cual los requisitos se traducen en un "plano". Se representa desde un nivel más global (abstracto) y va hacia lo más particular (comportamiento, funcionales y de datos) a medida que ocurren iteraciones el nivel de abstracción va disminuyendo.

Diseño y calidad

_ La calidad de la evolución del diseño se evalúa con una serie de revisiones técnicas formales:

- El diseño deberá implementar todos los requerimientos explícitos del modelo de análisis y deberán ajustarse a todos los requerimientos implícitos que desea el cliente.
- El diseño proporcionará una imagen completa del software (teniendo en cuenta funciones, comportamiento, y datos) desde una perspectiva de implementación.
- El diseño es una guía para los que generan código, testing y datos desde una perspectiva de implementación.

Criterios de calidad del diseño:

_ Con el fin de evaluar la calidad de una representación de diseño, deberán establecerse los criterios técnicos para un buen diseño. Un diseño deberá presentar una estructura arquitectónica que:

1. Se haya creado mediante patrones de diseño reconocibles.
2. Que esté formada por componentes que exhiban características de buen diseño.
3. Un diseño deberá contener distintas representaciones de datos, arquitectura, interfaces y componentes (módulos).
4. Deberá conducir interfaces fáciles de conectarse entre módulos y el entorno.
5. Un diseño deberá conducir a estructuras de datos adecuadas para los objetos que se van a implementar.

6. Deberá derivarse mediante un método repetitivo y controlado por la información obtenida en el análisis.

Principios de diseño:

_ El diseño de un software es tanto un proceso, es decir, una Secuencia de pasos que hacen posible que el diseñador describa todos los aspectos del software que se va construir, como así también un modelo (planos).

_ Los principios básicos de diseño hacen posible que el ingeniero del software navegue por el proceso de diseño, estos son:

- En el proceso de diseño no deberá utilizarse “orejeras” (enfoques alternativos).
- El diseño deberá poderse rastrear hasta el modelo de análisis.
- El diseño no deberá inventar nada que ya esté inventado.
- El diseño deberá estructurarse para admitir cambios.
- El diseño no es escribir código y escribir código no es diseñar.
- El diseño deberá evaluarse en función de la calidad mientras se va creando, no después de terminarlo.
- El diseño deberá revisarse para minimizar los errores conceptuales (semánticos).

_ Si aplicamos todo lo descripto anteriormente es fácil ver la calidad del software , que muestra los factores de calidad tanto internos como externos.

- Factores de calidad externos: son esas propiedades del software que pueden ser observadas fácilmente por los usuarios (por ejemplo, velocidad, fiabilidad, grado de corrección, usabilidad).
- Factores de calidad internos: tienen importancia para los ingenieros del software. Desde una perspectiva técnica conducen a un diseño de calidad alta. Para lograr los factores de calidad internos, el diseñador deberá comprender los conceptos de diseño básicos.



Arquitectura de software

Introducción

Arquitectura: en el nivel más sencillo, se considera la forma general de la estructura física. Podemos decir también que es la manera en la que los distintos componentes del edificio se integran para formar un todo cohesivo, es la sensación estética de la estructura, es arte.

Arquitectura de software

Concepto

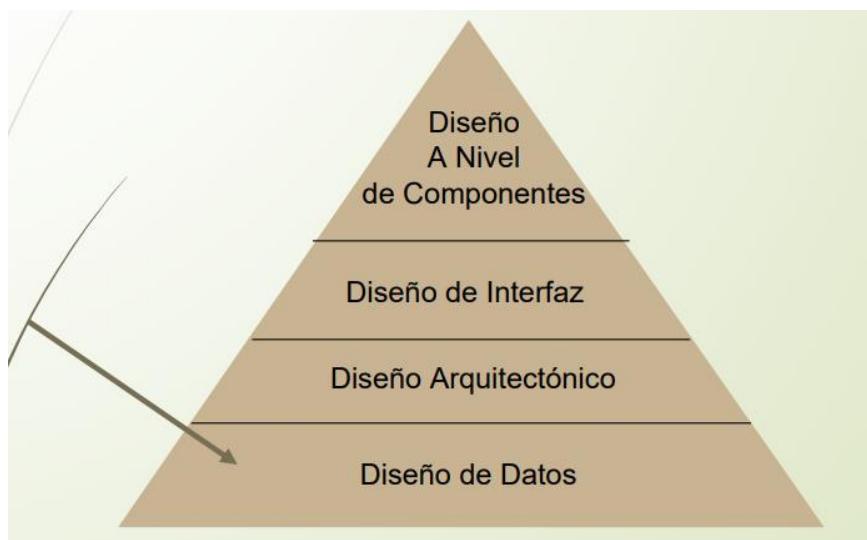
_ La arquitectura no es el software operativo. Es una representación que permite analizar la efectividad del diseño para cumplir los requerimientos establecidos, considerar alternativas arquitectónicas en una etapa en la que hacer cambios al diseño todavía es relativamente fácil y reducir los riesgos asociados con la construcción del software.

_ La arquitectura de software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución.

_ La arquitectura de un sistema define la división y estructura de un sistema en subsistemas y establece un marco de control, comunicación y cooperación entre los distintos subsistemas. Además, podemos decir que la arquitectura es:

- Un conjunto de elementos/componentes que conforman un sistema con sus responsabilidades.
- Un conjunto de conectores entre componentes (comunicación, coordinación, control, cooperación).
- Restricciones que definen cómo se integran los componentes para formar el sistema.

_ El diseño comienza con la consideración de la arquitectura.



Arquitectura multicapa

_ Una arquitectura multicapa es una clara organización en base a ciertos niveles de abstracción. Un sistema por capas es un conjunto ordenado de subsistemas, cada uno de los cuales está construido en términos de los que tiene por debajo y proporciona la base de la implementación de los que están por encima de él.

Documentar la arquitectura

_ Significa documentar las vistas relevantes bajo las cuales se puede observar un sistema. Es importante documentarla para que posteriormente la revisen los participantes que deseen entender la descripción de la arquitectura propuesta.

- La arquitectura alimenta la actividad del diseño.
- Muchas decisiones de diseño quedan sin consolidar por la arquitectura y se dejan a los diseñadores detallarlos.
- La arquitectura define restricciones en las actividades posteriores.
- La arquitectura sirve como medio de educación.
- La arquitectura sirve como un medio principal de comunicación entre los stakeholders.

_ Errores más comunes al documentar:

- Documento demasiado elaborado
- Incógnitas
- Se hace difícil de mantener

Estilos arquitectónicos

_ Un estilo provee un mecanismo rápido de visualización del estilo elegido. Para esto se usa una plantilla de construcción que orienta al constructor:

- Conjunto de componentes (como una base de datos o módulos de cómputo) que realizan una función requerida por el sistema.
- Un conjunto de conectores que permiten la “comunicación, coordinación y cooperación” entre los componentes.
- Restricciones que definen cómo se integran los componentes para formar el sistema.
- Modelos semánticos que permiten que un diseñador entienda las propiedades generales del sistema al analizar las propiedades conocidas de sus partes constituyentes.

Vistas arquitectónicas

_ Una vista es una representación de un conjunto de elementos del sistema y las relaciones asociadas con ellos.

_ Documentar una arquitectura significa documentar las vistas relevantes bajo las cuales se puede observar un sistema. Se tienen uno o más documentos de vista y documentación que explican cómo se relacionan las vistas entre sí. Las diferentes vistas exponen diferentes atributos de calidad en diferentes grados.

Documentos para una vista: la documentación para una vista incluye una representación gráfica de los elementos principales y sus relaciones, un catálogo de elementos que define sus propiedades, una especificación de las interfaces y el comportamiento de cada elemento, así como justificación e información de diseño relacionada.

_ Por otro lado, la documentación que se aplica a todas las vistas contiene información que describe cómo se relacionan entre sí y con el sistema en su conjunto. También incluye restricciones y justificación de la arquitectura utilizada, así como información de gestión necesaria para mantener todo el paquete arquitectónico. Esta documentación abarca aspectos más generales y tiene como objetivo proporcionar una visión completa y coherente del sistema arquitectónico en su conjunto.

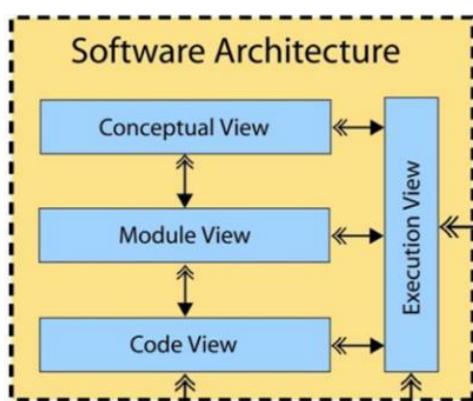
Estructuras o vistas de la arquitectura:

- Funcional: los componentes representan entidades de función o procesamiento. Los conectores representan interfaces que proveen la capacidad de “usar” o “pasar datos a” un componente. Las propiedades describen la naturaleza de los componentes y la organización de las interfaces.
- De implementación: los componentes son paquetes, clases, objetos, procedimientos, funciones, métodos, etc, que son vehículos para empacar funciones en varios niveles de abstracción estructuras o vistas de la arquitectura.
- De concurrencia: los componentes representan “unidades de concurrencia” que están organizadas como tareas o trayectorias paralelas. Las relaciones incluyen sincronizarse con, tiene mayor prioridad que, envía datos a, no corre sin y no corre con. Las propiedades relevantes para esta estructura incluyen prioridad, anticipación y tiempo de ejecución.
- Física: esta estructura es similar al modelo de despliegue desarrollado como parte del diseño. Los componentes son el hardware físico en el que reside el software. Los conectores son las interfaces entre los componentes del hardware y las propiedades incluyen la capacidad, ancho de banda y rendimiento, entre otros atributos.
- De desarrollo: esta estructura define los componentes, productos del trabajo y otras fuentes de información que se requieren a medida que avanza la ingeniería

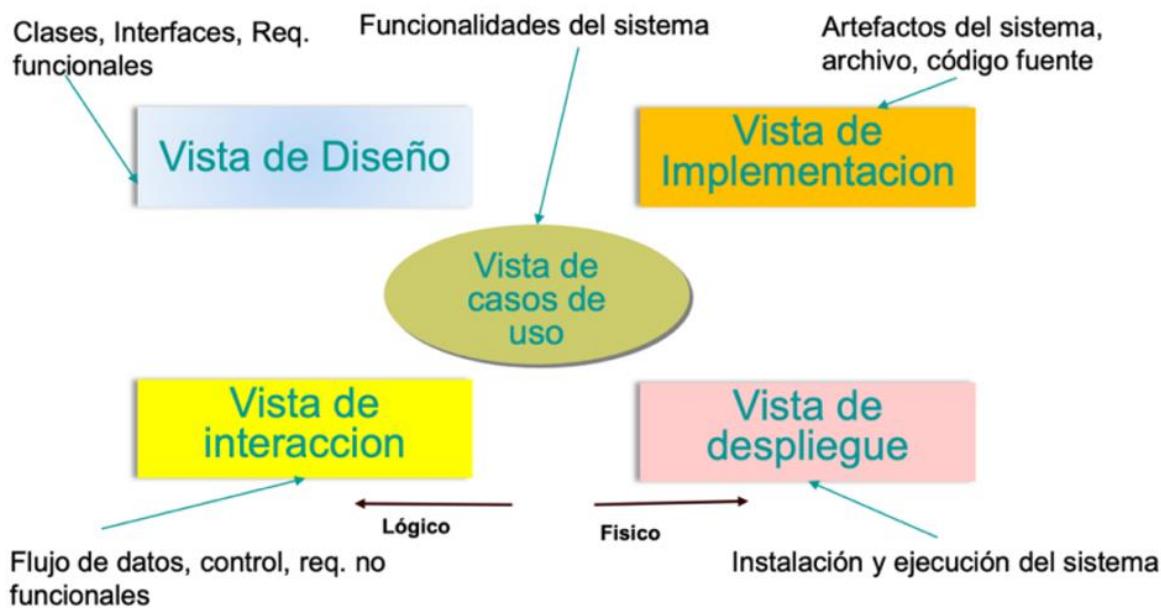
de software. Los conectores representan las relaciones entre los productos del trabajo; las propiedades identifican las características de cada aspecto.

Vistas arquitectónicas de Siemens:

- Vista conceptual: describe el sistema en términos de sus principales elementos de diseño y las relaciones entre ellos.
 - Vista de interconexión del módulo: abarca dos vistas ortogonales: descomposición funcional y capas.
 - Vista de ejecución: describe la vista dinámica del sistema.
 - Vista de código: describe cómo se organiza el código fuente.



Vistas según el modelo 4+1 de Kruchten



_ Tenemos varios stakeholders, varias vistas, donde se le muestra a cada uno distintas cosas. No todos los sistemas requieren todas las vistas, ya que depende del contexto. Además de las vistas comunes, como despliegue, procesos e implementación, también se consideran vistas como datos y seguridad.

Diseño arquitectónico

_ En la documentación de una arquitectura de software, es importante considerar lo siguiente:

1. Representación del sistema en contexto: se debe representar el sistema objetivo en relación con su entorno, identificando sistemas superiores, sistemas subordinados, sistemas entre iguales y actores que interactúan con el sistema.
2. Identificación de arquetipos de la arquitectura: los arquetipos son clases o patrones fundamentales para el diseño de la arquitectura. Representan abstracciones críticas y pueden ser componentes genéricos o módulos especializados con funciones específicas.
3. Refinamiento hacia los componentes: los arquetipos se refinan en componentes más específicos, definiendo interfaces e interacciones entre ellos. Esto permite un diseño detallado y una implementación coherente de la arquitectura.

_ Estos puntos son esenciales para documentar y diseñar una arquitectura de software, proporcionando una comprensión del contexto, identificación de partes clave y una estructura clara para el sistema.

Evaluación de diseños arquitectónicos:

1. Obtención de los requerimientos y restricciones, y descripción del ambiente.
2. Escenarios de investigación. Se desarrolla un conjunto de casos de uso para representar el sistema.
3. Descripción de los estilos o patrones de arquitectura elegidos para abordar los escenarios y requerimientos.
4. Evaluación de los atributos de calidad, considerando cada atributo por separado.
5. Identificación de la sensibilidad de los atributos de calidad de varios atributos arquitectónicos para un estilo de arquitectura específico.
6. Crítica de las arquitecturas candidatas con el uso del análisis de sensibilidad.
 - Calidad: incluye confiabilidad, desempeño, seguridad, facilidad de mantenimiento, flexibilidad, facilidad de hacer pruebas, portabilidad, reutilización e interacción.
 - Sensibilidad: es igual a atributos que se ven afectados en forma significativa por la variación de la arquitectura.
 - Lenguaje de descripción arquitectónica (LDA): se usa porque la notación UML se queda corta, y esta provee la semántica y sintaxis para describir una arquitectura de software.

Estilos arquitectónicos

Arquitectura en capas:

- Constituye uno de los estilos que aparecen con mayor frecuencia y muy utilizados.
- Organización jerárquica de capas.
- Cada capa opera sólo con capas adyacentes.
- Patrones que varían del estilo: Façade, Adapter, Bridge y Strategy.

_ Una capa es una entidad compleja, compuestas de varios paquetes o subsistemas, en donde:

- Cada capa siempre debe hacer algo.
- El cruce superfluo de muchos niveles involucra eventuales degradaciones de performance.
- Muchas veces se sacrifica la pureza de la arquitectura en capas precisamente para mejorarla.
- Número mínimo de capas: 2.
- Sub-estilo cliente-servidor.

_ Algunos casos representativos de este estilo son:

- Protocolos de comunicación en capas: modelo OSI.
- Arquitecturas de bases de datos y sistemas operativos.
- Especificaciones relacionadas con la tecnología de XML.

_ Ventajas:

- Soporta un diseño basado en niveles de abstracción crecientes.
- Admite optimizaciones y refinamientos.
- Proporciona amplia reutilización.

_ Desventajas:

- Muchos problemas no admiten un buen mapeo en una estructura jerárquica.
- Es difícil encontrar el nivel de abstracción correcto.
- Los cambios en las capas de bajo nivel tienden a filtrarse hacia las de alto nivel.

_ Otros estilos arquitectónicos son:

- Model View Controller
- Orientada a objetos
- Cliente servidor

Patrones de diseño

Patrones

_ Un patrón es un par problema/ solución con un nombre que se puede aplicar en nuevos contextos, con consejos acerca de cómo aplicarlo en nuevas situaciones y discusiones sobre sus compromisos. Estos patrones surgen como una acumulación de un repertorio de experiencias a cerca de soluciones que fueron exitosas en la solución de diversos problemas. Conforman una guía de principios y estilos de diseño cuya aplicación lleva a arquitecturas de diseño sólidas y reutilizables. Los nombres de los patrones permiten identificar claramente el concepto y facilita la comunicación entre miembros del equipo de desarrollo.

Patrones GRASP

_ GRASP es un acrónimo de General Responsibility Assignment Software Patterns (patrones generales de software para asignar responsabilidades). Estos describen los principios fundamentales del diseño de objetos y la asignación de responsabilidades expresados como patrones.

- Responsabilidades: contrato u obligación asociada a un objeto en cuanto a su comportamiento. Básicamente hay dos tipos de responsabilidades, conocer y hacer. Las responsabilidades no son lo mismo que los métodos, pero los métodos se implementan para llevar a cabo responsabilidades. Las responsabilidades se implementan utilizando métodos que actúan solos o colaboran con otros métodos u objetos.

_ Los patrones básicos GRASP son:

Patrón experto en información:

- Problema: ¿Quién asigna responsabilidades a los objetos?
- Solución: asignar una responsabilidad al experto en información (la clase que tiene la información necesaria para realizar la responsabilidad). Siguiendo la solución del patrón deberíamos buscar clases y objetos que tienen la información necesaria para determinar el total. Este patrón es un principio de guía básico en el diseño de objetos. Expresa la “intuición” de lo que creemos que los objetos del mundo real hacen y por eso es muy fácil de aplicar.
- Contraindicaciones: en algunas circunstancias la solución de “experto” no es deseable. Por ejemplo, para almacenar en una BD la información.
- Beneficios: se mantiene el encapsulamiento de la información (los objetos utilizan su propia información para trabajar) Se distribuye el comportamiento entre clases

obteniendo clases más pequeñas, fáciles de entender y manejar. Se logra una alta cohesión.

Patrón creador:

- Problema: ¿Quién debería ser el responsable de la creación de una nueva instancia de alguna clase?
- Solución: asignar a la clase B la responsabilidad de crear una instancia de A si se cumple uno o más de los siguientes casos, cuando B agrega objetos de A, B contiene objetos de A, B registra instancias de objetos de A, B es un creador de los objetos A, etc. La intención del patrón Creador es encontrar el creador que necesita a futuro conectarse al objeto creado en alguna situación. El Creador sugiere que las clases contenedoras son buenas para tener la responsabilidad de la creación de sus clases contenidas.
- Contraindicaciones: cuando la creación requiera alta complejidad (reciclo para mayor rendimiento) es complicado la creación de instancias.
- Beneficios: se soporta el bajo acoplamiento.

Patrón bajo acoplamiento:

- Problema: ¿Cómo soportar bajas dependencias, bajo impacto en el cambio e incremento de la reutilización?
- Solución: asignar una responsabilidad de manera que el acoplamiento permanezca bajo. El acoplamiento es una medida de la fuerza con que un elemento está conectado a, tienen conocimiento de, otros elementos. Un elemento con bajo acoplamiento no depende de demasiados otros elementos. La no existencia de acoplamiento (clase independientes) no es deseable ya que atenta contra el objetivo de los diseños orientados a objetos donde el sistema es un conjunto interconectado de objetos trabajando entre sí.
- Contraindicaciones: la aplicación de bajo acoplamiento para “Futuras Necesidades” donde no hay motivos realistas para hacerlo puede ser contraproducente.
- Beneficios: no afectan los cambios en otros componentes, fácil de entender de manera aislada, y conveniente para reutilizar.

Patrón alta cohesión:

- Problema: ¿Cómo mantener la complejidad manejable?
- Solución: Asignar una responsabilidad de manera que la cohesión permanezca alta. La cohesión es una medida de fuerza con la que se relacionan y del grado de focalización de las responsabilidades de un objeto. Una clase con baja cohesión hace muchas cosas no relacionadas o hace demasiado trabajo. Las clases con baja cohesión representan un grano grande de abstracción (deberían haberse delegado en otros objetos).

- Cohesión y acoplamiento el yin y el yang: una mala cohesión causa normalmente un mal acoplamiento y viceversa debido a que uno influye en el otro.
- Beneficios: se incrementa la claridad y compresión del diseño. Se simplifica el mantenimiento y mejoras. Se soporta a menudo bajo acoplamiento. El grado fino de funcionalidad incrementa la reutilización.

Patrón controlador:

- Problema: ¿Quién debe manejar eventos del sistema?
- Solución: asignar la responsabilidad de recibir o manejar eventos del sistema a una clase. Un controlador es un objeto responsable del manejo de los eventos del sistema. Se pueden asignar a una o más clases controlador. Normalmente los controladores deben delegar las tareas a hacer a otros objetos, su misión solo debería ser coordinar o controlar las actividades. Esto para no sobrecargar la clase y perder cohesión. Es decir, un controlador saturado.
- Beneficios: aumenta el potencial para reutilizar y permite el intercambio de interfaces fácilmente.

_ A continuación, tenemos otros cuatro patrones GRASP (GRASP Advanced) que tienen una relación muy estrecha entre sí:

Polimorfismo: en la programación orientada a objetos el polimorfismo es permitir que varias clases se comporten de manera distinta dependiendo del tipo que sean. Siempre que se tenga que llevar a cabo una responsabilidad que dependa de un tipo, se tiene que hacer uso del polimorfismo.

- Problema: ¿Cómo manejar alternativas basadas en el tipo? ¿Cómo crear componentes de software conectables?
- Solución: cuando las alternativas o comportamientos relacionados varían con el tipo de clase, asigne la responsabilidad del comportamiento (utilizando operaciones polimórficas) a los tipos para los que varía el comportamiento. Es un principio fundamental para gestionar variaciones similares.
- Contraindicaciones: se corre el riesgo de diseñar sistemas con interfaces para “futuras necesidades” frente a variaciones desconocidas.
- Beneficios: se añaden fácilmente extensiones necesarias de nuevas variaciones y Las nuevas implementaciones no afectan a los clientes.

Fabricación pura:

- Problema: ¿Qué hacer cuando se quiere mantener los objetivos de alta cohesión y bajo acoplamiento, pero las soluciones que ofrece experto no son adecuadas?
- Solución: asigne un conjunto de responsabilidades altamente cohesivo a una clase artificial o de conveniencia, que no representa un concepto del dominio del

problema. Muchas veces la elección de experto trae aparejados problemas de poca reutilización alto acoplamiento y baja cohesión como por ejemplo el caso del almacenamiento de datos en una base de datos.

- Contraindicaciones: la inexperiencia puede llevar a abusar de la creación de clases fabricación pura convirtiendo en muchos casos funciones en objetos.
- Beneficios: soporta alta cohesión y el potencial de reutilización aumenta debido a la presencia de clases fabricación pura que tienen aplicación en otros dominios.

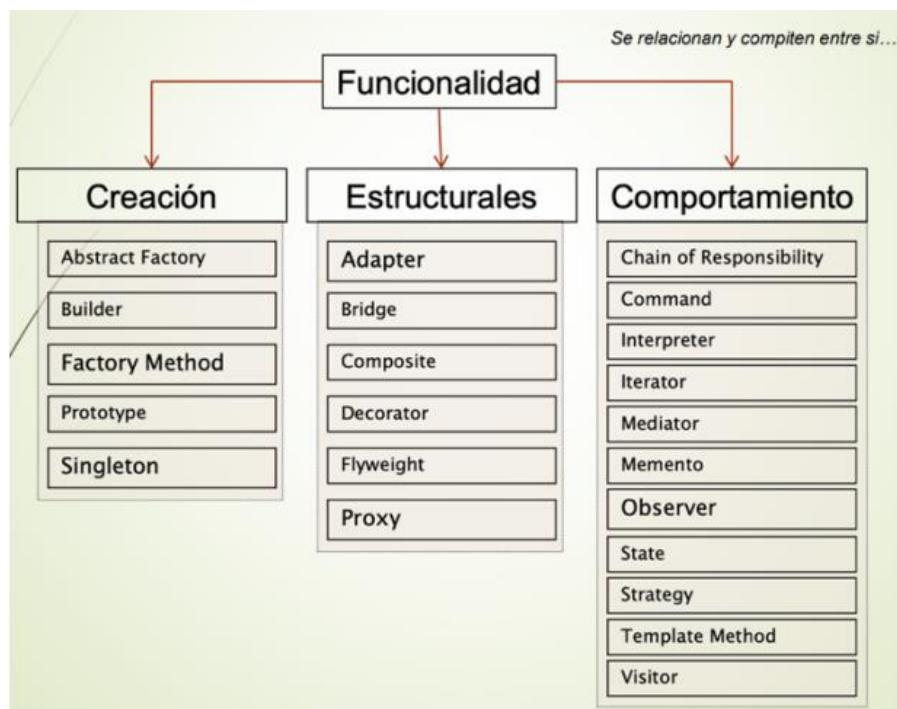
Indirección:

- Problema: ¿Dónde asignar una responsabilidad para evitar el acoplamiento directo entre dos o más cosas? ¿Cómo desacoplar objetos de manera que se soporte el bajo acoplamiento y el potencial para reutilizar permanezca alto?
- Solución: asigne la responsabilidad a un objeto intermedio que funcione como mediador entre otros componentes o servicios de manera que no se acoplen directamente.
- Beneficios: Disminuir acoplamiento entre componentes.

Variaciones protegidas:

- Problema: ¿Cómo diseñar objetos, subsistemas, y sistemas de manera que las variaciones o inestabilidades en estos elementos no tengan un impacto no deseable en otros elementos?
- Solución: asignar responsabilidades para crear una interfaz estable alrededor de puntos de variaciones previstas o de inestabilidad.
- Beneficios: se añaden fácilmente las extensiones que se necesitan para nuevas variaciones, se pueden introducir nuevas implementaciones es sin afectar a los clientes, se reduce el acoplamiento.

Patrones Gof (Gang if Four)



_ “Pedirle azúcar solo a conocidos, a los vecinos, y no irse más lejos para buscar, ya que es inestable”.

Adaptador: existen clases que realizan métodos parecidos, pero utilizan diferentes interfaces. ¿Cómo resolver interfaces incompatibles, o proporcionar una interfaz estable para componentes parecidos con diferentes interfaces?, se convierte la interfaz original de un componente en otra interfaz mediante un objeto Adaptador intermedio. El Adaptador incluirá al Adaptado, la cual podrá utilizar sus métodos originales sin redefinirlos a partir de una Interfaz incompatible.

Factory: necesito crear instancias de clases que implementan diferentes Interfaces Las instancias requieren de creaciones complejas y lógica de reutilización o de negocio para su creación. ¿Quién debe ser el responsable de la creación de los objetos cuando existen consideraciones especiales?, se crea un objeto Fabricación Pura denominado Factoría que maneje la creación.

Singleton: necesito acceder a instancias de clases de manera global, sin utilizar artimañas de programación (variables globales). Solo habrá una única instancia de la clase. Se admite exactamente una instancia de una clase (un Singleton), los objetos necesitan un único punto de acceso global