1- EL PROCESO UNIFICADO DE DESARROLLO DE SOFTWARE:

Es un proceso de desarrollo de SW, el cual es un conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema de SW.
Esta basado en componentes y utiliza UML.

- Esta dirigido por Casos de Uso (CU)
    - el usuario representa a alguien o algo que interactúa con el sistema que estamos desarrollando.
    - Un CU es un fragmento de funcionalidad del sistema que proporciona al usuario un resultado importante.
    - Representan los requisitos funcionales que constituyen el modelo de CU.
    - Guían el proceso de desarrollo.
    - Los desarrolladores revisan modelos para que sean conformes al modelo de CU.
    - Los CU se especifican se diseñan y son la fuente de los Casos de Prueba.
- Esta centrado en la arquitectura
    - La arquitectura es una vista del diseño completo con las características más importante resaltadas, dejando los detalles de lado.
    - La función corresponde a los CU y la Forma a la arquitectura.
- Es iterativo e incremental
    - Es práctico dividir el trabajo en miniproyectos, cada uno es una iteración que resulta en un incremento. Estas deben estar controladas
    - Si una iteración cumple con sus objetivos el desarrollo continúa con la siguiente, cuando no los cumple se deben revisar decisiones previas y probar un nuevo enfoque.
    - Las iteraciones deben secuenciarse en un orden lógico.
    - Beneficios:
        - Reducción del costo del riesgo a los costos de un solo incremento.
        - Reduce el riesgo de no sacar al mercado el producto en el calendario previsto.
        - Acelera el retorno del esfuerzo del desarrollo en su totalidad.
        - Reconoce las necesidades del usuario que no pueden definirse completamente al principio.

1.1- Vida del Proceso

El PU se repite a lo largo de una serie de ciclos, cada uno concluye con una versión del producto para los clientes.
Cada ciclo consta de 4 fases:

- Inicio:
    - Responde a ¿Cuáles son las principales funciones del sistema para sus usuarios más importantes? ¿Cómo podría ser la arquitectura del sistema? ¿Cúal es el plan del proyecto y cuánto costará desarrollar el producto?
    - En esta fase se identifican y priorizan los riesgos más importantes.
- Elaboración:
    - Se especifican en detalles, CU y se diseña la arquitectura del sistema.
    - Se realizan los CU más críticos que se identificaron en la fase del comienzo y el resultado es una línea base de la arquitectura.
- Construcción:
    - La línea base de la arquitectura crece hasta convertirse en el sistema completo.
- Transición:
    - Periodo de tiempo durante el cual el producto se convierte en version beta.
    - Usuarios con experiencia prueban el producto e informan defectos y deficiencias.
    - Categorías de los defectos:
        - Los que tienen suficiente impacto en la operación para justificar una versión incrementada.
        - Los que pueden corregirse en la siguiente versión normal.

El producto terminado incluye requisitos, CU, especificaciones no funcionales y Casos de Prueba, incluye el modelo de la arquitectura y el visual.
Los elementos de un modelo poseen dependencias de trazas hacia atrás y hacia adelante mediante enlaces hacia otros modelos.

1.2- Fases dentro de un Ciclo:

Cada fase termina con un hito.
Se determinan por la disponibilidad de un conjunto de artefactos.
Permiten controlar el proceso de trabajo según pasa por estos 4 puntos clave.

- Fase de inicio: responde ¿cuáles son las principales funciones del sistema para sus usuarios más importantes?¿cómo podría ser la arquitectura del sistema?¿cual es el plan del proyecto y cuanto costara desarrollar el producto?1 pregunta se responde en modelo de casos de uso. En esta fase se identifican y priorizan los riesgos más importantes.
- Fase de elaboración: se especifican en detalle casos de uso y se diseña la arquitectura del sistema. SE realizan los casos de uso más críticos que se identificaron en la fase del comienzo. REsultado es una línea base de la arquitectura.

- Fase de construcción: línea base de la arquitectura crece hasta convertirse en el sistema completo.
- Fase de transición: período de tiempo durante el cual el producto se convierte en versión beta. Usuarios con experiencia prueban el producto e informan defectos y deficiencias. Categorías de defectos: los que tienen suficiente impacto en la operación para justificar una versión incrementada y los que pueden corregirse en la siguiente versión normal.

2 - LAS 4 “P” EN EL DESARROLLO DE SOFTWARE

- Personas. Los principales autores de un proyecto software, además de los usuarios, clientes y otros interesados. Las personas son realmente seres humanos.
- Proyecto. Elemento organizativo a través del cual se gestiona el desarrollo de software.
- Producto. Artefactos que se crean durante la vida del proyecto
- Proceso. Conjunto completo de actividades necesarias para transformar los requisitos de usuario en un producto
- Herramientas. Software para automatizar las actividades definidas en el proceso

Convirtiendo recursos en trabajadores

Trabajador:

Puestos a los cuales se pueden asignar personas. Un tipo de trabajador es un papel que un individuo puede desempeñar en el desarrollo de software.
Cada trabajador es responsable de un conjunto completo de actividades.
Un trabajador también puede representar a un conjunto de personas que trabajan juntas.

Los proyectos construyen el producto

- Una secuencia de cambio: Los proyectos obtienen productos pero el camino hasta obtenerlos es una serie de cambios. El primer ciclo es un caso especial
- Una serie de iteraciones: los trabajadores llevan a cabo las actividades de la fase a través de una serie de iteraciones. Los desarrolladores progresan a través de una serie de flujos de trabajo
- Un patrón organizativo: Un proyecto implica a un equipo de personas asignadas para lograr un resultado dentro de las restricciones del negocio, es decir, tiempo, coste y calidad.

¿Qué es un sistema software?

Un sistema es todos los artefactos que se necesitan para representarlo en una forma comprensible por máquinas u hombres, para las máquinas, los trabajadores y los interesados.

Artefactos

Es un término general para cualquier tipo de información creada, producida, cambiada o utilizada por los trabajadores en el desarrollo del sistema.
Hay dos tipos de artefactos : artefactos de ingeniería y artefactos de gestión.
El desarrollo de software también requiere artefactos de gestión. Sólo duran lo que dure la vida del proyecto.

¿Qué es un modelo?

Un modelo es una abstracción del sistema , especificando el sistema modelado desde un cierto punto de vista y en un determinado nivel de abstracción
Cada modelo es una vista autocontenida del sistema
Un usuario de un modelo no necesita para interpretarlo más información.
El modelo también debería incluir elementos que describan partes relevantes de su entorno, actores

Relaciones entre modelos

Un sistema contiene todas las relaciones y restricciones entre elementos incluidos en diferentes modelos
Las actividades relacionadas conforman flujos de trabajo
Un flujo de trabajo es un conjunto de actividades.
Identificamos primero los distintos tipos de trabajadores que participan. Identificamos los artefactos que necesitamos . Podemos describir como fluye el proceso a través de los diferentes trabajadores, y cómo ellos crean, producen y utilizan los artefactos de los demás

Procesos especializados

Un proceso de desarrollo de software del mundo real debe ser adaptable y configurable para cumplir con las necesidades reales de un proyecto y/o organización concreta.
Los factores principales que influyen en cómo se diferenciará el proceso son:

- Factores organizativos
- Factores del dominio
- Factores del ciclo de vida
- Factores técnicos

Méritos del proceso

- Todo equipo puede comprender lo que tiene que hacer
- Desarrolladores pueden comprender mejor lo que los otros están haciendo
- Los supervisores y directores pueden comprender lo que los desarrolladores están haciendo
- Los desarrolladores, supervisores y los directores pueden cambiar de proyecto sin tener que aprender un nuevo proceso
- La formación puede estandarizarse dentro de la empresa

- El devenir del desarrollo del software es repetible

Las herramientas influyen en el proceso

Las herramientas son buenas para automatizar procesos repetitivos, mantener las cosas estructuradas, gestionar grandes cantidades de información y para guiarnos a lo largo de un camino de desarrollo concreto.

El proceso dirige las herramientas

Las herramientas que implementan un proceso automatizado deben ser fáciles de usar. Debe proporcionar un incremento de productividad sustancial.

El modelo visual soporta UML

UML define reglas sintácticas que especifican cómo combinar elementos del lenguaje. La herramienta debe ser capaz de garantizar que se cumplan esas reglas.

Las herramientas dan soporte al ciclo de vida completo

- Gestión de requisitos: para almacenar, examinar, revisar, hacer el seguimiento y navegar por los diferentes requisitos de un proyecto software.
- Modelado visual: se utiliza para automatizar el uso de UML
- Herramientas de programación:editores, compiladores,depuradores,detectores de errores y analizadores de rendimiento
- Aseguramiento de la calidad: probar aplicaciones y componentes.

3- UN PROCESO DIRIGIDO POR CASOS DE USO

Las necesidades del cliente no son fáciles de discernir. Tenemos que capturar las necesidades del usuario de forma que puedan comunicarse fácilmente a todas las personas implicadas en el proyecto. Después diseñar una implementación funcional que se ajuste a estas necesidades. Luego verificar que las necesidades del cliente se han cumplido mediante la prueba del sistema.
Captura de requisitos tiene 2 objetivos, encontrar los verdaderos requisitos y representarlos de un modo adecuado para los usuarios, clientes, desarrolladores. Verdaderos requisitos son aquellos que cuando se implementan añaden el valor esperado para los usuarios. Descripción obtenida de los requisitos debe ser comprensible por usuarios y clientes.
Cada tipo de usuario se representa por un actor. Utilizan el sistema interactuando con los casos de uso. Un caso de uso es una secuencia de acciones que el sistema lleva a cabo para ofrecer algún resultado de valor para un actor.
Modelo de análisis compuesto por clasificadores y conjunto de realizaciones de casos de uso
Modelo de diseño es jerárquico, contiene relaciones que atraviesan la jerarquía, asociaciones, generalizaciones y dependencias, realizaciones de los casos de uso son estereotipos de colaboraciones, es un esquema de la implementación.

Razones para usar casos de uso:

- Proporcionan un medio sistemático e intuitivo de capturar requisitos funcionales.
- Dirigen todo el proceso de desarrollo debido a que la mayoría de las actividades como el análisis, diseño y prueba se llevan a cabo partiendo de los casos de uso.
- Los casos de uso se utilizan como contenedores de los requisitos no funcionales(de rendimiento, disponibilidad, exactitud).

En cada iteración elegimos un conjunto de casos de uso y los reflejamos en el modelo de análisis. Cada caso de uso se desarrolla como una realización de caso de uso, y cada realización de caso de uso tiene un conjunto de clasificadores participantes que desempeñan diferentes roles. Luego utilizaremos diagramas de colaboración para saber cómo interactúan sus responsabilidades para llevar a cabo realización de casos de uso.
Las responsabilidades de una clase son la recopilación de todos los roles que cumple en todas las realizaciones de casos de uso.
Un desarrollador responsable de una clase agrupa todos los roles de la clase en un conjunto completo de responsabilidades para la clase, y después los integra en un conjunto consistente de responsabilidades.
Subsistema: agrupamiento semánticamente útil de clases o de otros subsistemas. Posee un conjunto de interfaces que ofrece a sus usuarios. Los de bajo nivel son de servicio porque sus clases llevan a cabo un servicio. Unidad manejable de funcionalidad opcional.
Prueba de los casos de uso: pueden llevarse a cabo bien desde la perspectiva de un actor que considera el sistema como una caja negra, o bien desde una perspectiva de diseño, en la cual el caso de prueba se construye para verificar que las instancias de las clases participantes en la realización del caso de uso hacen lo que deberían hacer.

4- UN PROCESO CENTRADO EN LA ARQUITECTURA

La arquitectura de un sistema es la visión común en la que todos los empleados deben estar de acuerdo. Al arquitecto de software y a los desarrolladores les resulta útil presentar el sistema desde diferentes perspectivas para comprender mejor el diseño. Estas perspectivas son vistas del modelo de sistema. Todas las vistas juntas representan la arquitectura. Abarca decisiones importantes sobre:

- Organización de sistema software
- Elementos estructurales que compondrán el sistema y sus interfaces, junto con sus comportamientos, tal y como se especifican en las colaboraciones entre estos elementos
- Composición de los elementos estructurales y del comportamiento en subsistemas progresivamente más grandes.
- Estilo de la arquitectura que guía esta organización: los elementos y sus interfaces, colaboraciones y composición.

Descripción de la arquitectura tiene actores y casos de uso, pero solamente aquellos que son arquitectónicamente significativos.

La arquitectura es necesaria porque:

- Para que una organización desarrolle un sistema, dicho sistema debe ser comprendido por todos los que vayan a intervenir en él. El primer requisito que tiene lugar en una descripción de la arquitectura es que se debe capacitar a los desarrolladores, directivos, clientes y otros usuarios para comprender qué se está haciendo con suficiente detalle como para facilitar su propia participación.
- Una buena arquitectura es la que define explícitamente estas interfaces, haciendo que sea posible la reducción en la comunicación. Una interfaz bien definida comunica eficientemente a los desarrolladores de ambas partes qué necesitan saber sobre lo que los otros equipos están haciendo.
- Un buen arquitecto ayuda a los desarrolladores para que sepan dónde buscar elementos reutilizables de manera poco costosa, y para que sepan encontrar los componentes adecuados para ser reutilizados.
- El sistema debe ser en sí mismo flexible a los cambios o tolerante a los cambios. El sistema debe ser capaz de evolucionar sin problemas

Casos de uso y arquitectura:

La arquitectura no sólo se ve condicionada por los casos de uso arquitectónicamente significativos, sino también por los siguientes factores:

- sobre qué productos software del sistema queremos desarrollar.
- qué productos de middleware queremos utilizar.
- qué sistemas heredados queremos utilizar en nuestro sistema.
- a qué estándares y políticas corporativas debemos adaptarnos.
- Requisitos no funcionales generales.
- Las necesidades de distribución especifican cómo distribuir el sistema.

Pasos hacia una arquitectura:

Se desarrolla mediante iteraciones principalmente durante las fases de elaboración: la línea base de la arquitectura es un sistema pequeño y flaco. Tiene las versiones de todos los modelos que un sistema terminado contiene al final del a fase de construcción. Incluye el mismo esqueleto de subsistemas, componentes y nodos que un sistema definitivo, pero no existe toda la musculatura. Contiene comportamiento y código ejecutable.

La descripción de la arquitectura puede adoptar diferentes formas. Puede ser un extracto de los modelos que son parte de la línea base de la arquitectura o puede ser una reescritura de los extractos de forma que serán más fácil leerlos.

Hay muchas soluciones genéricas que han evolucionado a lo largo de muchos años y con las que todo arquitecto con experiencia deberá estar familiarizado. Estas soluciones se llaman patrones. Muchos están documentados en libros, que presentan los patrones utilizando plantillas estándar.Éstas asignan un nombre a un patrón y presentan un resumen de los problemas y las fuerzas que lo hacen surgir, una solución en términos de colaboración de clases participantes e interacción entre objetos de esas clases. Proporcionan ejemplos de cómo se utiliza el patrón en algunos lenguajes de programación.

Se implementan de una forma muy directa en lenguajes orientados a objetos.
Patrón: plantilla de colaboración, que es una colaboración general que puede especializarse según lo definido en la plantilla.
Una capa es un conjunto de subsistemas que comparten el mismo grado de generalidad y de volatilidad en las interfaces, las capas inferiores son de aplicación general a varias aplicaciones y deben poseer interfaces más estables, mientras que las capas más altas son más dependientes de la aplicación y pueden tener interfaces menos estables. Podemos aplicar sobre un mismo sistema muchos patrones de arquitectura.
Descripción de la arquitectura: es un extracto o un conjunto de vistas de los modelos que están en la línea base de la arquitectura. Incluye los elementos arquitectónicamente significativos.
La mayoría de las clases, con operaciones, interfaces y atributos que son privadas en los subsistemas o en los subsistemas de servicio no son significativos para la arquitectura. Los subsistemas que son variantes de otros subsistemas no son importantes desde una perspectiva de la arquitectura. Tampoco son arquitectónicamente relevantes la mayoría de las realizaciones de los casos de uso debido a que no imponen ninguna restricción adicional al sistema.
UML: posee construcciones potentes para la formulación de la arquitectura, y el proceso unificado nos ofrece directivas detalladas sobre lo que constituye una buena arquitectura. Es el resultado de un juicio basado en aptitudes y experiencia. El arquitecto es responsable de emitir este juicio.
Un arquitecto calificado tiene 2 aptitudes: Una es el conocimiento del dominio en el que trabaja, por lo que debe trabajar adquiriendo experiencia con los usuarios. El otro es el conocimiento del desarrollo de software incluso debe ser capaz de escribir código.
Arquitecto no debería ser jefe de proyecto ya que éste puesto tiene muchas dificultades además de la arquitectura.
La descripción de la arquitectura tiene 5 secciones: una para cada modelo. Tiene una vista del modelo de casos de uso, del modelo de análisis, del modelo de diseño, del modelo de despliegue y del modelo de implementación. No de prueba porque no desempeña ningún papel en la descripción de la arquitectura.

- Vista de la arquitectura del modelo de casos de uso: presenta los actores y casos de uso más importantes.
- Vista de la arquitectura del modelo de diseño: presenta los clasificadores más importantes para la arquitectura pertenecientes al modelo de diseño. Presenta cómo se realizan los casos de uso en términos de esos clasificadores, por medio de realizaciones de casos de uso.
- Vista de la arquitectura del modelo de despliegue: define la arquitectura física del sistema por medio de nodos interconectados.
- Vista de la arquitectura del modelo de implementación: correspondencia directa de los modelos de diseño y de despliegue.

¿Qué es la arquitectura?
Es la que especifica el arquitecto en la descripción de la arquitectura permite al arquitecto controlar el desarrollo del sistema desde la perspectiva técnica. Se centra tanto en los elementos estructurales significativos del sistema ,como subsistemas, clases, componentes y nodos, como en las colaboraciones que tienen lugar entre estos elementos a través de las interfaces.
Se desarrolla de forma iterativa durante la fase de elaboración pasando por los requisitos, el análisis. el diseño, la implementación y las pruebas.

5- UN PROCESO ITERATIVO E INCREMENTAL

Un proceso iterativo e incremental:
En la fase de inicio: el criterio esencial es la viabilidad, llevamos a cabo mediante.

- identificación y la reducción de los riesgos críticos para la viabilidad del sistema.
- creación de una arquitectura candidata a partir del desarrollo de un subconjunto clave de los requisitos.
- realización de una estimación inicial de coste, esfuerzo, calendario y calidad del producto con límites amplios.
- inicio del análisis del negocio.

En la fase de elaboración: el criterio es la capacidad de construir el sistema dentro de un marco de trabajo económico.

- identificación y reducción de los riesgos que afectan de manera significativa a la construcción del sistema.
- especificación de la mayoría de los casos de uso que representan la funcionalidad que ha de desarrollarse.
- extensión de la arquitectura candidata hasta las proporciones de una línea base
- preparación del plan del proyecto.
- realización de una estimación con sus límites suficientemente ajustados como para justificar la inversión.
- terminación del análisis del negocio.

fase de construcción: el criterio esencial es un sistema capaz de una cooperatividad inicial en el entorno del usuario

- viabilidad del sistema siempre es evidente en la forma de ejecutables.

Fase de transición: es criterio esencial es un sistema que alcanza una operatividad final llevada a cabo mediante:

- modificación del producto para subsanar problemas que no se identificaron en fases anteriores.
- corrección de defectos.

Iterativo e incremental
Estar dirigido por casos de uso significa que cada fase en el camino al producto final está relacionada con la que los usuarios hacen realmente. Estar centrado en la arquitectura significa que el trabajo de desarrollo se centra en obtener el patrón de la arquitectura que dirigirá la construcción del sistema en las primeras fases, garantizando un progreso continuo no sólo para la inversión en curso del producto, sino para la vida entera del mismo.

Desarrollo en pequeños pasos.
Dividimos el proyecto en un número de mini proyectos, siendo cada uno de ellos una iteración. Cada iteración tiene todo lo que tiene un proyecto de desarrollo de software. una iteración es una etapa dentro de un proyecto y se ve fuertemente condicionada por ello.
Éxito fundamental es conseguir una serie de iteraciones que siempre avanzan, nunca hay que volver dos o tres iteraciones atrás para corregir el modelo debido a algo que hemos aprendido en la última iteración

¿Por qué un desarrollo iterativo e incremental?

- Para tomar las riendas de los riesgos críticos y significativos desde el principio.
- Para poner en marcha una arquitectura que guíe el desarrollo del software.
- Para proporcionar un marco de trabajo que gestione de mejor forma los inevitables cambios en los requisitos y en otros aspectos.
- Para construir el sistema a lo largo del tiempo en lugar de hacerlo de una sola vez cerca del final, cuando el cambiar algo se ha vuelto costoso.
- Para proporcionar un proceso de desarrollo a través del cual el personal puede trabajar de manera más eficaz.

Desde la perspectiva de los usuarios y de otros interesados, es más productivo hacer evolucionar el producto a lo largo de una serie de versiones ejecutables o construcciones. Una construcción es una versión operativa de parte de un sistema que demuestra un subconjunto de posibilidades del sistema. Cada iteración debe progresar mediante una serie de construcciones hasta alcanzar el resultado esperado, el incremento.

Permitir cambios tácticos:
Los problemas se van descubriendo con un ritmo de goteo constante que los desarrolladores pueden tratar fácilmente.

La aproximación iterativa está dirigida por los riesgos:
Un riesgo es una variable del proyecto que pone en peligro o impide el éxito del proyecto. Riesgos importantes son temas de rendimiento disponibilidad, integridad de las interfaces de usuario adaptabilidad y portabilidad.
Intentamos ordenar las iteraciones de manera que cada una de ellas se construya sobre la anterior. Intentemos reducir así el riesgo concreto de que si no ordenamos bien las iteraciones podríamos tener que rehacer las iteraciones previas.

Categorías de los riesgos :

- Riesgos relacionados con nuevas tecnologías.
    - puede que haya que distribuir los procesos en muchos nodos, lo cual posiblemente origine problemas de sincronización.
    - Algunos casos de uso pueden depender de técnicas informáticas que aún no están bien
- Relativos a la arquitectura.
    - mediante el establecimiento temprano de una arquitectura que los riesgos, eliminamos el riesgo de no ser capaces de incorporar fácilmente los cambios.
    - Eliminamos el riesgo de tener que rechazar después una buena parte del trabajo.
    - otra ventaja de una arquitectura robusta mostrar dónde encajan los componentes reutilizables.
    - Reduce el riesgo de descubrir demasiado tarde que el sistema es demasiado caro de construir.
- Riesgos relativos a construir el sistema adecuado.
    - Subraya la importancia de identificar los requisitos funcionales y no funcionales.
    - Importante encontrar las funciones más importantes al principio para estar seguros de que se implementan al principio.
- Riesgos relativos al rendimiento
- Riesgos no técnicos son aquellos que una dirección atenta puede detectar y desviar.
    - La organización de desarrollo debería identificarlos, poner los medios daministrativos necesarios para seguir los desarrollos en cada una de las áreas de riesgo y garantizar que los directivos responsables emprendan acciones cuando uno de estos riesgos aparecen.

Para el Tratamiento de riesgo hay 4 acciones: evitarlo, limitarlo, atenuarlo o controlarlo

<u>Iteración genérica:</u>

iteración es un miniproyecto que obtiene como resultado una versión interna de trabajo.
El método iterativo no planifica el proyecto entero en detalle durante la fase de inicio, sólo da los primeros pasos.
Las primeras iteraciones del proyecto dan como resultado un mayor conocimiento de los requisitos, los problemas, los riesgos y el dominio de la solución, las siguientes dan como resultado incrementos aditivos que finalmente confirman la versión externa, el producto para el cliente.
El resultado de una iteración es un incremento.
Un incremento es la diferencia entre la versión interna de una iteración y la versión externa de la siguiente.
Al final de una iteración, el conjunto de modelos que representa al sistema queda en un estado concreto. Llamamos a este estado de la línea base.

Las iteraciones sobre el ciclo de vida

Los objetivos fundamentales de la fase de elaboración son obtener la línea base de la arquitectura, capturar la mayoría de los requisitos, y reducir los siguientes peores riesgos. Al final de esta fase, somos capaces de estimar los costos, y las fechas y de planificar la fase de construcción con algún detalle.

Los objetivos fundamentales de la fase de construcción son el desarrollo del sistema entero y la garantía de que el producto puede comenzar su transición a los clientes.

Fase de transición garantizar que tenemos un producto preparado para su entrega a la comunidad de usuarios.

Dentro de cada fase hay hitos menores. Al llegar a los hitos secundarios, desarrolladores deciden cómo continuar en las subsiguientes iteraciones. Al llegar a los hitos principales del final de las fases, los jefes toman decisiones cruciales de tipo continuar/no continuar.

Un hito secundario es un paso planificado hacia el hito principal al final de la fase. Hitos principales actúan como puntos de sincronización donde se juntan los dominios de gestión y técnicos.

Consecuencias de la aplicación del método iterativo e incremental:

- Para crear el análisis del negocio en la fase de inicio, la organización se ha centrado en la reducción de riesgos críticos y en demostrar una prueba de concepto.
- Para hacer una apuesta que merezca la pena para el negocio al término de la fase de elaboración la organización debe saber que va a contratar para desarrollo.
- Para minimizar los costos, los defectos, y el tiempo de salida al mercado, la organización debe emplear componentes reutilizables.
- Para evitar el retraso en la entrega, el sobrepasar los costes y obtener un producto de baja calidad, la organización debe hacer primero la parte difícil.
- Para evitar construir un producto fuera de plazos la organización no puede decir no a todos los cambios tozudamente.

6 - CAPTURA DE REQUISITOS: DE LA VISIÓN A LOS REQUISITOS

Captura de requisitos: Proceso de averiguar lo que se debe construir

Es difícil porque los usuarios no saben cuáles son los requisitos ni tampoco cómo especificarlos de una forma precisa.

El sistema debería proporcionar valor al negocio que lo utiliza y a sus clientes. Es difícil identificar o comprender qué es este valor, y a veces es imposible hacer que el sistema lo proporcione. Este valor escurridizo probablemente cambiará durante la vida del proyecto.

El propósito es guiar el desarrollo hacia el sistema correcto. El cliente debe ser capaz de leer y comprender el resultado de la captura de requisitos. Debemos utilizar el lenguaje del cliente para describir esos resultados.

Ciertos pasos son factibles en la mayoría de los casos, lo que nos permite sugerir un flujo de trabajo arquetípico.

Pasos:

- Enumerar los requisitos candidatos: lista de ideas que podemos decidir implementar en una versión futura del sistema. Esta lista de características se utiliza sólo para la planificación del trabajo. Cada característica tiene también un conjunto de valores de planificación que podríamos incluir: estado, coste estimado de implementación, prioridad, nivel de riesgo asociado a la implementación de la característica
- Comprender el contexto del sistema: para capturar los requisitos correctos y para construir el sistema correcto los desarrolladores clave requieren un firme conocimiento del contexto en el que se emplaza el sistema

2 aproximaciones para expresar el contexto:
modelado del dominio y modelado del negocio.
El modelado del dominio: describe los conceptos importantes del contexto como objetos del dominio y enlaza estos objetos unos con otros. Ayuda a desarrollar un glosario de términos que permitirá comunicarse mejor a todos los que están trabajando en el sistema. Ayudarán a identificar algunas de las clases.
Un modelo del negocio: puede describirse como un supraconjunto de un modelo del dominio, e incluye algo más que sólo los objetos del dominio.
Objetivo describir los procesos con el objetivo de comprenderlos.
El modelo del negocio especifica qué procesos de negocio soportará el sistema. Objetos del dominio o del negocio implicados en el negocio, establece las competencias requeridas en cada proceso.

- Capturar requisitos funcionales: se basa en los casos de uso. Estos casos de uso capturan tanto los requisitos funcionales como los no funcionales que son específicos de cada caso de uso. Para el usuario un caso de uso es un modo de utilizar el sistema.
- Capturar requisitos no funcionales: los requisitos no funcionales especifican propiedades del sistema, restricciones del entorno o de la implementación, rendimiento, dependencias de la plataforma, facilidad de mantenimiento, extensibilidad y fiabilidad. Los requisitos de rendimiento afectan sólo a ciertos casos de uso y por tanto deberían conectarse a ese caso de uso.
  Algunos requisitos no funcionales son más genéricos y no pueden relacionarse con un caso de uso o requisitos adicionales.

¿Qué es un modelo del dominio?
Este captura los tipos más importantes de objetos en el contexto del sistema. Los objetos del dominio representan las cosas que existen o los eventos que suceden. Las clases del dominio aparecen en tres formas típicas:

- Objetos del negocio que representan cosas que se manipulan en el negocio
- Objetos del mundo real y conceptos de los que el sistema debe hacer un seguimiento
- Sucesos que ocurrirán o han ocurrido

El modelo del dominio se describe mediante diagramas de UML.
El objetivo del modelado del dominio es comprender y describir las clases más importantes dentro del contexto del sistema.
Se utiliza: al desribir los casos de uso y al diseñar la interfaz de usuario, para sugerir clases internas al sistema en desarrollo durante el análisis.

Modelado de Negocio:
el objetivo: es identificar los casos de uso del software y las entidades de negocio relevantes que el software debe soportar, de forma que podríamos modelar sólo lo necesario para comprender el contexto.
El modelo del negocio está soportado por dos tipos de modelos UML: modelos de casos de uso y modelos de objetos.
El modelo de negocio describe los procesos de negocio de una empresa en términos de casos de uso del negocio y actores del negocio con los procesos del negocio y los clientes.
El modelo de casos de uso se describe mediante diagramas de casos de uso.
Un modelo de objetos del negocio describe cómo cada caso de uso de negocio es llevado a cabo por parte de un conjunto de trabajadores que utilizan un conjunto de entidades del negocio y de unidades de trabajo.
Una entidad de negocio representa algo que los trabajadores utilizan en un caso de uso del negocio. Una unidad de trabajo es un conjunto de esas entidades que conforman un todo reconocible para un usuario final.
Se desarrolla en dos pasos:

- Confeccionar un modelo de casos de uso del negocio que identifique los actores del negocio y unidades de trabajo que juntos realizan los casos de uso del negocio.
- Desarrollar un modelo de objetos del negocio compuesto por trabajadores, entidades de negocio, y unidades de trabajo que juntos realizan los casos de uso del negocio.

Podemos pensar en el modelado de dominio como en una variante simplificada del modelado de negocio, en el cual solo nos centramos en las cosas.
Diferencias: clases del dominio se obtienen de la base del conocimiento de unos pocos expertos del dominio o posiblemente del conocimiento asociado con sistemas similares al que estamos desarrollando, las entidades del negocio se derivan a partir de los clientes del negocio.
Negocio: cada entidad debe venir rotulada por su utilización en un caso de uso del negocio.
Dominio: puede hacer la traza de las clases hasta la experiencia de los expertos del dominio.
Negocio: puede hacer la traza de la necesidad de cada elemento del modelo hasta los clientes.
Las clases del dominio tienen atributos pero normalmente ninguna o muy pocas operaciones, no es así para las entidades del negocio.
Modelado del negocio y la técnica de ingeniería de software combinados en el proceso unificado nos permite hacer el seguimiento de las necesidades del cliente a lo largo del camino completo a través de procesos del negocio, trabajadores y casos de uso, hasta el código de software.
Cuando se utiliza solamente un modelo del dominio, no hay forma evidente de hacer la traza entre el modelo del dominio y los casos de uso del sistema.

Búsqueda de casos de uso a partir de un modelo del negocio
El analista identifica un actor por cada trabajador y por cada actor del negocio que se convertirá en usuario del sistema de información.
Por cada trabajador identificamos todas las realizaciones de casos de uso del negocio diferentes en las que participa.
Por cada papel de un trabajador o un actor del negocio, necesitamos un caso de uso para el actor del sistema correspondiente.
Para identificar casos de uso crear un caso de uso para el actor correspondiente a cada rol de cada trabajador y de cada actor del negocio
Requisitos adicionales: se capturan de forma muy parecida a como se hacía en la especificación de requisitos tradicional (lista de requisitos). Se utilizan durante el análisis y el diseño junto al modelo de casos de uso.
Un requisito de interfaz especifica la interfaz con un elemento externo.
Un requisito físico especifica una característica física que debe poseer un sistema.
Una restricción de diseño limita el diseño de un sistema (extenisbilidad y mantenibilidad).
Una restricción de implementación especifica o limita la codificación o construcción de un sistema.

7 - CAPTURA DE REQUISITOS COMO CASOS DE USO

Mediante la utilización de los casos de uso, los analistas se ven obligados a pensar en términos de quiénes son los usuarios y qué necesidades u objetivos de la empresa pueden cumplir. Su papel es clave en la dirección del resto del trabajo de desarrollo.

Artefactos:
Es el modelo de casos de uso, que incluye los casos de uso y los actores, prototipos de interfaz de usuario.

- Modelo de casos de uso: permite que los desarrolladores de software y los clientes lleguen a un acuerdo sobre los requisitos. Contiene actores, casos de uso y sus relaciones.
- UML: nos permite presentar el modelo en diagramas que muestran los actores y los casos de uso desde diferentes puntos de vista y con diferentes propósitos. Puede ser útil introducir paquetes.
- Actor: tipo de usuario. Se representa mediante uno o más actores cada sistema externo con el que interactúa el sistema. Representan terceros fuera del sistema que colaboran con el sistema. Entorno externo al sistema.
- Casos de uso: son fragmentos de funcionalidad que el sistema ofrece para aportar un resultado de valor para sus actores. Especifica una secuencia de acciones que el sistema puede llevar a cabo interactuando con sus actores. Es un clasificador, tiene operaciones y atributos.

    - Diagramas de estados: especifican el ciclo de vida de las instancias de los casos de uso en términos de estados y transiciones entre los estados.
    - Los diagramas de actividad: describen el ciclo de vida con más detalle describiendo también la secuencia temporal de acciones que tiene lugar dentro de cada transición.
    - Diagramas de colaboración y secuencia: se emplean para describir las interacciones.
    - Una instancia de caso de uso es la realización de casos de uso.

La mayoría de las veces es una instancia de un actor la que invoca a la instancia del caso de uso, pero también puede ser un evento interno al sistema.
Los casos de uso tienen atributos que representan los valores que una instancia de un caso de uso utiliza y manipula durante la ejecución de sus caso de uso. Estos valores son locales a la instancia del caso de uso. El único tipo de interacciones en el modelo de casos de uso tiene lugar entre instancias de actores e instancias de casos de uso. Las instancias de los Casos de Uso son atómicas

Flujo de sucesos:
Descripción textual de la secuencia de acciones del caso de uso. Incluye un conjunto de secuencias de acciones que pueden ser modificadas, revisadas, diseñadas, implementadas y probadas juntas y pueden ser descritas como una sección o subsección del manual de usuario.
Requisitos especiales: descripción textual que agrupa todos los requisitos del tipo de los requisitos no funcionales sobre el caso de uso.

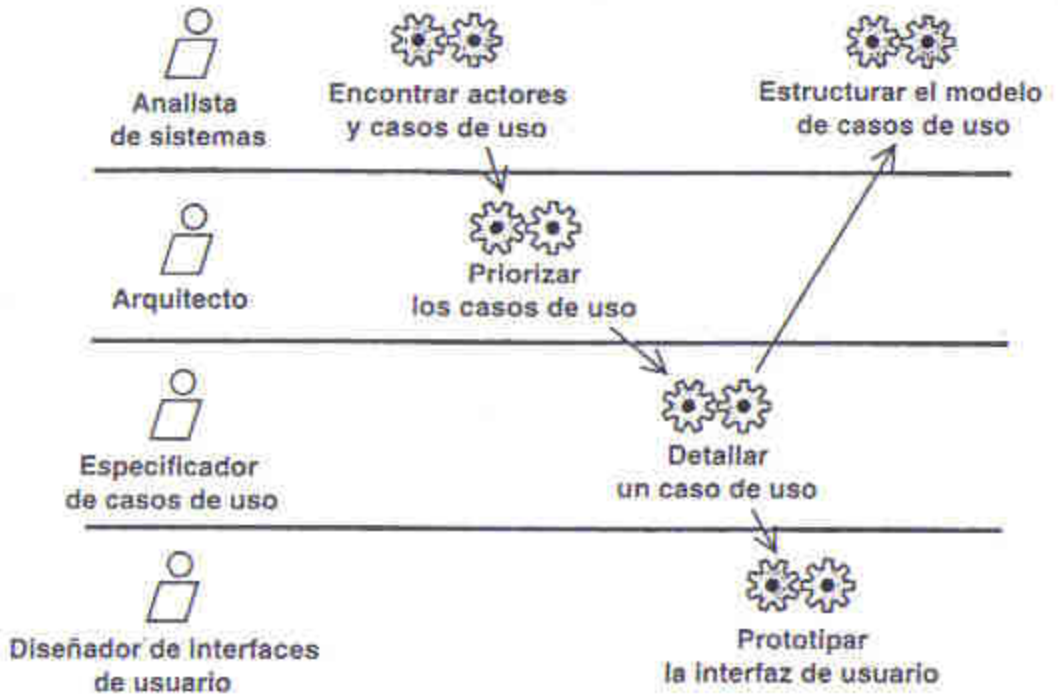
- Artefacto Descripción de la arquitectura: vista de la arquitectura del modelo de casos de uso deberán incluir los casos de uso que describen alguna funcionalidad importante y crítica o que impliquen algún requisito importante que deba desarrollarse pronto dentro del ciclo de vida del software.
- Artefacto glosario: para definir términos comunes importantes que los analistas utilizan al describir el sistema.
- Artefacto prototipo de interfaz de usuario: nos ayudan a comprender y especificar las interacciones entre actores humanos y el sistema durante la captura de requisitos

Trabajador:
Puesto al cual se puede asignar una persona real. Una misma persona puede estar asignada a diferentes trabajadores durante un proyecto.

- Analista de sistemas: responsable del conjunto de requisitos que están modelando en los casos de uso, los que incluyen todos los requisitos funcionales y no funcionales que son casos de uso especìficos. Responsable de delimitar el sistema. No es responsable de cada caso de uso en particular. Dirige el modelado y coordina la captura de requisitos.
- Especificador de casos de uso: necesita trabajar con los usuarios reales de sus casos de uso.

- Diseñador de interfaz de usuario: dan forma visual a las interfaces del usuario
- Arquitecto: participa en el flujo de trabajo de los requisitos para describir la vista de la arquitectura del modelo de casos de uso.

Flujo de trabajo:



- Encontrar actores y casos de uso: para delimitar el sistema de su entorno, esbozar quién y qué interactuará con el sistema y qué funcionalidad se espera del sistema. Capturar y definir un glosario de términos comunes esenciales para la creación de descripciones detalladas de las funcionalidades del sistema. Esto consta de 4 pasos: encontrar actores, encontrar casos de uso, describir básicamente cada caso de uso, describir el modelo de casos de uso completo.
    - Encontrar actores: con modelado de negocio es sencillo. El analista puede asignar un actor a cada trabajador del negocio y un actor a cada actor del negocio. Debería existir una coincidencia mínima entre los roles que desempeñan las instancias de los diferentes actores en relación con el sistema.
    - Encontrar los casos de uso: analista de sistemas va repasando los actores uno por uno y proponiendo los casos de uso por cada actor.
      Algunos de los candidatos no llegarán a casos de uso por sí mismos, podrán ser parte de otros casos de uso. Elegimos un nombre para cada caso de uso de forma que nos haga pensar en la secuencia de acciones concreta que añade valor a un actor.

Criterios útiles para identificar casos de uso:
- Resultados de valor
- un actor en concreto.

- Describir básicamente cada caso de uso: necesitamos explicar cómo están relacionados entre sí los casos de uso y los actores y cómo juntos constituyen el modelo de casos de uso.
- Descripción del modelo de casos de uso en su totalidad

Cuando la descripción del modelo de casos de uso está preparada, dejamos que dé el visto bueno del modelo de casos de uso la gente que no forma parte del equipo de desarrollo convocando una revisión informal para determinar si se han capturado como casos de uso todos los requisitos funcionales necesarios, la secuencia de acciones es correcta, completa y comprensible para cada caso de uso. Se identifica algún caso de uso que no proporcione valor.

- Priorizar casos de uso: determinar cuáles son los necesarios para el desarrollo en las 1º iteraciones y cuáles pueden dejarse para más tarde. Los resultados se recogen en la vista de la arquitectura del modelo de casos de uso. Esta planificación también necesita la consideración de otros aspectos no técnicos, como los aspectos económicos o del negocio del sistema que va a ser desarrollado.
- Detallar un caso de uso: objetivo describir su flujo de sucesos en detalle, incluyendo cómo comienza, termina e interactúa con los actores. El resultado de esta actividad es la descripción detallada de un caso de uso en particular en forma de texto y diagramas.
  - Estructuración de la descripción de casos de uso: describir la posible transición de estados de memoria simple y precisa. Elegir un camino básico completo y describir ese camino en una sección de la descripción. Podemos describir en secciones separadas el resto de los caminos como caminos alternativos o desviaciones del camino básico. Alternativas pueden ocurrir por muchas razones, actor puede elegir entre diferentes caminos en el caso de uso, si está implicado más de un actor en el caso de uso, las acciones de uno de ellos pueden influenciar el camino de las acciones del resto. El sistema puede detectar entradas erróneas de los actores. Algunos recursos del sistema pueden tener un mal funcionamiento.
    El camino básico elegido debe ser el camino normal, el más habitual y que proporciona el valor más obvio al actor.
  - Descripción de casos de uso debe incluir: estado inicial como precondición, cómo y cuándo comienza el caso de uso, orden en que las acciones se deben ejecutar, posibles estados finales como postcondiciones, caminos de ejecución que no están permitidos, descripciones de caminos alternativos, interacción del sistema con los actores y qué cambios producen. Utilización de objetos, valores y recursos de sistema, requisitos no funcionales.

    - Formalización de la descripción de casos de uso: pueden utilizarse los diagramas de estado de UML para describir los estados de los casos de uso y las transiciones entre estados con más detalles como secuencias de acciones. Se pueden utilizar los diagramas de interacción para describir cómo interactúa una instancia de casos de uso con la instancia de un actor.
- Prototipar la interfaz de usuario: comenzamos con los casos de uso e intentamos discernir qué se necesita de las interfaces de usuario para habilitar los casos de uso para cada actor. Creamos el diseño físico de la interfaz de usuario y desarrollamos prototipos que ilustran cómo pueden utilizar el sistema los usuarios para ejecutar los casos de uso.
  El resultado final es un conjunto de interfaces de esquemas de interfaces de usuario y prototipos de interfaces de usuario que especifican la apariencia de esas interfaces de cara a los actores más importantes.
- Estructurar el modelo de casos de uso: para extraer descripciones de funcionalidad generales y compartidas que pueden ser utilizadas por descripciones más específicas, extraer descripciones de funcionalidad adicionales u opcionales que pueden extender descripciones más específicas.
  Debemos ir buscando acciones o parte de acciones comunes o compartidas por varios casos de uso. Esta compartición puede extraerse y describirse en un caso de uso separado que puede ser después reutilizado por el caso de uso original. Mostramos la relación de reutilización mediante una generalización. La otra relación es extensión, modela la adición de una secuencia de acciones a un caso de uso. Se comporta como si fuera algo que se añade a la descripción original de un caso de uso. Incluye condición para la extensión y referencia a un punto de extensión en el caso de uso destino. Una vez que una instancia de un caso de uso llega al punto de extensión al cual se refiere una relación de extensión, se evalúa la condición de la relación. Si la condición se cumple, la secuencia seguida por la instancia del caso de uso incluye la secuencia del caso de uso extendido. Especifica el comportamiento adicional para otros casos de uso. Otra relación inclusión proporciona una extensión explícita e incondicional a un caso de uso.

8 - ANÁLISIS

Objetivo es conseguir una comprensión más precisa de los requisitos y una descripción de los mismos que sea fácil de mantener y que nos ayude a estructurar el sistema externo.
Los casos de uso deben mantenerse tan independientes como sea posible. Aspectos relativos a la interferencia, la concurrencia y los conflictos entre casos de uso pueden quedar sin resolver en la captura de requisitos. Los casos de uso deben describirse utilizando el lenguaje del cliente, perdemos poder expresivo. Debe estructurarse cada caso de uso para que forme una especificación de funcionalidad completa e intuitiva. Aspectos relativos a esas redundancias entre requisitos descritos puede que queden sin resolver durante captura de requisitos.

El Análisis puede utilizar lenguajes de los desarrolladores podemos usar un lenguaje más formal para apuntar detalles relativos a los requisitos del sistema.

Comparación:

Modelo de Casos de Uso	Modelo de Análisis
Descrito con lenguajes del cliente	Descrito con lenguaje del desarrollador
Visión externa del sistema	Visión interna del sistema
Estructurado por casos de uso, proporciona la estructura a la vista externa	Estructurado por clases y paquetes, proporciona la estructura a la vista interna
Utilizada como contrato entre cliente y desarrollador sobre qué debería hacer el sistema	Utilizado por desarrolladores para comprender cómo debería darse forma al sistema
Puede contener redundancias, inconsistencias entre requisitos	No debe contener redundancias, inconsistencias entre requisitos
Captura la funcionalidad del sistema, incluida la funcionalidad significativa para la arquitectura	Esboza cómo llevar a cabo la funcionalidad dentro del sistema. 1º aproximación al diseño
Define casos de uso que se analizarán con más profundidad en el modelo de análisis	Define realizaciones de casos de uso, y cada una de ellas representa el análisis de un caso de uso del modelo de caso de uso

Análisis permite razonar sobre los aspectos internos del sistema. Ofrece mayor poder expresivo y una mayor formalización. Proporciona una estructura centrada en el mantenimiento, flexibilidad ante los cambios y reutilización. Primera aproximación al modelo del diseño. Llevando a cabo el análisis conseguimos una separación de intereses que prepara y simplifica las subsiguientes actividades de diseño e implementación, delimitando los temas que deben resolverse y las decisiones que deben tomarse en esas actividades.

¿Cuándo hacer análisis?

- Podemos analizar sin grandes costos una gran parte del sistema y podemos utilizar el resultado para planificar el trabajo de diseño e implementación subsiguientes.
- Visión general puede ser muy valiosa para recién llegados al sistema o para desarrolladores que en general mantienen el sistema proporciona una vista conceptual, precisa y unificadora de implementaciones alternativas. Cuando se construyen sistemas utilizando un sistema heredado complejo.

Tres variaciones:

- Usar modelo de análisis para describir resultados del análisis y mantener la consistencia del modelo a lo largo de todo el ciclo de vida del software.
- Usar modelo para describir resultados del análisis pero considerarlo como herramienta transitorio e intermedia
- No usarlo

Artefactos:

- Modelo de análisis: clases de análisis representan abstracciones de clases y de subsistemas del diseño del sistema. Los casos de uso se describen mediante clases de análisis y sus objetos.
- Clase del análisis: se centra en el tratamiento de los requisitos funcionales y pospone los no funcionales. Más evidente en el contexto del dominio del problema. Raramente define u ofrece una interfaz en términos de operaciones y de sus signaturas. Define atributos conceptuales y reconocibles en el dominio del problema. PArticipa en relaciones más conceptuales. Encajan en uno de tres estereotipos:
    i. De interfaz: se usa para modelar la interacción entre el sistema y sus actores. Representan abstracciones de ventanas, formularios, paneles. no es necesario que describan cómo se ejecuta físicamente la interacción. Debe asociarse al menos con un actor.
    ii. Entidad: para modelar información que posee una vida larga y persistente. Se deriva directamente de una clase de entidad del negocio. Diferencia es que las entidades representan objetos manejados por el sistema en consideración, mientras que las últimas representan objetos presentes en el negocio.
    iii. De Control: representa coordinación, secuencia, transacciones y control de otros objetos y se usan con frecuencia para encapsular el control de un caso de uso en concreto. Aspectos dinámicos del sistema se modelan con clases de control. Encapsulan los cambios del control, coordinación, secuencia, transacciones y lógica del negocio compleja que implica a varios otros objetos
- Realización de casos de uso-análisis: colaboración dentro del modelo de análisis que describe cómo se lleva a cabo y ejecuta un caso de uso determinado en términos de las clases del análisis y de sus objetos del análisis en interacción.
  Posee una descripción textual del flujo de sucesos, diagramas de clases que muestran sus clases del análisis participantes, y diagramas de interacción que muestran la realización de un flujo.
    - Diagramas de clases: es importante durante el análisis coordinar todos los requisitos sobre una clase y sus objetos que pueden tener diferentes casos de uso.

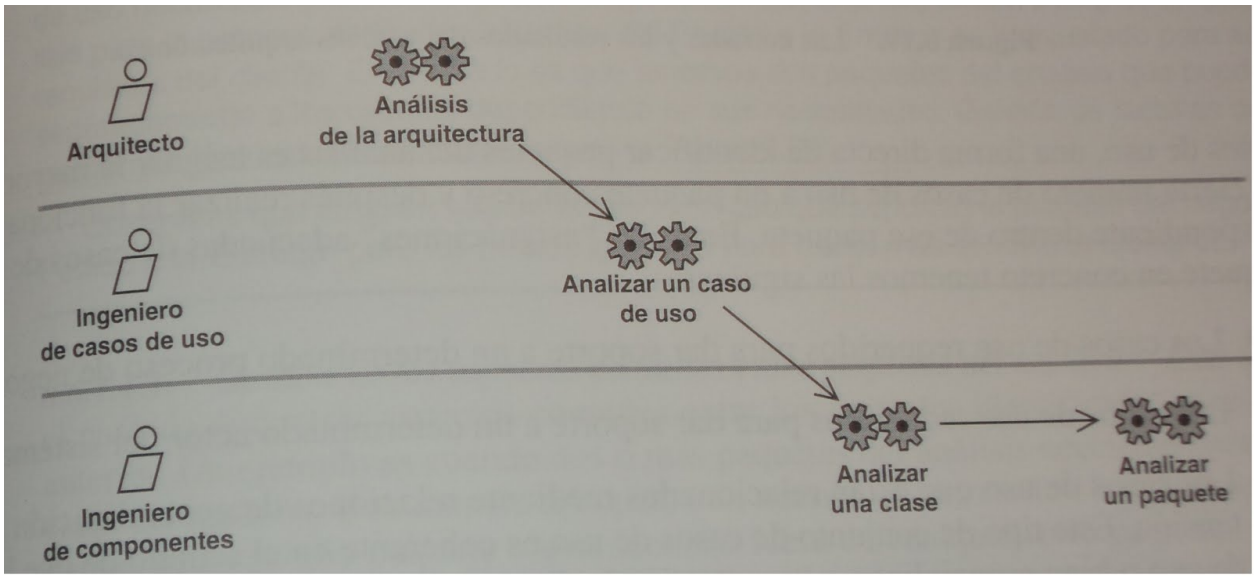
    - Diagramas de interacción: en los diagramas de colaboración mostramos interacciones entre objetos creando enlaces entre ellos y añadiendo mensajes a esos enlaces.
      Un objeto de interfaz no tiene porqué ser particular de una realización de casos de uso si debe aparecer en una ventana y participar en dos o más instancias de caso de uso.
      Un objeto de entidad normalmente no es particular de una realización de casos de uso. Las clases de control suelen encapsular el control asociado con un caso de uso concreto.
    - Requisitos especiales: descripciones textuales que recogen todos los requisitos no funcionales sobre una realización de casos de uso. Algunos de ellos pueden ser requisitos nuevos o derivados que se encuentran a medida que alcanza el trabajo de análisis.
- Artefacto: paquete de análisis: medio para organizar los artefactos del modelo de análisis en piezas manejables. Deben ser cohesivos(contenidos fuertemente relacionados) y débilmente acoplados (dependencias unos de otros minimizadas). Características: pueden representar una separación de intereses del análisis. Deberían crearse basándonos en los requisitos funcionales y en el dominio del problema. No deberían basarse en requisitos no funcionales o en el dominio de la solución. Probablemente se convertirán en subsistemas en las dos capas de aplicación superiores del modelo de diseño, o se distribuirán entre ellos.
    - Paquetes de servicio: cliente adquiere una combinación adecuada de servicios para ofrecer a su usuario los casos de uso necesarios para llevar a cabo su negocio: un caso de uso especifica una secuencia de acciones. Un servicio representa un conjunto coherente de acciones relacionadas funcionalmente que se utiliza en varios casos de uso. Los casos de uso son para los usuarios, y los servicios son para el cliente. Un paquete de servicio contiene un conjunto de clases relacionadas funcionalmente, es indivisible, depende a menudo de otro paquete de servicio, sólo es relevante para unos pocos actores, pueden representar cierta funcionalidad adicional del sistema, pueden ser mutuamente excluyentes, pueden representar diferentes aspectos o variantes del mismo servicio, constituyen una entrada fundamental para las actividades de diseño e implementación subsiguientes, ayudarán a estructurar los modelos de diseño e implementación en términos de subsistemas de servicios.
      Esto da como resultado un sistema robusto que es resistente al cambio.
      Los paquetes de servicio cuyos servicios se centran alrededor de una o más clases de entidad son probablemente reutilizables en diferentes sistemas que soporten el mismo negocio o dominio.

- Artefacto: descripción de la arquitectura: contiene una vista de la arquitectura del modelo de análisis que muestra sus artefactos significativos para la arquitectura.
    - Artefactos significativos: descomposición del modelo de análisis en paquetes de análisis y sus dependencias, clases fundamentales del análisis como las clases de entidad que encapsulan un fenómeno importante del dominio del problema. Suele ser suficiente con considerar significativa para la arquitectura una clase abstracta pero no sus subclases, realizaciones de casos de uso que describen cierta funcionalidad importante y crítica.

Trabajadores:

- Arquitecto: responsable de la integridad del modelo de análisis, garantizando que éste sea correcto, consistente y legible como un todo. responsable de la arquitectura del modelo de análisis. No responsable del desarrollo y mantenimiento continuo de los diferentes artefactos del modelo de análisis.
- Ingeniero de casos de uso: responsable de la integridad de una o más realizaciones de caso de uso, garantizando que cumplen los requisitos que recaen sobre ellos. No es responsable de las clases de análisis ni de las relaciones que se usan en la realización del caso de uso.
- Ingeniero de componentes: define y mantiene las responsabilidades, atributos, relaciones y requisitos especiales de una o varias clases del análisis, asegurándose de que cada clase del análisis cumple los requisitos que se esperan de ella de acuerdo a las realizaciones de caso de uso en las que participan.
  Mantiene la integridad de uno o varios paquetes del análisis. Responsable de un paquete del análisis lo será también de las clases del análisis contenidas en él.

Flujo de trabajo:



Los arquitectos comienzan la creación del modelo de análisis identificando los paquetes de análisis principales, las clases de entidad evidentes y los requisitos comunes. Los ingenieros de casos de uso realizan cada caso de uso en términos de las clases del análisis participantes exponiendo los requisitos de comportamiento de cada clase. Ingeniero de componentes especifican posteriormente estos requisitos y los integran dentro de cada clase creando responsabilidades, atributos y relaciones consistentes para cada clase.

- Actividad: análisis de la arquitectura: el propósito es esbozar el modelo de análisis y la arquitectura mediante la identificación de paquetes de análisis, clases de análisis evidentes y requisitos especiales comunes.
    - Identificación de paquetes de análisis: asignar la mayor parte de un cierto número de casos de uso a un paquete concreto y después realizar la funcionalidad correspondiente dentro de ese paquete. Entre las asiganaciones adecuadas de CU a un paquete en concreto tenemos las siguientes:
        - Los casos de uso requeridos para dar soporte a un determinado proceso de negocio.
        - Los casos de uso requeridos para dar soporte a un determinado actor del sistema.
        - Los casos de uso que están relacionados mediante relaciones de generalización y de extensión. Este tipo de conjunto de casos de uso es coherente en el sentido de que los casos de uso o bien especializan o extienden a los otros.
    - Identificación de paquetes de servicio: identificar un paquete de servicio por cada servicio opcional. Será unidad de compra. Identificar un paquete de servicio por cada servicio que podría hacerse opcional, incluso aunque todos los clientes siempre lo quieran.
    - Definición de dependencias entre paquetes de análisis: deberían definirse dependencias entre los paquetes del análisis si sus contenidos están relacionados. Dirección debe ser la misma de la relación. Intentar reducir el número de relaciones entre clases en paquetes diferentes debido a que ello reduce las dependencias entre paquetes. Esbozo inicial de las clases significativas para la arquitectura. Clase de entidad que no participa en una realización de casos de uso no es necesaria.

- Actividad: analizar un caso de uso: analizamos un CU para:
        - Identificar las clases del análisis cuyos objetivos son necesarios para llevar a cabo el flujo de sucesos del caso de uso
        - distribuir el comportamiento del caso de uso entre los objetos del análisis que interactúan
        - capturar requisitos especiales sobre la realización del caso de uso.

- Identificación de clases del análisis: identificamos las clases de control, entidad, e interfaz necesarios para realizar los casos de uso y esbozamos sus nombres, responsabilidades, atributos y relaciones. Para identificar las clases del análisis puede que tengamos que refinar las descripciones de los casos de uso en lo referente al interior del sistema.
  - Normas generales:
    1. identificar clases de entidad mediante el estudio en detalle de la descripción del caso de uso y de cualquier modelo del dominio que se tenga y considerar qué información debe utilizarse y manipularse en la realización del caso de uso.
    2. Identificar una clase de interfaz central para cada actor humano.
    3. Identificar una clase de interfaz primitiva para cada actor que sea un sistema externo.
    4. Identificar una clase de control responsable del tratamiento del control y de la coordinación de la realización del caso de uso y refinar esta clase de control de acuerdo a los requisitos del caso de uso.
- Descripción de interacciones entre objetos del análisis: cuando tenemos un esbozo del as clases necesarias para realizar el caso de uso, debemos describir cómo interactúan sus correspondientes objetos del análisis mediante diagramas de colaboración. Se crea comenzando por el principio del flujo del caso de uso y continuando el flujo paso a paso decidiendo qué interacciones de objetos del análisis y de instancias de actor son necesarias para realizarlo.
  - Observamos en los diagramas de colaboración:
    1. Caso de uso se invoca mediante un mensaje proveniente de una instancia de un actor sobre un objeto de interfaz.
    2. Cada clase del análisis identificada en el paso anterior debería tener al menos un objeto que participará en un diagrama de colaboración.
    3. Los mensajes no se asocian a operaciones debido a que no especificamos operaciones en las clases del análisis.
    4. Los enlaces en el diagrama normalmente deben ser instancias de asociaciones entre clases del análisis.
    5. La secuencia en el diagrama no debería ser nuestro objetivo principal y puede eliminarse si es dificil de mantener o crea confusión en el diagrama.
    6. El diagrama de colaboración debería tratar todas las relaciones del caso de uso que se está realizando.

- Actividad: analizar una clase:
    - *Objetivos:*
        - identificar y mantener las responsabilidades de una clase del análisis, basadas en su papel en las realizaciones de caso de uso,
        - identificar y mantener los atributos y relaciones de la clase del análisis,
        - capturar requisitos especiales sobre la realización de la clase del análisis.
    - Identificar responsabilidades: pueden recopilarse combinando todos los roles que cumple en distintas realizaciones de caso de uso. Los requisitos de cada realización de caso de uso respecto a sus clases a veces están escritos textualmente en el artefacto flujo de sucesos-análisis. Técnica extraer responsabilidades de cada rol una detrás de otro, añadiendo responsabilidades adicionales o modificando las existentes basándonos en una realización de caso de uso tras otra.
    - Identificación de atributos: atributos especifican una propiedad de una clase del análisis. Normas: nombre de un atributo debería ser un nombre [1,2]. Tipo conceptual del análisis: reutilizar tipos ya existentes. Instancia de atributo no puede compartirse por varios objetos del análisis. Si se hace demasiado difícil entender una clase de análisis por sus atributos, separarlos en otras clases.
    - Identificación de asociaciones y agregaciones: objetos del análisis interactúan unos con otros mediante enlaces en los diagramas de colaboración. Suelen ser instancias de asociaciones entre sus correspondientes clases. Debemos Minimizar número de relaciones entre clases. Agregaciones cuando objetos representan conceptos que se contienen físicamente uno al otro, que están compuestos uno del otro, que forman una colección conceptual de objetos
    - Identificación de generalizaciones: para extraer comportamiento compartido y común entre varias clases del análisis diferentes.
    - Captura de requisitos especiales: recogemos todos los requisitos de una clase del análisis que se han identificado en el análisis pero que deberían tratarse en el diseño y la implementación (requisitos no funcionales).

- Actividad: analizar un paquete:
    - *Objetivo:*
        - garantizar que el paquete es tan independiente como sea posible,
        - que cumpla su objetivo de realizar algunas clases o casos de uso,
        - descubrir dependencias de forma que pueda estimarse el efecto de cambios futuros.
    - *Normas:*
        - definir y mantener las dependencias con otros paquetes cuyas clases contenidas están asociadas con él,
        - asegurarnos de que el paquete contiene las clases correctos.
        - Limitar dependencias con otros paquetes.

9 - DISEÑO

Modelamos el sistema y encontramos su forma para que soporte todos los requisitos.
Propósitos:

- adquirir una comprensión en profundidad de los aspectos relacionados con los requisitos no funcionales y restricciones.
- Crear una entrada apropiada y un punto de partida para actividades implementación subsiguientes capturando los requisitos o subsistemas individuales, interfaces y clases.
- Ser capaces de descomponer los trabajos de implementación en partes más manejables que puedan ser llevadas a cabo por diferentes equipos de desarrollo
- Capturar las interfaces entre los subsistemas antes en el ciclo de vida del software.
- Ser capaces de visualizar y reflexionar sobre el diseño utilizando notación común
- Crear una abstracción de implementación del sistema.

Modelo de análisis	**Modelo del diseño**
Modelo conceptual, porque es una abstracción del sistema y permite aspectos de la implementación	Modelo físico, porque es un plano de la implementación
Genérico respecto al diseño	No genérico, específico para una implementación
Estereotipos clases: control, entidad, interfaz	Cualquier número de estereotipos sobre las clases
Menos formal	Más formal
Menos caro de desarrollar	Más caro de desarrollar
Menos capas	Más capas
Dinámico (no muy centrado en secuencias)	Dinámico(muy centrado en secuencias)
Bosquejo del diseño del sistema, incluyendo arquitectura	Manifiesto del diseño de sistema incluyendo arquitectura
Creado como trabajo de a pie	Creado como programación visual
puede no mantenerse durante todo el ciclo de vida de software	Debe mantenerse durante todo el ciclo de la vida del software
Define estructura que es entrada esencial para modelar el sistema	Da forma al sistema mientras que intenta preservar la estructura definida por el modelo de análisis lo más posible

El diseño es el centro de atención del diseño es al final de fase de elaboración y comienzo de iteraciones de construcción.

Artefactos:
1- Modelo de diseño: modelo de objetos que describe la realización física de los casos de uso centrándose en cómo los requisitos funcionales y no funcionales junto con otras restricciones con el entorno de implementación, tienen impacto en el sistema a considerar. Se representa por un sistema de diseño que denota el subsistema de nivel más alto del modelo. Los subsistemas de diseño y clases del diseño representan abstracciones del subsistema y componentes de la implementación del sistema. Los casos de uso son realizados por las clases de diseño y sus objetos. Se representa por colaboraciones y denota realización de caso de uso-diseño.

2- Clase de diseño: abstracción sin costuras de una clase o construcción similar en la implementación del sistema

- lenguaje utilizado para especificar una clase de diseño es lo mismo que el lenguaje de programación
- visibilidad de los atributos y operaciones se especifica con frecuencia . Relaciones de aquellas clases del diseño implicadas con otras clases, a menudo tienen un significado directo cuando la clase es implementada
- Los métodos de una clase de diseño tienen correspondencia directa con el correspondiente método en implementación de clases
- Clase de diseño puede posponer el manejo de algunos requisitos para las subsiguientes actividades de implementación indicándolas como requisitos de implementación de clase
- Clase de diseño a menudo aparece como un estereotipo sin costuras que se corresponde con una construcción en el lenguaje de programación dado
- Una clase de diseño puede realizar interfaces si tiene sentido hacerlo con el lenguaje de programación
- Una clase de diseño puede activarse implicando que objetos de la clase mantengan su propio hilo de control y se ejecuten concurrentemente con otros objetos activos

3- Realización caso de uso-diseño: colaboran en el modelo de diseño que describe cómo se realiza un caso de uso específico y cómo se ejecuta en términos de clase de diseño y sus objetos.Tiene una descripción de flujo de eventos textual, diagramas de clases que muestra sus clases de diseño participantes y diagramas de interacción que muestran la realización de un flujo o escenario concreto de un caso de uso en términos de interacción entre objetos del diseño

- Diagrama de clases: clase de diseño y sus objetos y de ese modo también los subsistemas que contienen las clases de diseño a menudo participan en varias realizaciones de casos de uso.
- Diagramas de interacción: interacciones entre objetos mediante transferencia de mensajes entre objetos o subsistemas.
- Flujo de sucesos-diseño: descripción textual que explica y complementa a los

diagramas y a sus etiquetas.
- Requisitos de la implementación: descripción textual que recoge requisitos sobre una realización de caso de uso.

4- Subsistema de diseño forma de organizar los artefactos de modelo de diseño en piezas más manejables. Puede constar de clases de diseño, realizaciones de caso de uso, interfaces y otros subsistemas. Debería ser cohesivo; deberían ser débilmente acoplados. Pueden representar una separación de aspectos del diseño.

- Las dos capas de aplicación de más alto nivel y sus subsistemas dentro del modelo de diseño suelen tener trazas directas hacia paquetes y/o clases del análisis. Los subsistemas pueden representar componentes de grano grueso en la implementación del sistema.
- Los subsistemas pueden representar productos software reutilizados que han sido encapsulados en ellos.
- Los subsistemas pueden representar sistemas heredados encapsulándolos.
- Subsistemas de servicio. se utilizan en un nivel inferior de la jerarquía de subsistemas de diseño por el mismo motivo por el cual utilizábamos los paquetes de servicio en el modelo de diseño.

5- Interfaz: se utilizan para especificar las operaciones que proporcionan las clases y los subsistemas del diseño.

6- Descripción de la arquitectura: contiene una vista de la arquitectura del modelo de diseño que muestra sus artefactos relevantes para la arquitectura. Significativos: descomposición del modelo de diseño en subsistemas, interfaces y dependencias .Clases del diseño fundamentales (clases activas, clases del análisis significativas, clases del diseño que sean generales y centrales, que representen mecanismos de diseño genéricos)

- Realizaciones de casos de uso-diseño que describen alguna funcionalidad importante y crítica que debe desarrollarse pronto dentro del ciclo de vida del software, que impliquen muchas clases del diseño.

7- Modelo de despliegue: modelo de objetos que describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos de cómputo. Cada nodo representa un recurso de cómputo.

- Los nodos poseen relaciones que representan medios de comunicación entre ellos
- Modelo de despliegue puede describir diferentes configuraciones de red, incluidas las configuraciones para pruebas y para simulación.
- Funcionalidad de un nodo se define por los componentes que se distribuyen sobre ese nodo.Modelo de despliegue en sí mismo representa una correspondencia entre la arquitectura software y la arquitectura del sistema.

8- Descripción de la arquitectura: contiene una vista de la arquitectura del modelo de despliegue que muestra sus artefactos relevantes para la arquitectura.

Trabajadores

- Arquitecto: responsable de la integridad de los modelos de diseño y despliegue. Responsable de la arquitectura de los modelos de diseño y despliegue

- Ingeniero de casos de uso: responsable de integridad de una o más realizaciones de casos de uso-diseño y debe garantizar que cumplan los requisitos que se esperan de ellos
- Ingeniero de componentes. define y mantiene las operaciones, métodos, atributos, relaciones y requisitos de implementación de una o más clases de diseño.

Flujo de trabajo

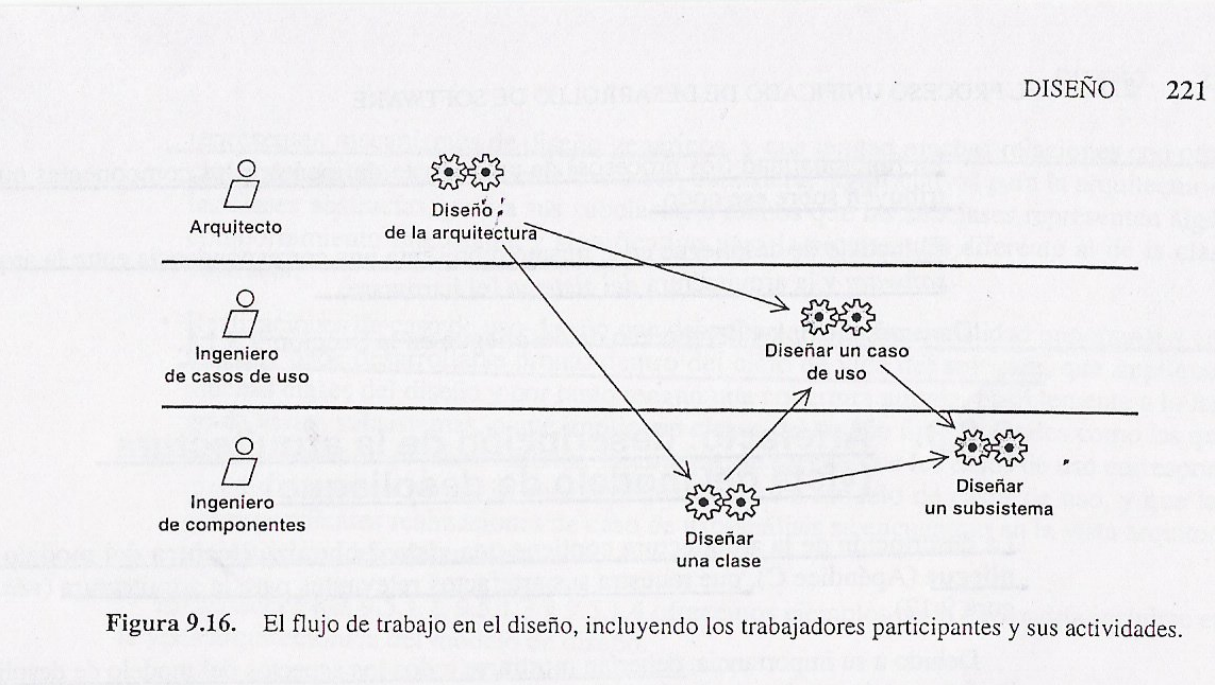


Figura 9.16. El flujo de trabajo en el diseño, incluyendo los trabajadores participantes y sus actividades.

Actividades

- Diseño de la arquitectura: esbozar los modelos de diseño y despliegue y su arquitectura mediante la identificación de nodos y sus configuraciones de red, sistemas y sus interfaces, clases de diseño significativas para la arquitectura . mecanismos de diseño genéricos que tratan requisitos comunes.
    - Identificación de nodos y configuraciones de red: aspectos a resaltar
        - qué nodos se necesitan, y cuál debe ser su capacidad en términos de potencia de proceso y tamaño de memoria
        - qué tipo de conexiones debe haber entre los nodos, y qué protocolos de comunicaciones se deben utilizar sobre ellos
        - qué características deben tener las conexiones y los protocolos de comunicaciones, en aspectos tales como ancho de banda, disponibilidad y calidad

        - es necesario tener alguna capacidad de proceso redundante?
    - Identificación de subsistemas y de sus interfaces:
        - identificación de subsistemas de aplicación: los subsistemas de las capas específica de la aplicación y general de la aplicación. una parte de un paquete del análisis se corresponde con un subsistema por sí misma. Algunas partes de un paquete del análisis se realizan mediante productos software reutilizados . Los paquetes del análisis no representan una división del trabajo adecuada. Los paquetes del análisis no representan la incorporación de un sistema heredado. Los paquetes del análisis no están preparados para una distribución directa sobre los nodos.
        - identificación de subsistemas intermedios y de software del sistema. toda la funcionalidad descansa sobresoftwaree como sistemas operativos, sistemas de gestión de base de datos, software de comunicaciones, tecnologías de distribución de objetos, kits de diseño de IGU y tecnologías de gestión de transacciones.El arquitecto verifica que los productos software elegidos encajan en la arquitectura y permiten una implementación económica del sistema.
        - definición de dependencias entre subsistemas. definirse dependencias entre subsistemas si sus contenidos tienen relaciones unos con otros
        - identificación de interfaces entre subsistemas. las interfaces proporcionadas por un subsistema definen operaciones que son accesibles “desde fuera” del subsistema
    - Identificación de clases de diseño relevantes para la arquitectura:
        - a partir de clases de análisis clases de análisis significativas para la arquitectura que encontramos en el análisis
        - identificación de clases activas. se esbozan mediante la consideración del ciclo de vida de sus objetos activos y de cómo deberían comunicarse, sincronizarse y compartir información los objetos activos
    - Identificación de mecanismos genéricos de diseño: estudiamos requisitos comunes. Deben tratarse aspectos de: persistencia, distribución transparente de objetos, características de seguridad, detección y recuperación de errores, gestión de transacciones.

- Diseño de un caso de uso:
  Objetivos:
    a. identificar las clases de diseño y/o los subsistemas cuyas instancias son necesarias para llevar a cabo el flujo de sucesos del caso de uso
    b. distribuir el comportamiento del caso de uso entre los objetos del diseño que interactúan y/o entre los subsistemas participantes

c. definir los requisitos sobre las operaciones de las clases del diseño y/o sobre los subsistemas y sus interfaces
d. capturar los requisitos de implementación del caso de uso

- identificación de clases del diseño participantes: identificar las clases de diseño que se necesitan para realizar el caso de uso. estudiar las clases del análisis que participan en la correspondiente realización de caso de uso-análisis. identificar las clases del diseño que poseen una traza hacia esas clases del análisis. estudiar los requisitos especiales de la correspondiente realización de caso de uso-análisis. identificar las clases del diseño que realizan esos requisitos especiales.identificar clases necesarias y asignar su responsabilidad a algún ingeniero de componentes
- Descripción de las interacciones entre objetos del diseño: mediante diagramas de secuencia que contienen las instancias de los actores, los objetos del diseño y las transmisiones de mensajes entre éstos
- identificación de subsistemas e interfaces participantes: definir caso de uso en términos de los subsistemas y/o interfaces que participan en él. identificar subsistemas necesarios para realizar caso de uso
- Descripción de interacciones entre subsistemas: describir cómo interactúan los objetos de las clases que contiene en el nivel de subsistema. mediante diagramas de secuencia que contengan las instancias de actores, subsistemas y transmisiones de mensajes entre éstos que participan
- Captura de requisitos de implementación: incluimos en la realización del caso de uso todos los requisitos identificados durante el diseño que deberían tratarse en la implementación

- Diseño de una clase: el propósito es crear una clase del diseño que cumpla su papel en las realizaciones de los casos de uso y los requisitos no funcionales que se aplican a todos
  - Esbozar las clases de diseño: diseñar clases de interfaz es dependiente de la tecnología de interfaz específica que se utilice. clases de entidad a menudo implica el uso de tecnología de bases de datos específicas
    - Diseño clases de control: considerar aspectos:
      1. aspectos de distribución: si la secuencia necesita ser distribuida y manejada por diferentes los nodos de una red, se puede requerir separar las clases del diseño en diferentes nodos
      2. aspectos de rendimiento
      3. aspectos de transacción
    - Identificar operaciones: identificamos operaciones que la clase de diseño va a necesitar y describimos estas operaciones utilizando la sintaxis de los lenguajes de programación

- Identificar atributos: identificamos atributos requeridos por la clase de diseño y los describimos utilizando la sintaxis del lenguaje y programación
- Identificar asociaciones y agregaciones: objetos de diseño interactúan unos con otros en los diagramas de secuencia. Estas interacciones requieren asociaciones entre las clases correspondientes. Número de relaciones entre clases debe estar minimizado
- Identificar generalizaciones: deben ser utilizadas con la misma semántica definida en el lenguaje de programación
- Describir los métodos métodos pueden ser utilizados durante el diseño para especificar cómo se deben realizar las operaciones. Puede especificarse en lenguaje natural o pseudocódigo
- x Describir estados: algunos objetos de diseño son estados controlados. Significa utilización de diagramas de estado para describir transiciones de estado de un objeto del diseño
- Tratar requisitos especiales

- Diseño de un subsistema:
  Objetivos:
  i. garantizar que el subsistema es tan independiente como sea posible de otros subsistemas y/o interfaces
  ii. garantizar que el subsistema proporciona las interfaces correctas
  iii. garantizar que el subsistema cumple su propósito de ofrecer una realización correcta de las operaciones tal y como se define en las interfaces
  - Mantenimiento de las dependencias entre subsistemas: cuando los elementos contenidos en estos últimos estén asociados como elementos dentro de aquel. Si estos subsistemas proporcionan interfaces, las dependencias de uso deberían declararse hacia las interfaces en vez de hacia los propios subsistemas
  - Mantenimiento de la interfaces proporcionadas por el subsistema : las operaciones definidas por las interfaces que proporciona un subsistema deben soportar todos los roles que cumple el subsistema en la diferentes realizaciones de caso de uso
  - Mantenimiento de los contenidos de los subsistemas: un subsistema cumple sus objetivos cuando ofrece una realización correcta de las operaciones tal y como están descritas por las interfaces que proporcionan:
    i. por cada interfaz que proporciona el subsistema, debería haber clases del diseño u otros subsistemas dentro del subsistema que proporcionen la interfaz

ii. para clarificar cómo el diseño interno de un subsistema realiza cualquiera de sus interfaces o casos de uso, podemos crear colaboraciones en términos de los elementos contenidos en el subsistema

10 - IMPLEMENTACIÓN:

Empezamos con el resultado de diseño e implementamos el sistema en términos de componentes.

Propósitos:

- Planificar las integraciones de sistema necesarias en cada iteración
- Distribuir el sistema asignando componentes ejecutables a nodos en el diagrama de despliegue.
- Implementar las clases y subsistemas encontrados durante el diseño
- Probar los componentes individualmente e integrarlos compilándolos y enlazando en uno o más ejecutables

Implementación es el centro durante las iteraciones de construcción, se lleva a cabo también durante transición para tratar defectos tardíos.

Artefactos:

- Modelo de implementación: describe cómo los elementos del diseño se implementan en términos de componentes. Se representa con un sistema de implementación que denota el subsistema de nivel superior del modelo
- Componente: empaquetamiento físico de los elementos de un modelo. Estereotipos: <<executable>>(programa que puede ser ejecutado en un nodo)<<file>>(fichero que contiene código fuente o datos)<<library>> librería estática o dinámica <<table>>tabla de bases de datos<<document>>(documento) normal que un componente implemente varios elementos
    - Stub: componente con una implementación esquelética o de propósito especial que puede ser utilizada para desarrollar o probar otro componente que depende del stub.
- Subsistema de implementación: proporciona una forma de organizar los artefactos del modelo de implementación en trazos más manejables. Un subsistema puede estar formado por componentes, interfaces y otros subsistemas.

Un subsistema de implementación se manifiesta a través de un mecanismo de empaquetamiento concreto en un entorno de implementación determinado.

- Interfaz: un componente que implementa una interfaz ha de implementar correctamente todas las operaciones definidas por la interfaz.

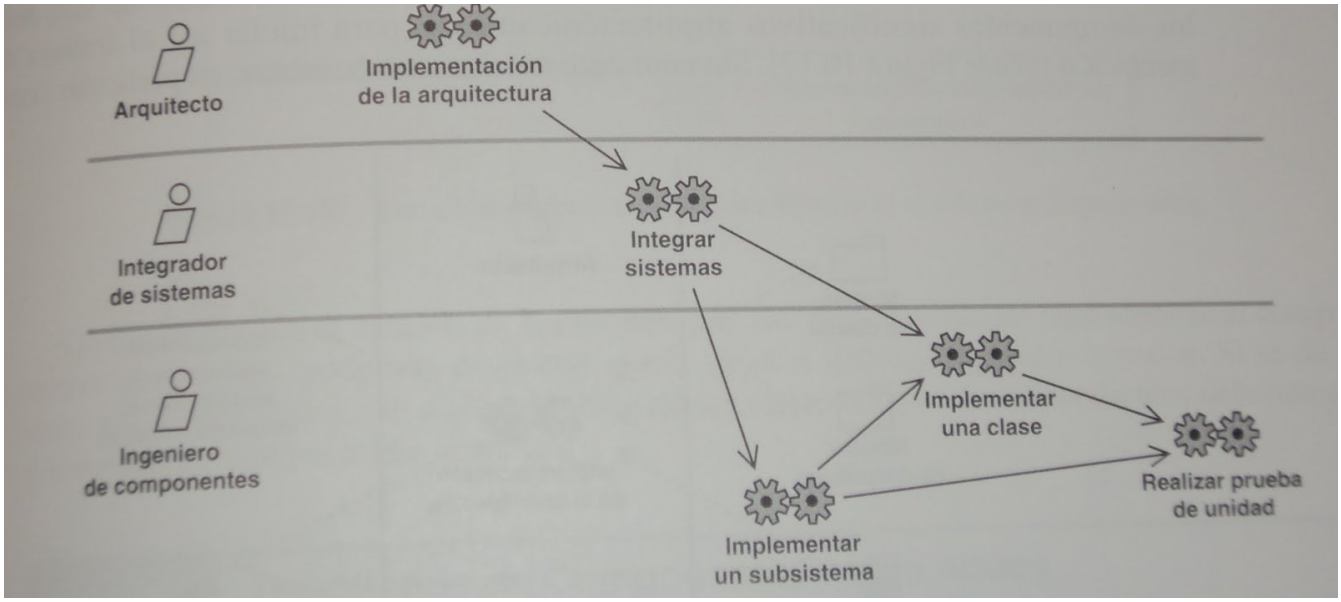
- Descripción de la arquitectura: contiene una visión de la arquitectura del modelo de implementación representa sus artefactos arquitectónicamente significativos. Artefactos considerados: descomposición del modelo de implementación en subsistemas, interfaces y dependencias entre ellas.
- Plan de integración de construcciones: construir software incrementalmente en pasos manejables de forma que cada paso de lugar a pequeños problemas de integración o prueba. Cada construcción es sometida a pruebas de integración antes de que se cree una construcción. Beneficios:
    i. Se puede crear una versión ejecutable del sistema y pronto
    ii. Más fácil localizar defectos durante las pruebas de integración porque añade una parte pequeña y manejable
    iii. Prueba de integración más minuciosa

  Plan de integración de construcciones describe la secuencia de construcciones necesarios en una iteración. Describe: funcionalidad que se espera que sea implementada en dicha construcción,partes del modelo de implementación que están afectadas por la construcción.

Trabajadores:

- Arquitecto: responsable de la integridad del modelo de implementación y asegura que el modelo como un todo es correcto, completo y legible. El modelo es correcto cuando implementa la funcionalidad descrita en el modelo de diseño y en los requisitos adicionales y sólo esa funcionalidad. Responsable de arquitectura del modelo de implementación.
- Ingeniero de componentes: define y mantiene el código fuente de uno o varios componentes, garantizando que cumplen la funcionalidad correcta. También mantiene la integridad de subsistemas de implementación. Debe diseñar e implementar las clases bajo su responsabilidad.
- Integrador de sistemas: responsabilidad planificar la secuencia de construcciones necesarias en cada iteración e integración de cada construcción cuando sus partes han sido implementadas.

Flujo de trabajo: Objetivo principal es implementar el sistema



Actividades:

- Implementación de la arquitectura: propósito esbozar modelo de implementación y arquitectura mediante identificación de componentes significativos arquitectónicamente y asignación a los nodos de componentes
    - Identificación de componentes ejecutables y asignación de éstos a nodos: Consideramos clases activas y asignamos un componente ejecutable por cada clase activa. Representar en una vista de la arquitectura del modelo de despliegue.
- Integrar el sistema: objetivo es crear un plan de integración de construcciones que describa construcciones en una iteración y requisitos de cada construcción. Integrar cada construcción antes de que sea sometida a pruebas de integración.
    - Planificación de una construcción: Criterios:
        - una construcción debería añadir funcionalidad a la construcción previa implementando casos de uso completos o escenarios de éstos.
        - una construcción no debería incluir demasiados componentes nuevos o refinados
        - Construcción debería estar basada en la anterior y debería expandirse hacia arriba y a los lados en la jerarquía de subsistemas.

        - considerar el diseño del caso de uso identificando su realización de caso de uso-diseño correspondiente. Identificar subsistemas y clases de diseño que participan en la realización de caso de uso-diseño. Identificar subsistemas y componentes de implementación en el modelo de implementación. Considerar impacto de implementar requisitos de éstos subsistemas de implementación y de los componentes sobre la construcción en cuestión.
    - Integración de construcción: recopilando las versiones correctas de los subsistemas de implementación y de los componentes, comparándolas y enlazandonos para generar una construcción. Construcción se somete a pruebas de integración y a pruebas de sistema (si es la última dentro de la iteración).

- Implementar un subsistema: asegurar que un subsistema cumple su papel en cada construcción.
    - Mantenimiento de los contenidos de los subsistemas: puede que contenido de un subsistema necesite ser refinado por Ingeniero de componentes. Puede incluir:
        - Cada clase necesaria en la construcción actual debería ser implementada mediante componentes en el subsistema de implementación.
        - Cada interfaz proporcionada por subsistema de diseño requerido en construcción debería ser proporcionada por el subsistema de implementación

- Implementar una clase: implementar una clase de diseño en un componente fichero. Incluye esbozo de un componente fichero que contendrá el código fuente, generación de código fuente a partir de la clase de diseño y relaciones, implementación de operaciones en forma de métodos y comprobación de que el componente proporciona las mismas interfaces que la clase de diseño
    - Esbozo de los componentes de fichero: tipo de modularización y convenciones de los lenguajes de programación en uso restringirán la forma en que los componentes fichero son esbozados.
    - Generación de código a partir de una clase de diseño: forma en que esto se haga depende del lenguaje de programación
    - Implementación de operaciones: incluye la elección de un algoritmo y unas estructuras de datos apropiadas y la codificación de las acciones requeridas por el algoritmo.
    - Componentes: han de proporcionar interfaces adecuadas.
- Realizar pruebas de unidad: probar los componentes implementados como unidades individuales.
  Tipos de pruebas:
        - Pruebas de especificación para verificar comportamiento del componente

sin tener en cuenta cómo se implementa dicho comportamiento en el componente. El número de entradas, salidas y estados se divide en clases de equivalencia (conjunto de valores de entrada, salida o estado para los que se supone que un objeto se comporta de forma similar)

- Realización de pruebas de estructura: para verificar que un componente funciona internamente como se quería debe asegurarse probar los caminos más importantes en el código y probar todo el código.

11 - PRUEBA

Verificamos el resultado de la implementación probando cada construcción, incluyendo construcciones internas e intermedias, y versiones finales del sistema a ser entregadas.

Objetivos:

- Planificar las pruebas necesarias en cada iteración, incluyendo pruebas de integración y pruebas de sistema.
- Diseñar e implementar las pruebas creando los casos de pruebas que especifican qué probar, creando los procedimientos de prueba que indican cómo realizar las pruebas y creando componentes de pruebas ejecutables para automatizar las pruebas.
- Realizar las diferentes pruebas y manejar los resultados de cada prueba sistemáticamente. Las construcciones en las que se detectan defectos son probadas de nuevo y posiblemente devueltas a otro flujo de trabajo, de forma que los defectos importantes puedan ser arreglados.

Fase de inicio puede hacerse parte de la planificación inicial de las partes cuando se define el ámbito del sistema. Pruebas se llevan a cabo cuando una construcción es sometida a pruebas de integración y de sistema. Se centra en fases de elaboración cuando se prueba línea base ejecutable de arquitectura, y de construcción cuando el grueso del sistema está implementado. Durante transición corrección de defectos donde los primeros usos y a las pruebas de regresión.
Modelo de prueba cambia debido a eliminación de casos de prueba obsoletos, refinamiento de casos de prueba en casos de prueba de regresión, creación de nuevos casos de uso para cada nueva construcción.

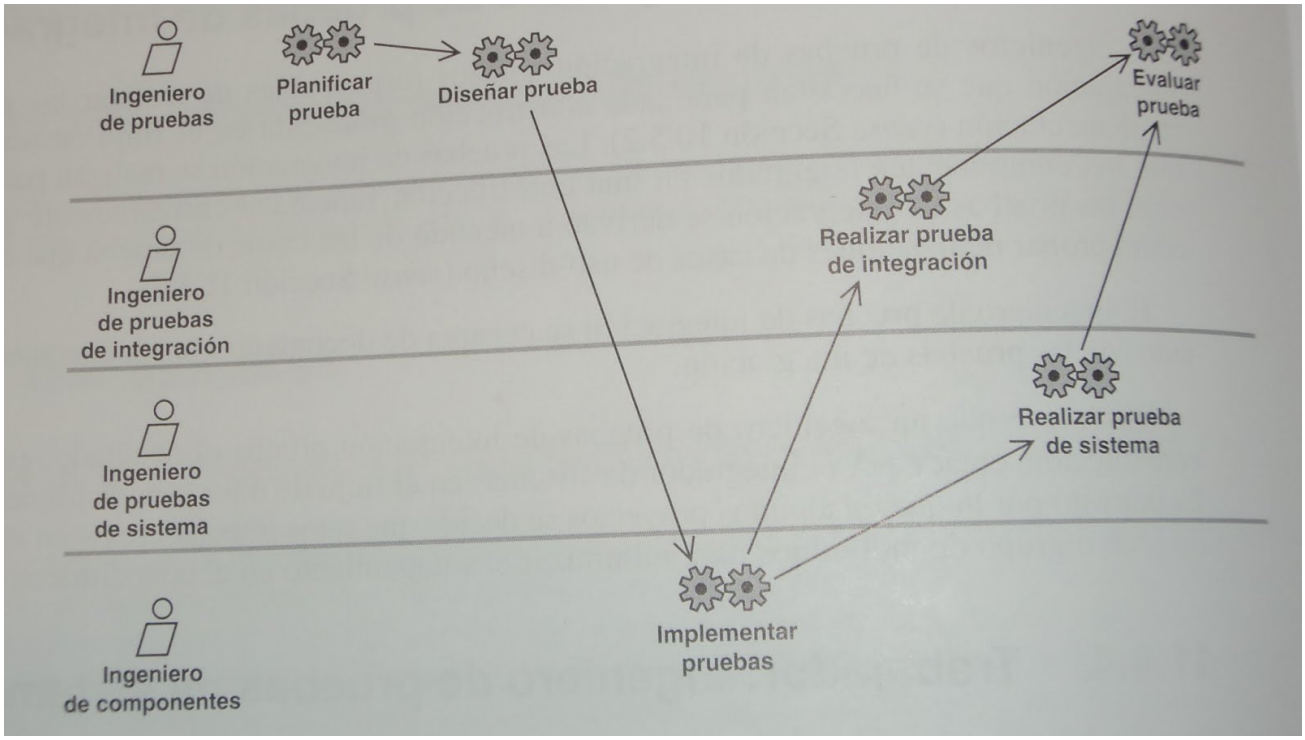
Artefactos:

- Modelo de pruebas: describe cómo se prueban los componentes ejecutables en el modelo de implementación con pruebas de integración y de sistema. Puede describir cómo han de ser probados aspectos específicos del sistema.
- Casos de pruebas: específica una forma de probar el sistema, incluyendo la entrada o resultado con la que se han de probar y las condiciones bajo las que ha de probarse
    - caso de prueba que especifica cómo probar un caso de uso o un escenario específico de un caso de uso. Especifica típicamente una prueba del sistema como caja negra.
    - Caso de prueba que especifica como probar una realización de casos de uso-diseño o un escenario específico de la realizaci{on. Especifica una prueba del sistema como caja blanca.
    - Otros casos de prueba: - prueba de instalación: verifica que el sistema pueda ser instalado -prueba de configuración: verifica que el sistema funcione correctamente en diferentes configuraciones. - pruebas negativas: intentan provocar que el sistema falle - pruebas de tensión o estrés: identifica problemas con el sistema cuando hay recursos insuficientes o cuando hay competencia por recursos.
- Procedimientos de prueba: especifica cómo realizar uno o varios casos de prueba parten de éstos.
- Componentes de pruebas: automatiza un9o o varios procedimientos de prueba o partes de ellos. Pueden ser desarrollados utilizando un lenguaje de guiones o de programación o grabados como una herramienta de automatización de pruebas. Se utilizan para probar componentes en el modelo de implementación.
- Plan de prueba: describe estrategias, recursos y planificación de las pruebas. Definición tipo de pruebas y objetivos, nivel de cobertura, porcentaje de pruebas que deben realizarse con un resultado específico.
- Defecto: anomalía del sistema. Utilizado para localizar cosas a registrar como síntoma de un problema en el sistema que necesitan controlar y resolver.
- Evaluación de prueba: evaluación de los resultados de los esfuerzos de prueba.

Trabajadores:

- Diseñador de pruebas: responsable de integridad de modelo de pruebas. Planea las pruebas. Selecciona y describe casos de prueba y los procedimientos de prueba correspondiente que se necesitan. REsponsable de evaluación de las pruebas de integración.
- Ingeniero de componentes: responsable de componentes de prueba que automatizan algunos de los procedimientos de prueba.
- Ingeniero de pruebas de integración: responsable de realizar las pruebas de integración que se necesitan para cada construcciń producida en el flujo de trabajo de la implementación. Documenta defectos en los resultados de las pruebas de integración.
- Ingeniero de pruebas del sistema: responsable de realizar las pruebas de sistema necesarias sobre una construcción que muestra el resultado de una iteración completa. Documenta defectos en los resultados de las pruebas de sistema.

Flujo de Trabajo:



Actividad:

- Planificar pruebas: planificar los esfuerzos de prueba en una iteración llevando a cabo:
    - descripción de estrategia de prueba
    - estimar los requisitos para el esfuerzo de la prueba
    - planificar el esfuerzo de la prueba

- Diseñar prueba: propósito es identificar y describir los casos de prueba para cada construcción, identificar y estructurar los procedimientos de prueba especificando cómo realizar los casos de prueba.
    - Diseño de los casos de prueba de integración: la mayoría de los casos de prueba de integración pueden ser derivados de las realizaciones de casos de uso-diseño, ya que las realizaciones de casos de uso describen cómo interaccionan las clases y los objetos y cómo interaccionan los componentes. Diseñadores de prueba intentan encontrar un conjunto de casos de prueba con un solapamiento mínimo, cada uno de los cuales prueba un camino o escenario interesante a través de una realización de casos de uso.
    - Diseño de los casos de prueba del sistema: prueba combinaciones de casos de uso instanciados bajo condiciones diferentes (diferente configuración de hardware, niveles de carga, número de actores). Dar prioridad a las combinaciones de casos de uso que: se necesita que funcionen en paralelo, posiblemente funcionen en paralelo, posiblemente se influencien mutuamente si se ejecutan en paralelo, involucran varios procesadores, usan frecuentemente recursos del sistema.
    - Diseño de los casos de prueba de regresión: para ser adecuados los casos de prueba han de ser suficientemente flexibles para ser resistentes a cambios en el software que está siendo probado.
    - Identificación y estructuración de los procedimientos de prueba: cada caso de prueba precisará varios procedimientos de prueba, quizá uno por cada subsistema de servicios probado en el caso de pruebas.
- Implementar prueba: automatizar los procedimientos de prueba creando componentes de prueba si esto es posible. Los componentes de prueba se crean usando los procedimientos de prueba como entrada:
    - cuando usamos una herramienta de automatización de pruebas realizamos o especificamos las acciones como se describen en los procedimientos de prueba.
    - cuando programamos los componentes de prueba explícitamente usamos los procedimientos de prueba como especificaciones iniciales.
- Realizar pruebas de integración: se realizan las pruebas de integración necesarias por cada una de las construcciones creadas en una iteración y se recopilan los resultados de las pruebas. Pasos:
    - realizar las pruebas de integración relevantes a la construcción realizando los procedimientos de prueba manualmente por cada caso de prueba o ejecutando cualquier componente de prueba que automatice los procedimientos de prueba.
    - comparar resultados de las pruebas con los resultados esperados e investigar los resultados de las pruebas que no coinciden con los esperados.
    - -informar de los defectos a los ingenieros de componentes responsables de los componentes que se cree contienen fallos.

    - informar de los defectos a los desarrolladores de pruebas, quienes utilizarán los defectos para evaluar los resultados del esfuerzo de prueba.
- Realizar prueba de sistema: propósito es el realizar las pruebas de sistema necesarias en cada iteración y el recopilar los resultados de las pruebas. Puede empezar cuando las pruebas de integración indican que el sistema satisface los objetivos de calidad de integración fijados en el plan de pruebas de la iteración actual. Se realiza de forma análoga a la forma en que se realiza la prueba de integración.
- Evaluar prueba: evaluar los esfuerzos de prueba en una iteración. Evaluar resultado comparando los resultados obtenidos con objetivos esbozados en el plan de prueba. Diseñadores de pruebas preparan métricas que les permiten determinar el nivel de calidad del software y qué cantidad de pruebas es necesario hacer. Ingeniero de pruebas observa dos métricas:
    - Compleción de la prueba: obtenida a partir de la cobertura de los casos de prueba y de la cobertura de los componentes probados. Indica porcentaje de casos de prueba que han sido ejecutados y porcentaje de código probado.
    - Fiabilidad: se basa en el análisis de las tendencias en los defectos detectados y en las tendencias en las pruebas que se ejecutan con el resultado esperado. PAra determinar la fiabilidad del sistema, diseñadores de pruebas crean diagramas de las tendencias de los defectos , representa distribución de tipos específicos de defectos a lo largo del tiempo. Diagramas de tendencias representan porcentaje de pruebas ejecutadas con éxito a lo largo del tiempo. Tendencias de los defectos siguen a menudo patrones que se repiten en varios proyectos.

El número de defectos se incrementa bastante cuando comienza la prueba, se mantiene estable y empieza a caer lentamente.