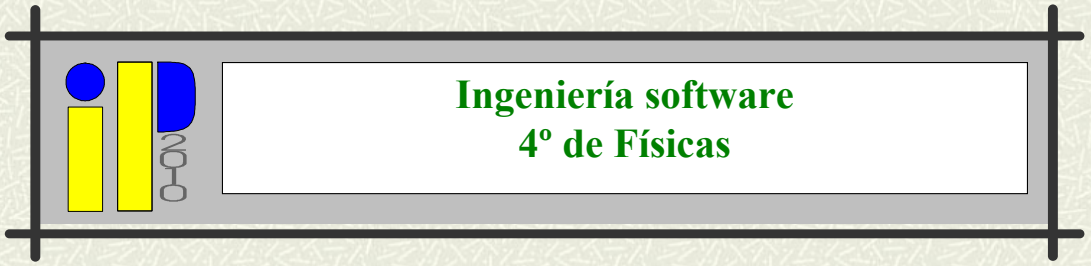


Proceso de desarrollo de aplicaciones software

Julio Medina & J.M. Drake
Computadores y Tiempo Real

Santander, 2010

Proceso de desarrollo de aplicaciones software.

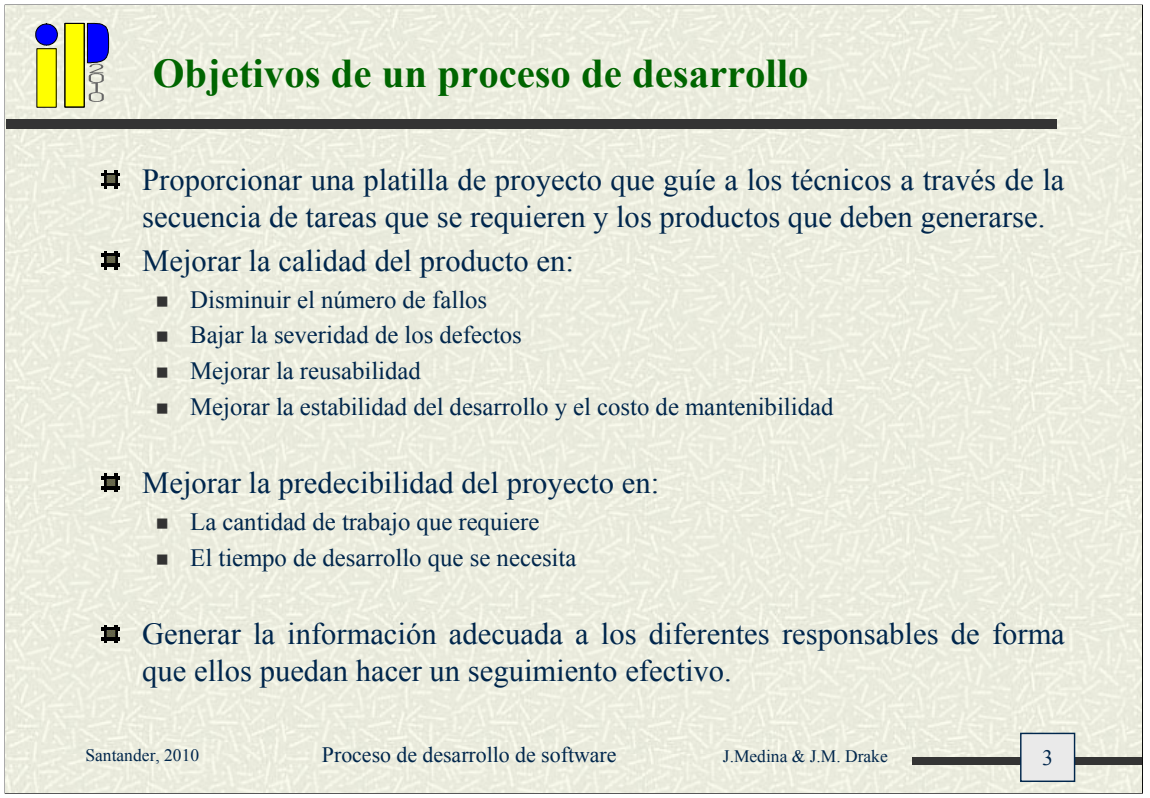
- Un proceso de desarrollo de software es la descripción de una secuencia de actividades que deben ser seguida por un equipo de trabajadores para generar un conjunto coherente de productos, uno de los cuales en el programa del sistema deseado.

- El objetivo básico del proceso es hacer predecible el trabajo que se requiere:
  - Predecir el costo.
  - Mantener un nivel de calidad
  - Predecir el tiempo de desarrollo

Santander, 2010 Proceso de desarrollo de software J.Medina & J.M. Drake 1

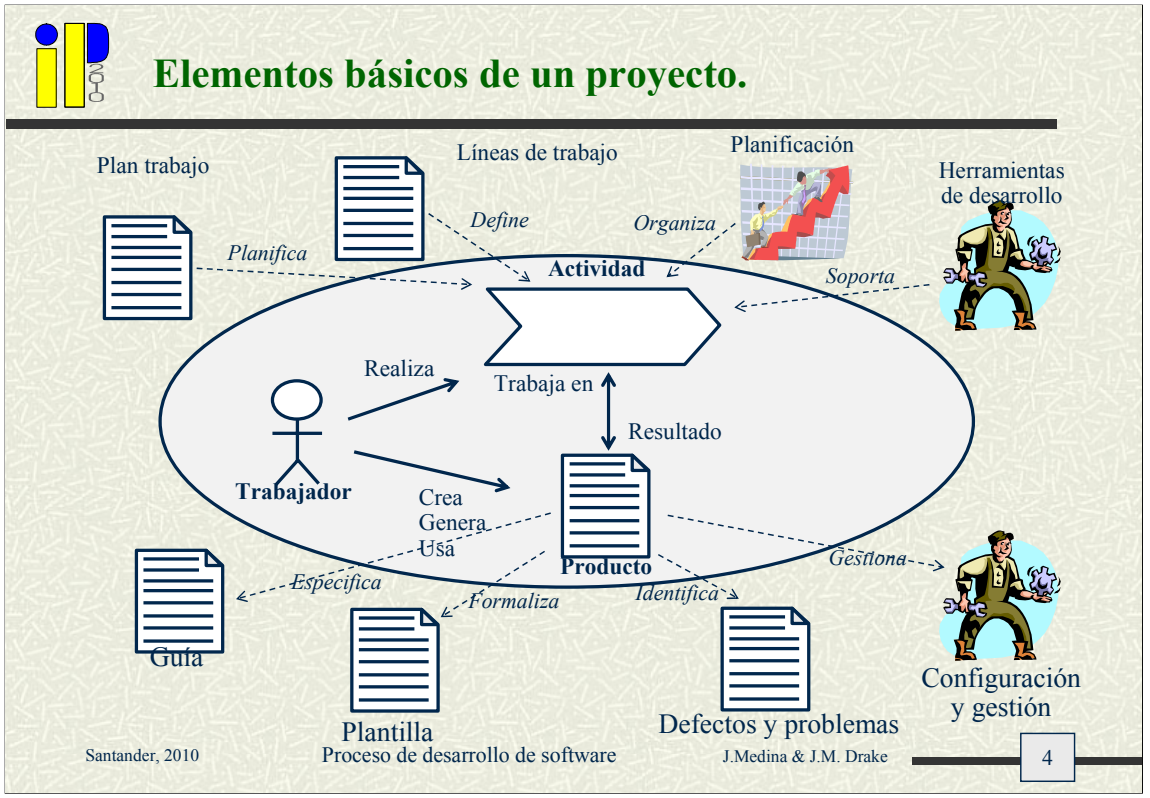
El objetivo de un proceso de desarrollo de programas es la formalización de las actividades relacionadas con el desarrollo del software de un sistema informático.
La mayoría de los proyectos que se desarrollan, finalizan tarde, cuesta mucho mas de lo estimado. ¿Por qué ocurre esto?. El software se encuadra entre los artefactos mas complejos que es capaz de desarrollar el hombre, y además dado que no tiene límites físicos por su carácter inmaterial, su dimensión se puede imaginar ilimitada.

La adopción por una empresa de un proceso de desarrollo contrastado, le permite producir aplicaciones software con plazos y costos predecibles y con calidad constante.
En esta sección se estudia un marco de desarrollo basados en criterios genéricos, y que cada empresa debe configurar y refinar de acuerdo con las características de la empresa y del producto.
No existe un proceso único aplicable al desarrollo de cualquier tipo de aplicación, adoptable por cualquier empresa y valido para cualquier cultura productiva

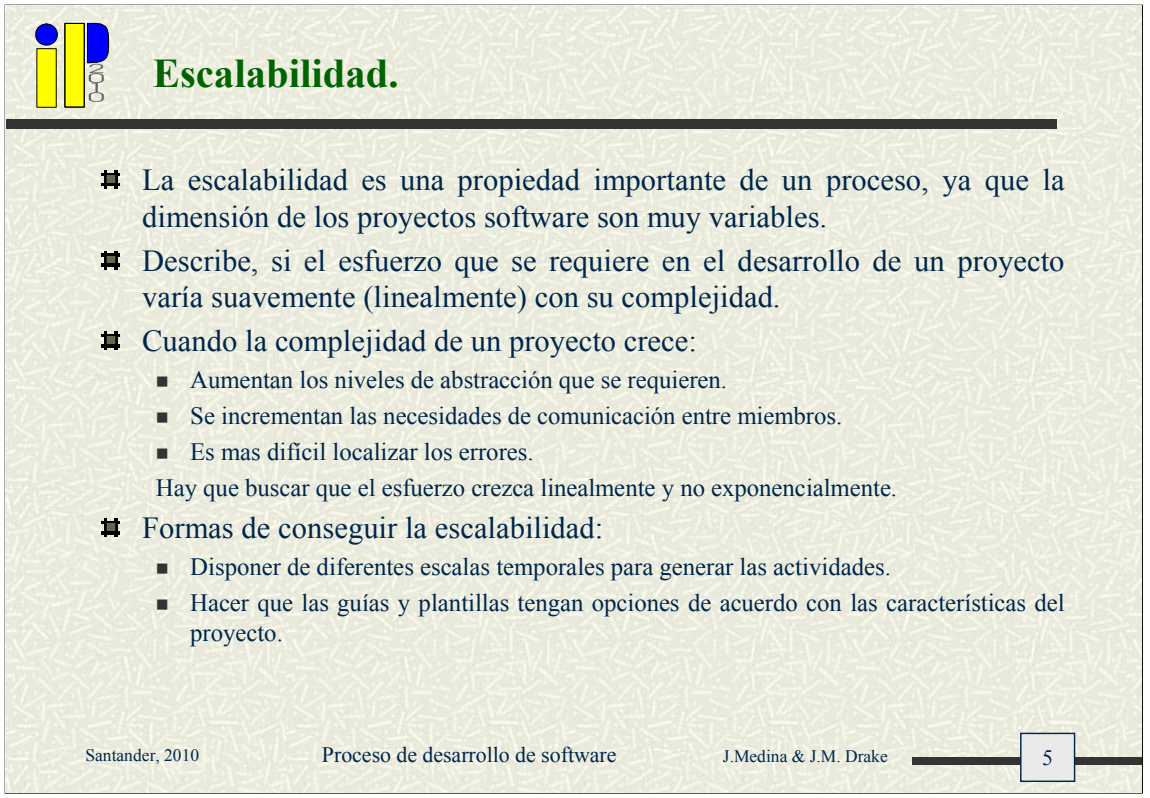


La razón básica por la que se requiere disponer de un proceso de desarrollo es mejorar la seguridad de trabajo eliminando riesgos innecesarios y conseguir un producto de la máxima calidad.
Específicamente un proceso de desarrollo debe conseguir:

- Proporcionar una plantilla de desarrollo del proyecto en el que quede definido lo que cada trabajador que interviene debe realizar y los productos que debe generar a lo largo de él.
- Mejorar la calidad del producto que se genera en función de:
  - Disminuir el número de defectos que se producen y que deben ser corregidos.
  - Disminuir la severidad de los defectos residuales que al final pueden permanecer en el productos final.
  - Mejorar la reusabilidad, de forma que gran parte del trabajo que se realiza pueda ser reutilizado en próximos proyectos.
  - Mejorar la estabilidad del proceso de forma que se minimicen las reelaboraciones del producto.
  - Generar un producto que sea de fácil mantenimiento posterior.
- Mejorar la predecibilidad del proyecto en función de:
  - La cantidad de esfuerzo humano y de recursos que requiera.
  - Disminuir los plazos de desarrollo y llegada al mercado.
- Generar a lo largo del desarrollo de la información adecuada y diferenciada para que los diferentes responsable del proyecto puedan hacer su seguimiento de forma efectiva.



Los elementos básicos a definir en un proceso de desarrollo de software son los papeles que juegan los trabajadores, las actividades que desarrollan y los productos que deben generarse.
En un plan de desarrollo cada trabajador debe tener su papel dentro de él, lo que define las actividades que debe realizar y los productos que debe generar.
Las actividades son las tareas que deben realizar los trabajadores para cumplir sus obligaciones. A alto nivel, estas actividades son concebidas como las fases del proceso (especificación, análisis, ..), mientras que a mas bajo nivel son tareas mas concretas (crear cierto diagramas, escribir código,..).
Los productos son los documentos o información que debe ser creada como consecuencia de la actividad que se desarrolla. El producto último es el sistema que se desarrolla, pero en las fases intermedias deben generarse una amplia gama de documentos intermedios. Cada actividad debe tener siempre como principal objetivo generar ciertos productos bien definidos y especificados.
Los procesos deben estar condicionados por el tipo de producto que se desarrolla y por la tradición y experiencia de la empresa que lo desarrolla.



Una de las propiedades que deben ser exigidas a un proceso de desarrollo de aplicaciones software es la escalabilidad, lo que hace posible que sea aplicable tanto a sistemas complejos como a sistemas sencillos.

En general la propiedad de escalabilidad representa que si para desarrollar un proyecto de complejidad (y) es necesario realizar un esfuerzo (x), para desarrollar un proyecto de complejidad (100y) se requiere un esfuerzo (100cx) (donde c es una constante).

Cuando un proyecto crece, se produce que:

•Sus objetivos se hacen menos concretas y mas globales.

•El sistema gana en niveles de abstracción (tales como subsistemas, subsistemas, y así).

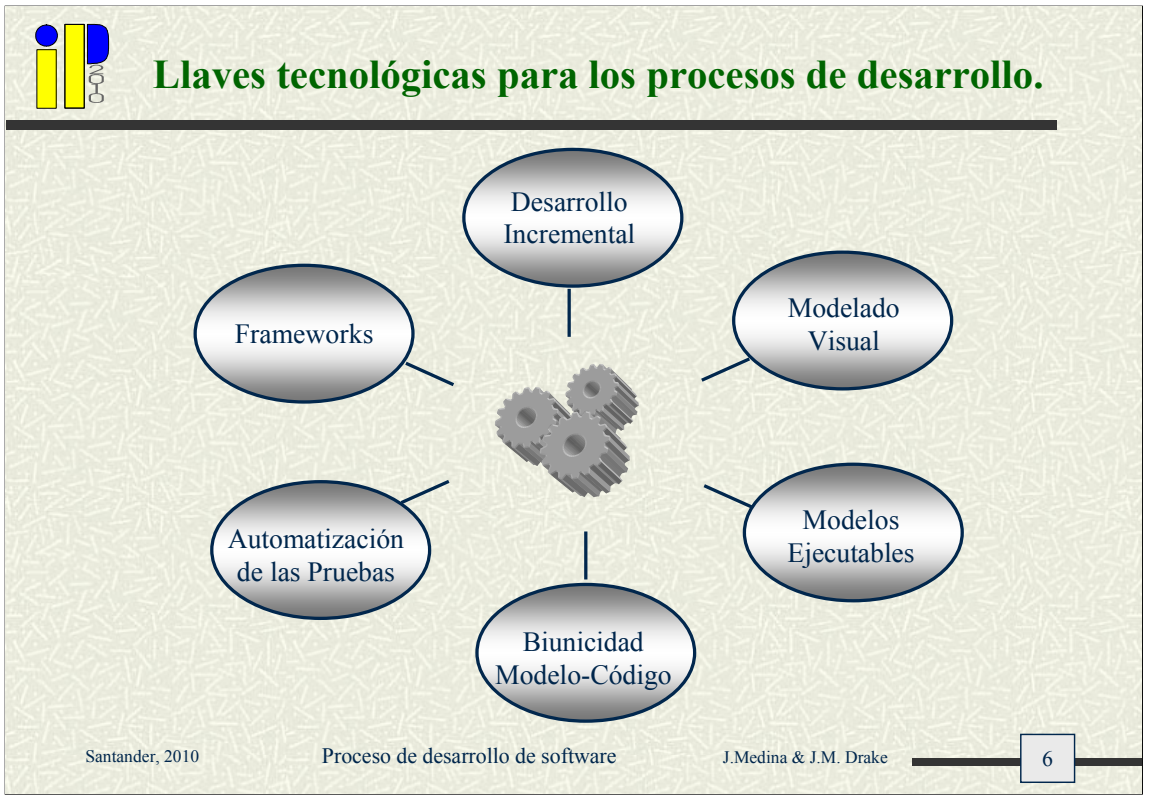
•Líneas potenciales de comunicación crece exponencialmente con el número de miembros del proyecto.

•El costo de los errores que hay que corregir se incrementa ya que aumentan las posibilidades de interferencias.

En general, existen dos soluciones para conseguir la escalabilidad de un proceso:

•El proceso es visto desde diferentes escalas de tiempo: macro, micro y nano escala, y en función de que el proyecto crezca mas relevancia adquieren la escala macro a fin de organizar y gestionar el proceso de desarrollo mas global.

•Muchas de las fase y mecanismos del proceso se hacen opcionales en función de que la complejidad de la aplicación se requiera.



Existen un conjunto de tecnologías y criterios que facilitan los procesos de desarrollo:

•Modelado Visual: Facilita la capacidad de apreciar los diferentes elementos e interacciones del sistema en los diferentes niveles de abstracción. Si se contase sólo con el código sería muy difícil fijarse en los diferentes níveles de abstracción.

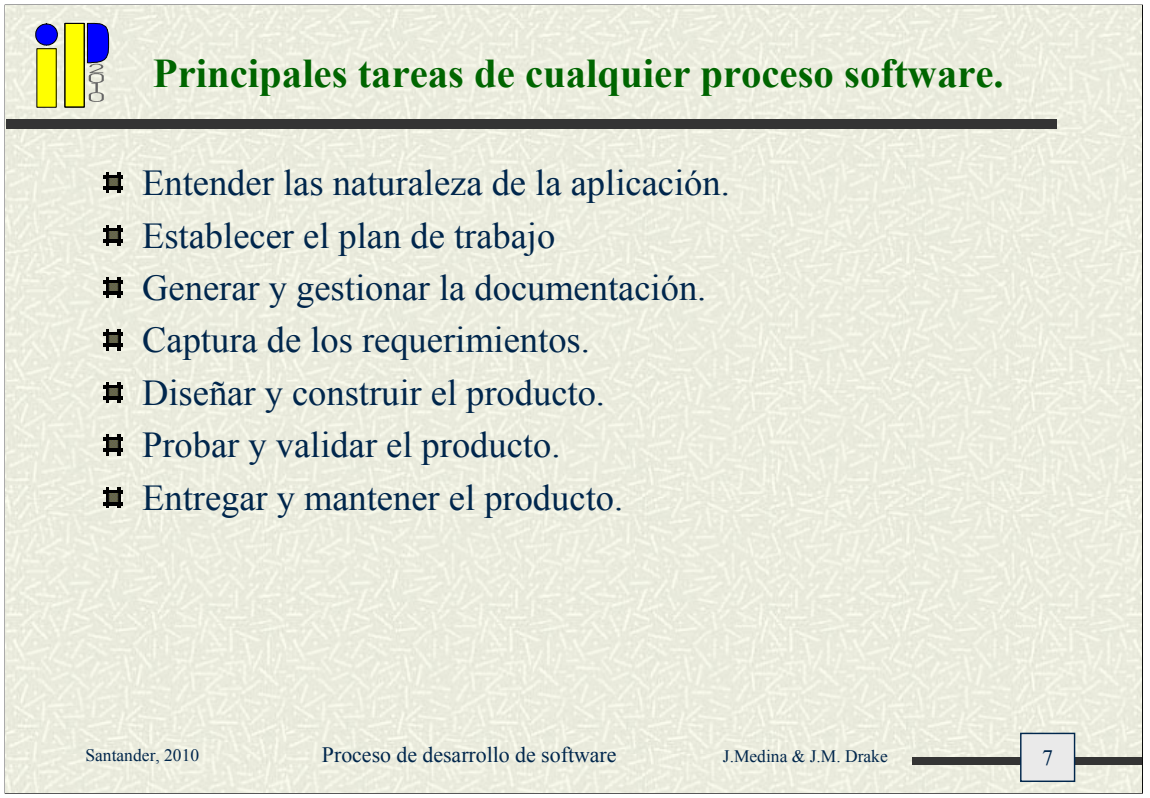
•Modelos Ejecutables: La mejor gestión de los errores que inevitablemente se introducen a lo largo del proceso de desarrollo consiste en detectarlos y corregirlos tan pronto como se cometan. Para que esto se pueda realizar eficientemente conviene tener capacidad de realizar las pruebas directamente desde los modelos, bien mediante debbuger a nivel de modelo, o bien, mediante la generación automática de prototipos que los hagan ejecutables.

•Relación biunívoca entre modelos y códigos: El proceso de desarrollo de las aplicaciones se basa en modelos que se desarrollan para que los diseños sean comprensibles y gestionables, y en código que es el producto final y es necesario que ambos se mantengan en todas las fases sintonizados. Esto se consigue si se automatiza el tránsito entre ambos. Por supuesto, para alcanzar este objetivo es necesario el soporte de herramientas de modelado.

•Automatización de las pruebas a partir de las especificaciones: El número de pruebas que hay que realizar para detectar errores se incrementa de forma acumulativa, como consecuencia de que no solo hay que verificar los nuevos elementos sino su interferencia con lo ya probado.

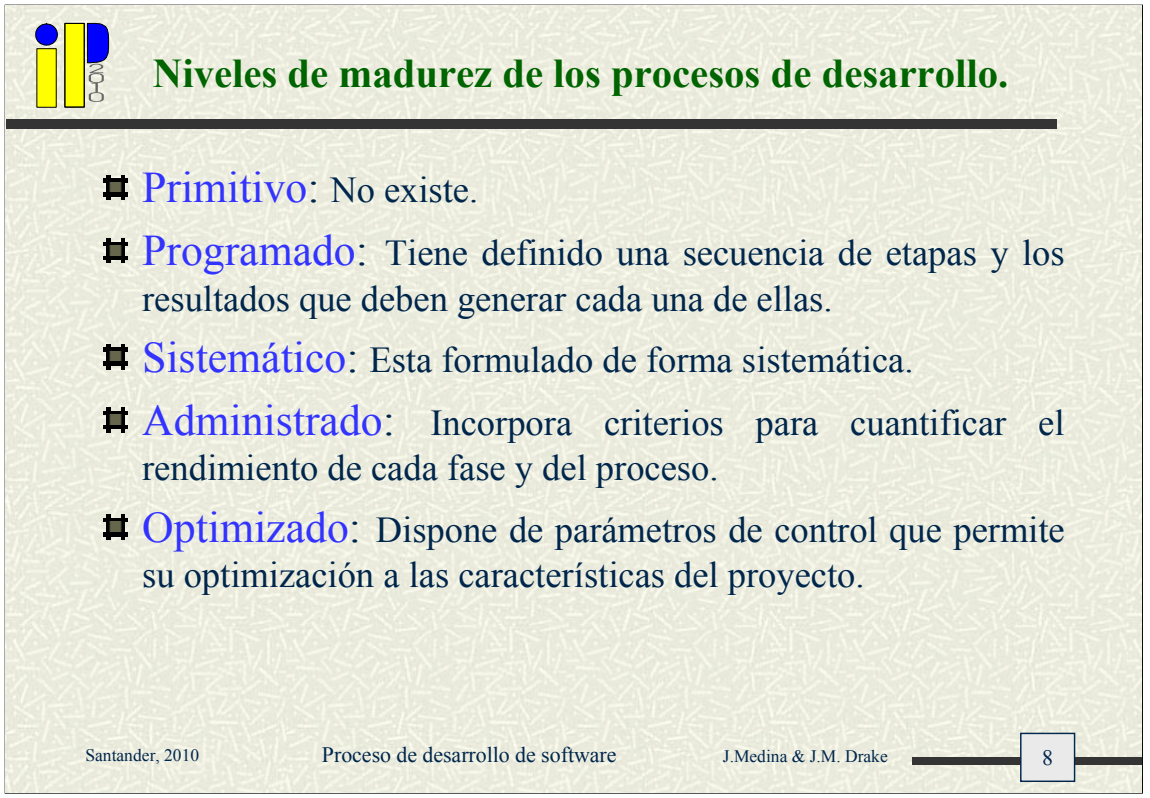
•Frameworks: Los frameworks son aplicaciones parcialmente desarrolladas que se utilizan como plantillas para el desarrollo de nuevas aplicaciones.

•Desarrollo incremental e iterativo: Los sistemas de desarrollo deben basarse en la generación iterativas de prototipos utilizables que vayan aumentando gradualmente su funcionalidad en las sucesivas etapas hasta conseguir que sea plena.



La siguiente es una secuencia común de las actividades para un proyecto software:

1. Es necesario comprender la naturaleza del proyecto. Esto parece obvio, pero casi siempre lleva tiempo entender lo que desean los clientes, en especial cuando ellos mismos no saben por completo que quieren.
2. Los proyectos requieren documentación desde el principio; es muy probable que esta documentación sufra muchos cambios. Por esta razón, desde el principio debe disponerse de una estrategia para mantener los documentos que se generen. Este proceso es denominado “Gestión de la Configuración”.
3. Hay que reunir los requisitos que ha de cumplir la aplicación. Gran parte de esta actividad es conversar con los “interesados”.
4. Hay que analizar el problema, diseñar la solución y codificar los programas.
5. El producto inicial y final debe probarse en forma exhaustiva en todos sus aspectos.
6. Una vez entregado el producto, entra el modo “mantenimiento, que incluye reparaciones y mejoras. Es una actividad que consume muchos recursos, a veces hasta el 65% de los recursos utilizados en el desarrollo de la aplicación. Ello hace hace que ha sea considerado un objetivo fundamental.



Se ha propuesto un procedimiento de evaluación de los procesos de desarrollo, que definen cinco niveles de madurez:

•Primitivo: El proceso no está formalizado y el equipo resuelve los problemas sobre la marcha. Sólo reconoce que el equipo es capaz de producir productos software. El éxito de proceso depende en gran medida de la experiencia de las personas que lo desarrollan. Cuando termina un proyecto nada se registra de su costo, tiempo ni calidad.

•Programado: El proceso es capaz de definir plazos razonables y verificar el avance del proyecto respecto de los plazos establecidos. Mantiene registros de los costos y tiempos del proyecto- En 1999 se comprobó que solo un 20% de los proyectos alcanzaban este nivel.

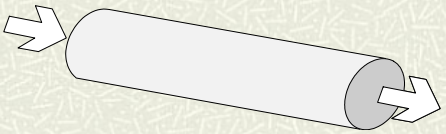
•Sistemático: El proceso está sistemáticamente definido y se reduce la dependencia de los individuos concretos que la realizan. Es conocido y comprendido por todas las partes que intervienen en el proyecto. Con este nivel se consigue predecir los resultados de proyectos similares a los que previamente desarrollado.

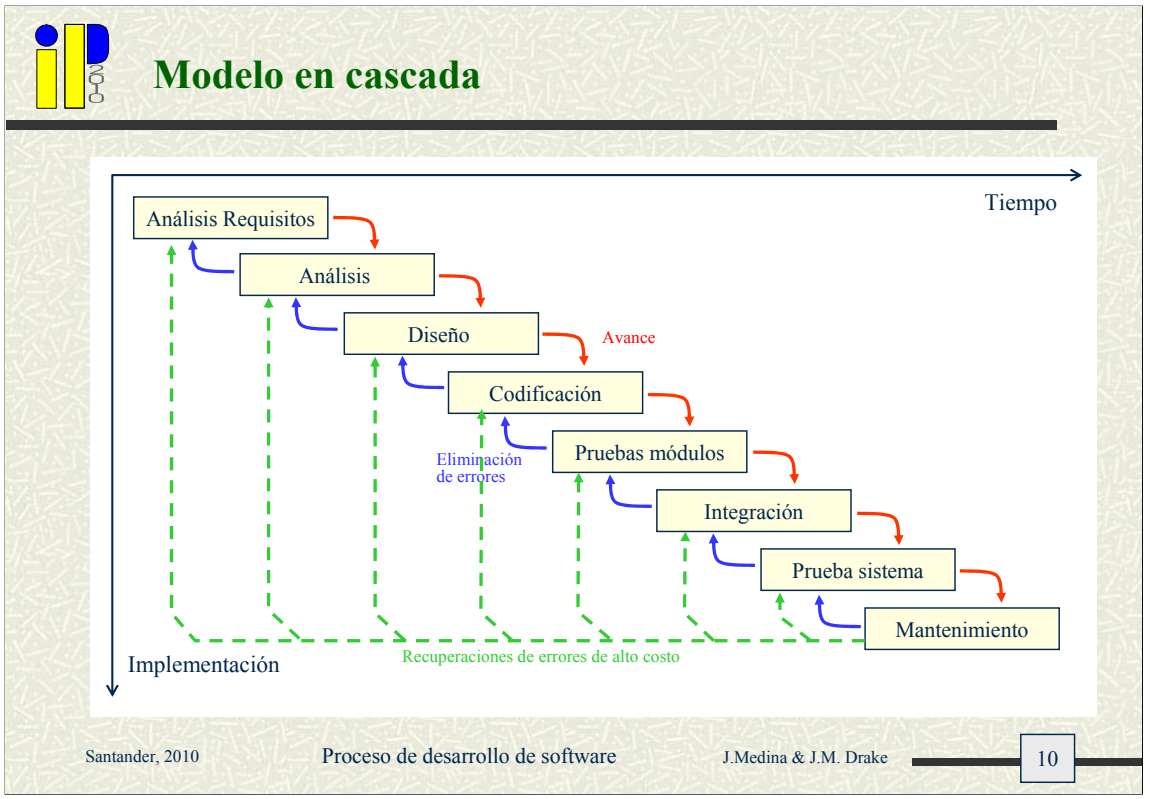
•Administrado: Puede predecir los costos y la programación de las tareas. El rendimiento del proceso es medible objetivamente y cuantitativamente.

•Optimizado: La ingeniería software está en continua evolución y es necesario disponer de estrategias adaptativas que permitan la adopción de las nuevas tecnologías. Su proceso es un proceso de metas e incluye manaras sistemáticas de evaluar el proceso mismo de la organización.

Modelo de procesos lineales.

- Modelo Túnel:
  - Ausencia de modelo
  - No hay ningún control
  - Sólo válido en proyectos muy pequeños.

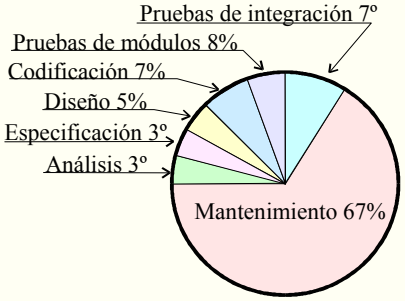




El modelo clásico del proceso de desarrollo de software es el modelo en cascada. Consiste en una secuencia de actividades: análisis de requisitos, diseño, codificación, integración y pruebas. No se comienza una fase hasta que no se ha terminado la anterior.

Problemas del proceso en cascada.

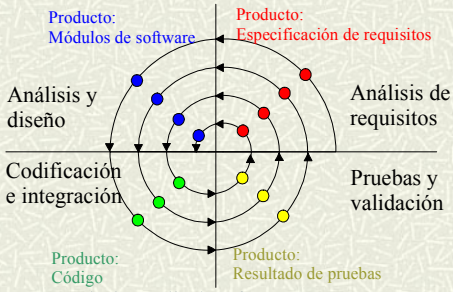
- La prueba efectiva solo se hace cuando ya está todo diseñado.
- Presupone que antes de iniciar el proceso el producto que se genera está perfectamente definido.
- Cuando se descubren problemas en la fase de mantenimiento hay que adaptar el problema a la aplicación, y no al revés.



Costo de las fases del desarrollo de una aplicación

Proceso en espiral

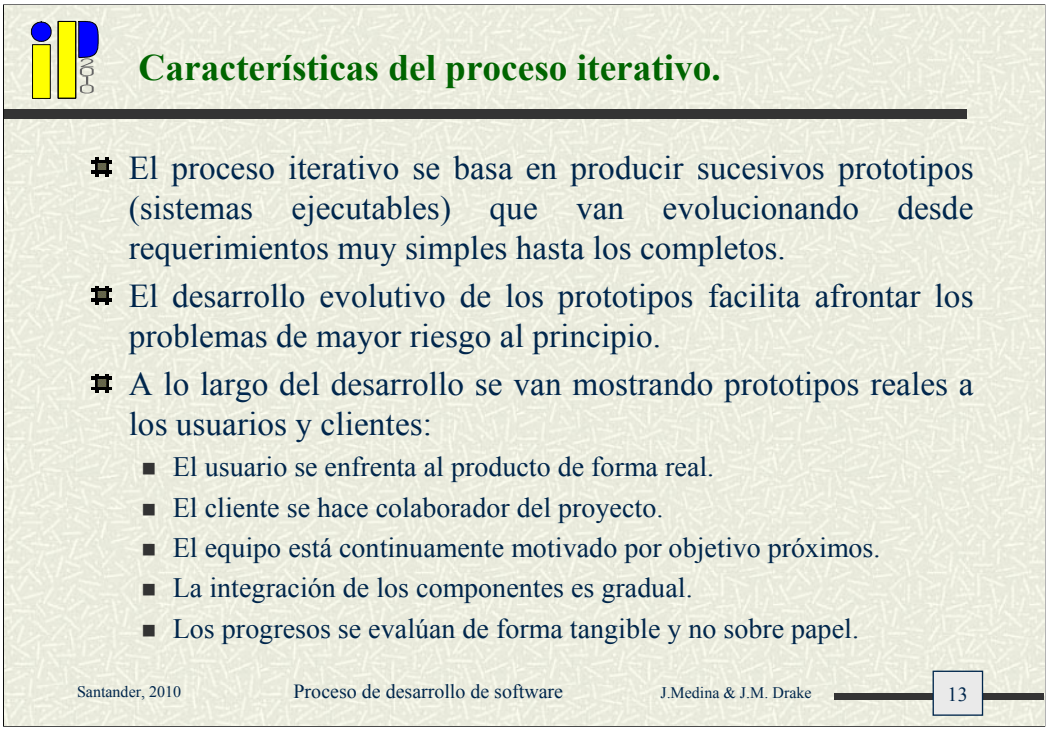
- La aplicación es desarrollada en sucesivas fases por evolución de sistemas mas simples a sistemas mas complejos.
- En la naturaleza y en la tecnología, los sistemas complejos han sido siempre resultado de la evolución de sistemas mas simples.
- La programación orientada a objetos facilita la programación evolutiva:
  - Se diseñan prototipos con solo algunos objetos.
  - Se diseñan prototipos con objetos con funcionalidad limitada.



Santander, 2010 Proceso de desarrollo de software J.Medina & J.M. Drake 12

El ciclo de vida iterativo se basa en una idea simple: Para comprender, diseñar e implementar un sistema complejo, deben emplearse sucesivas etapas en las que en cada una de ellas se realiza una evolución respecto a la anterior.
Los sistemas complejos que funcionan en la Naturaleza y en el campo de la tecnología han resultado siempre de la evolución de sistemas mas simples.
En cada etapa de desarrollo del sistema es necesario conseguir que sea estable frente a los cambios que va a suponer su evolución en la siguiente etapa. Las metodologías orientadas a objetos se construyen alrededor de conceptos como encapsulamiento y modularidad y presentan una mayor robustez frente a cambios. La orientación a objetos favorece el desarrollo de programas según un método iterativo.
Hay varias razones para utilizar el modelo en espiral:
•Con este modelo se pueden reducir riesgos.
•La generación de versiones parciales permite que el cliente se involucre en el proceso de desarrollo de la aplicación.
•Permite reunir a lo largo del proceso métrica del proceso en cada iteración. Y utilizar la información para las siguientes.

El proceso en espiral se ajusta al avance de los proyectos típicos; sin embargo requiere una administración mucho mas cuidadosa, ya que hay que conseguir que la documentación sea consistente después de cada iteración. En particular el código debe corresponder al diseño documentado y debe satisfacer los requisitos documentados.



El proceso se basa en el desarrollo de prototipos ejecutables, y por tanto tangibles y medibles. Las entregas fuerzan al equipo a dar regularmente resultados concretos, lo que evita el síndrome del "90% terminado y 90% por hacer"

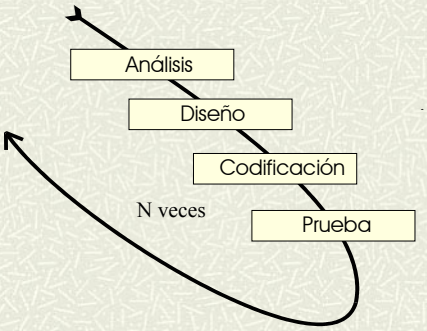
El desarrollo regular de las iteraciones facilita que se aborden desde el principio los problemas, y si se presentan, se tenga un tiempo razonable para resolverlo.

A lo largo del desarrollo se muestran prototipo a los clientes o a los usuarios, lo que supone que:

- El usuario se enfrenta a situaciones de uso concreto y le requiere estructurar mejor sus deseos.
- El usuario se constituye en colaborador del proyecto y asume su responsabilidad.
- El equipo está continuamente motivado por objetivos tangibles muy próximos.
- La integración se hace de forma progresiva y desaparece el Bing-Bang de la integración.
- Los progresos se evalúan sobre sistemas demostrables, lo que hace que los gestores responsables se tomen en serio los resultados intermedios.

N minicascadas.

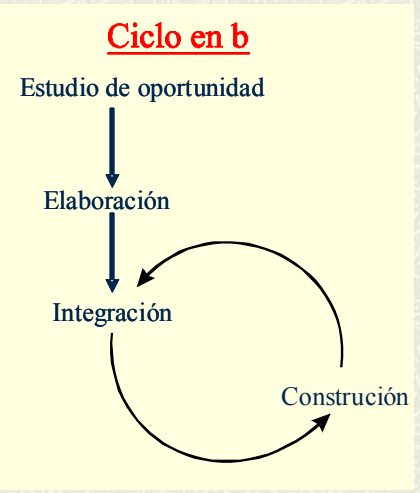
- En el proceso iterativo, cada ciclo reproduce básicamente un ciclo en cascada, solo que con objetivos mas simples.

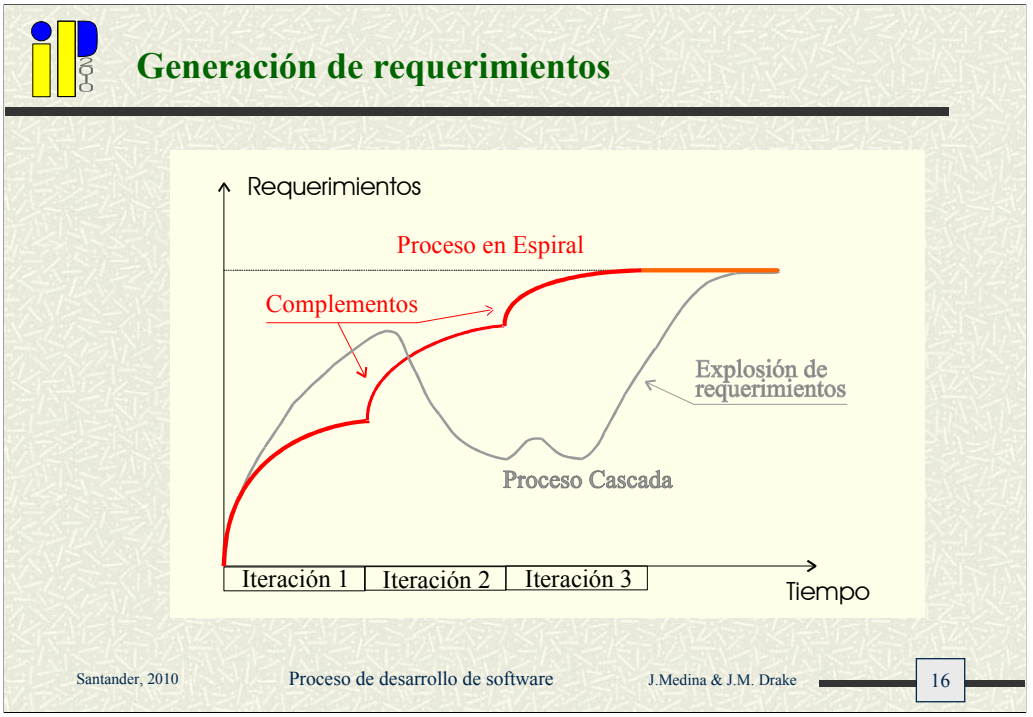


Santander, 2010 Proceso de desarrollo de software J.Medina & J.M. Drake 14

En el ciclo de vida iterativo, cada iteración reproduce el ciclo de vida en cascada, solo que los objetivos son mas simples. Los objetivos de una iteración se establecen en función de la evaluación de las iteraciones precedentes.
Cada iteración comprende las actividades siguientes:
•Planificación:Se elabora en función del estudio de riesgos de los resultados previos.
•Análisis: Estudia los casos de uso y los escenarios a realizar. Se descubren nuevas clases y asociaciones.
•Diseño: Se estudian las opciones necesarias para realizar la iteración. Si se necesita se retoca la arquitectura.
•Codificación y pruebas: Se codifica el nuevo código y se integra con el resultante de iteraciones anteriores.
•Evaluación del prototipo parcial: Los resultados se evalúan respecto a los criterios definidos para la iteración.
•Documentación del prototipo: Se congela y documenta el conjunto de elementos del prototipo obtenido.

Variantes de ciclo iterativo.





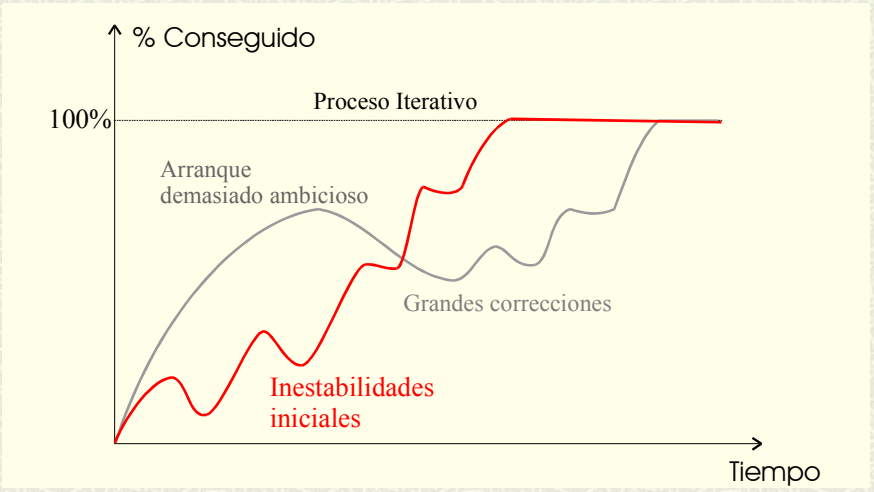
Al método iterativo se le suele atribuir que fomenta la generación ilimitada de requerimientos por parte de los clientes y usuarios.

Este temor no es fundado. Sea cual sea el procedimiento de desarrollo, las necesidades siempre aparecen cuando el cliente o el usuario se enfrenta con la aplicación. Cuanto mejor esté elaborada la fase de especificación, se generarán menos nuevos requerimientos.

En la figura se compara el ritmo de generación de requerimientos en el caso de proceso en cascada y de un proceso iterativo:

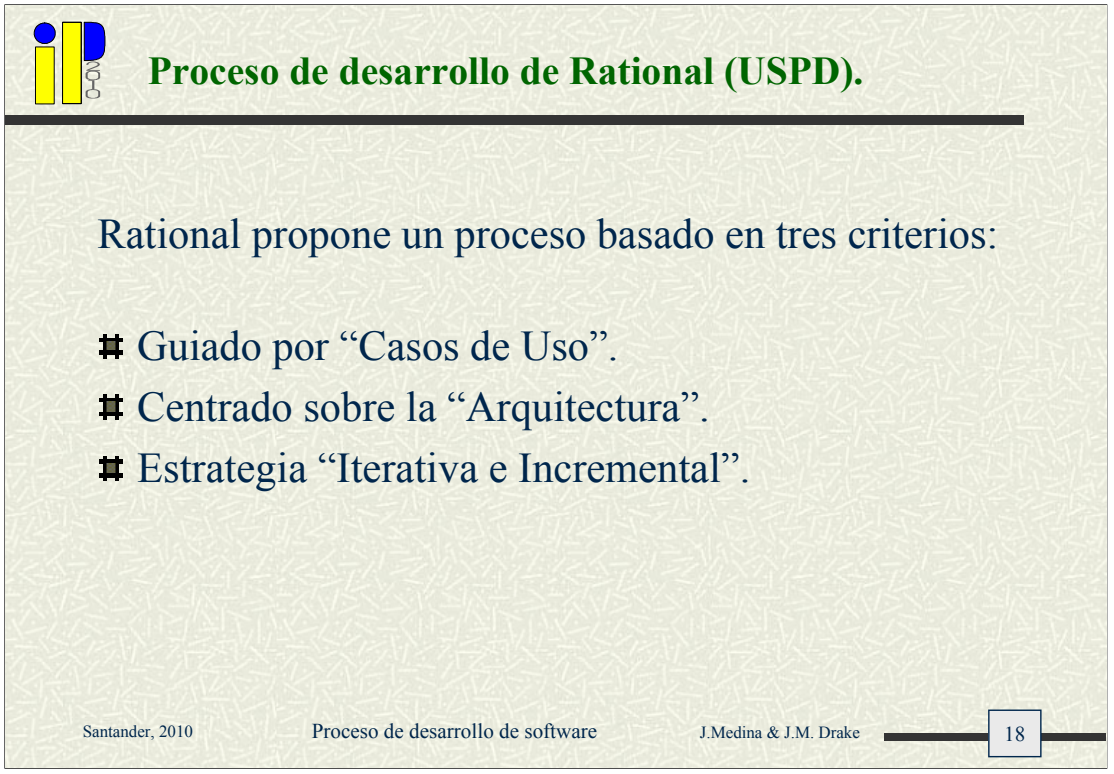
- En el proceso en cascada se definen unos requerimientos iniciales como consecuencia de la fase inicial de especificación. Luego el cliente se desentiende y los requerimientos no suben (si acaso bajan porque el programador trata de evitar los problemas que se le presentan). Al final, después de la integración el cliente se enfrenta con el sistema y se produce una explosión de requerimientos cuando el plazo de finalización está muy próximo.
- En el proceso iterativo, la generación de requerimientos iniciales es la misma. En las sucesivas iteraciones, el usuario se enfrenta con los prototipos y generan nuevos requerimientos incrementales. Con el proceso iterativo no se produce la explosión final de requerimientos.

Planificación de las iteraciones



Santander, 2010 Proceso de desarrollo de software J.Medina & J.M. Drake 17

Para un proyecto de unos 18 meses puede haber entre tres y seis iteraciones. Las iteraciones suelen tener una duración similar.
En la gráfica se comparan dos procesos con diferente planificación de iteraciones:
•La curva gris corresponde a una planificación en la que la primera iteración ha sido excesivamente ambiciosa, y presenta los problemas del ciclo en cascada. La integración presenta grandes problemas. Para resolverlos se pierde mucho del trabajo realizado y se incrementa mucho el tiempo de desarrollo.
•La curva roja presenta un proyecta que ha empleado las dos primeras iteraciones en estabilizar la arquitectura, y con perdida de parte del trabajo realizado se consigue eliminar los riesgo identificados. En las siguientes fases el proyecto avanza de forma incremental pero regular.

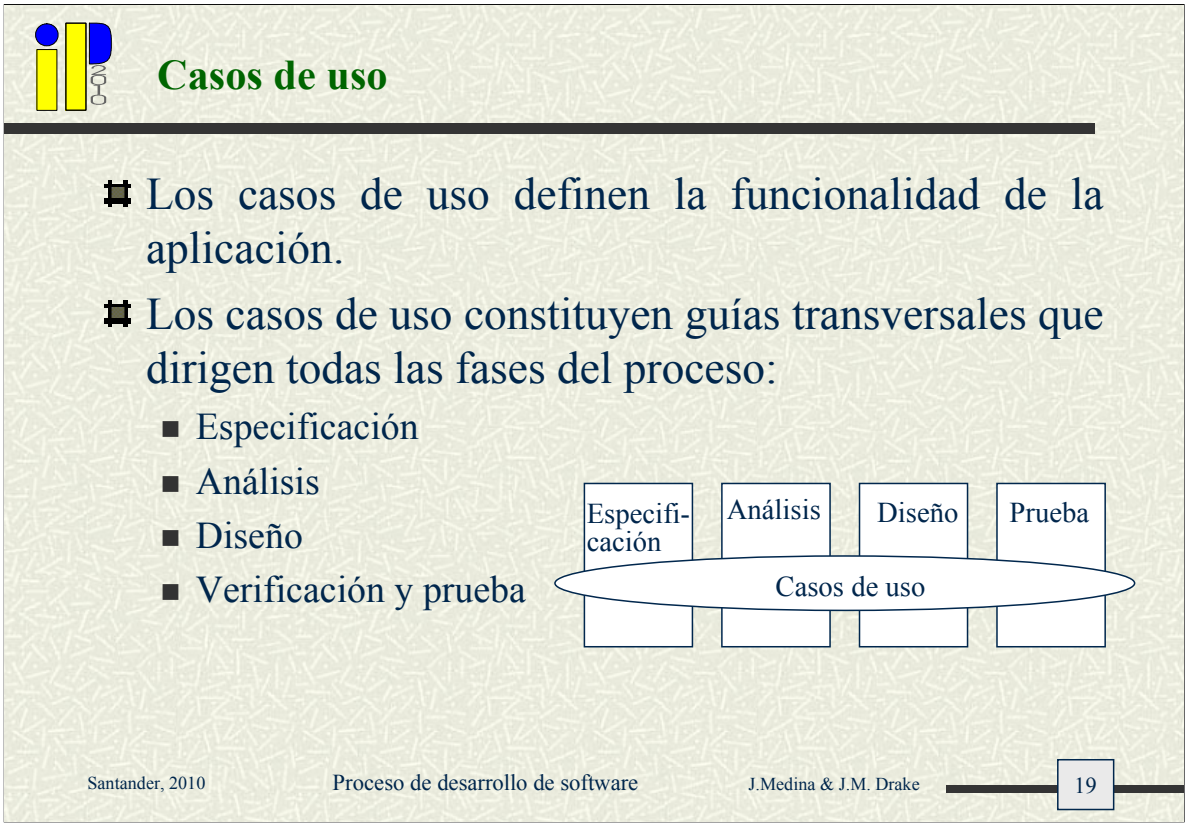


Rational que es una de las promotoras de UML, propone un proceso de desarrollo basado en tres principios:

Controlado por los “Casos de Uso”: Todas las actividades (Especificación, análisis, diseño, verificación y mantenimiento) son guiados por los casos de uso que describen la funcionalidad de la aplicación.

Centrado sobre la Arquitectura: La arquitectura se formula desde el principio del proyecto y se toma como referencia central del proceso. La arquitectura se introduce para satisfacer no solo las necesidades de la funcionalidad, sino también para conseguir flexibilidad frente a la evolución posterior.

Estrategia Iterativa e Incremental: El proceso se divide en pequeñas iteraciones definidas a partir de los casos de uso y de los análisis de riesgos. El desarrollo se realiza por sucesivas iteraciones que proporcionan prototipos incrementales del sistema. Las iteraciones pueden conducirse en paralelo.



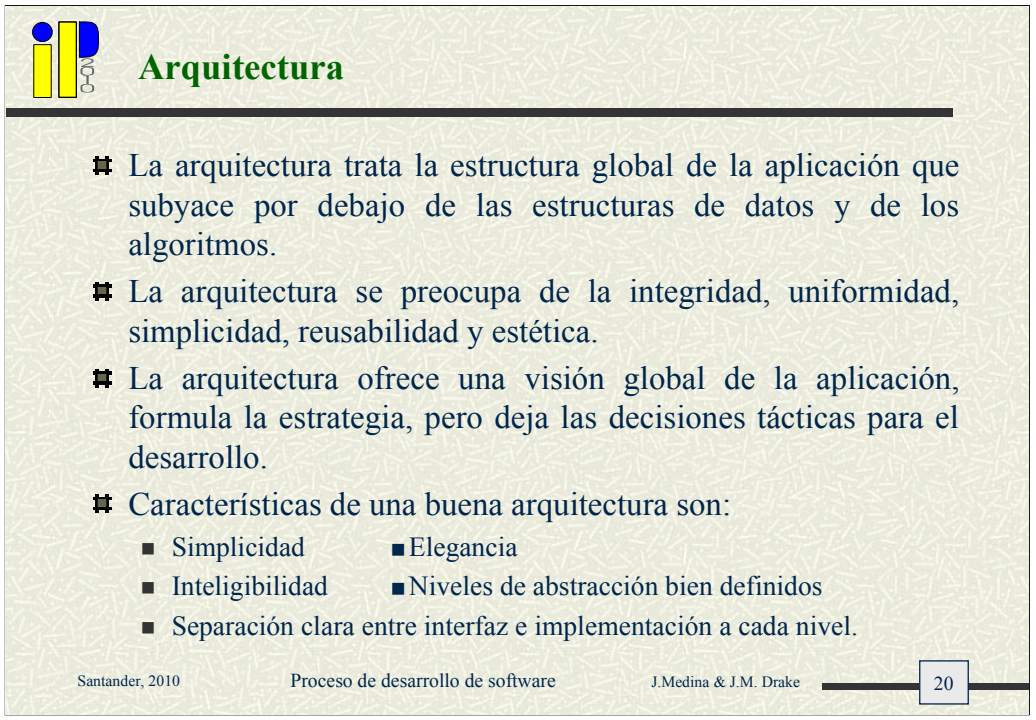
Los casos de uso expresan la funcionalidad que los usuarios requieren de la aplicación que se desarrolla y deben tenerse presente como los objetivos que guían las sucesivas actividades del proceso de desarrollo de la misma.

Los casos de uso formulan las necesidades de los usuarios con el lenguaje de los actores. Los modelos de casos de uso describen los servicios que se esperan del sistema, utilizando para ello la forma de interacciones entre los actores y el sistema.

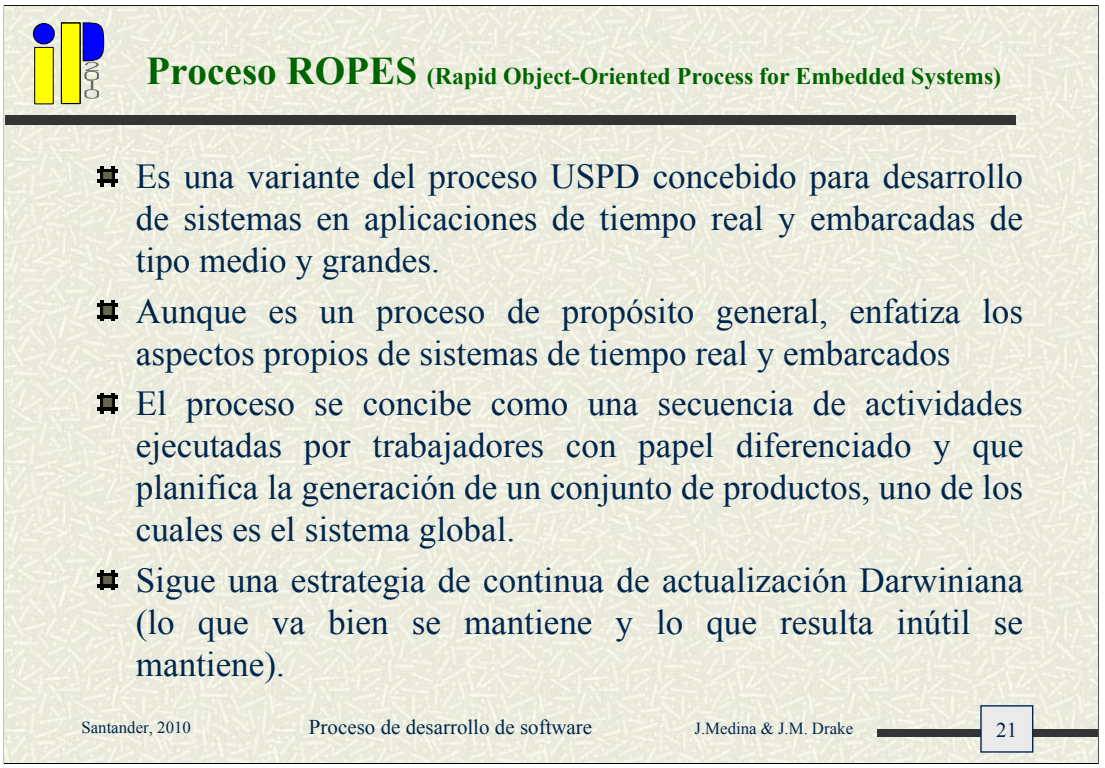
Durante la fase de análisis de los objetos, se comprueba mediante diagramas de secuencias o de colaboración que el conjunto de objetos resultante satisfacen las necesidades requeridas en los casos de uso.

Los casos de uso marcan las necesidades de la aplicación a efectos de la implementación, fundamentalmente en lo que afecta a los requerimientos no funcionales.

Por último, los casos de uso sirven de base para establecer las pruebas funcionales que validan la operatividad de la aplicación.



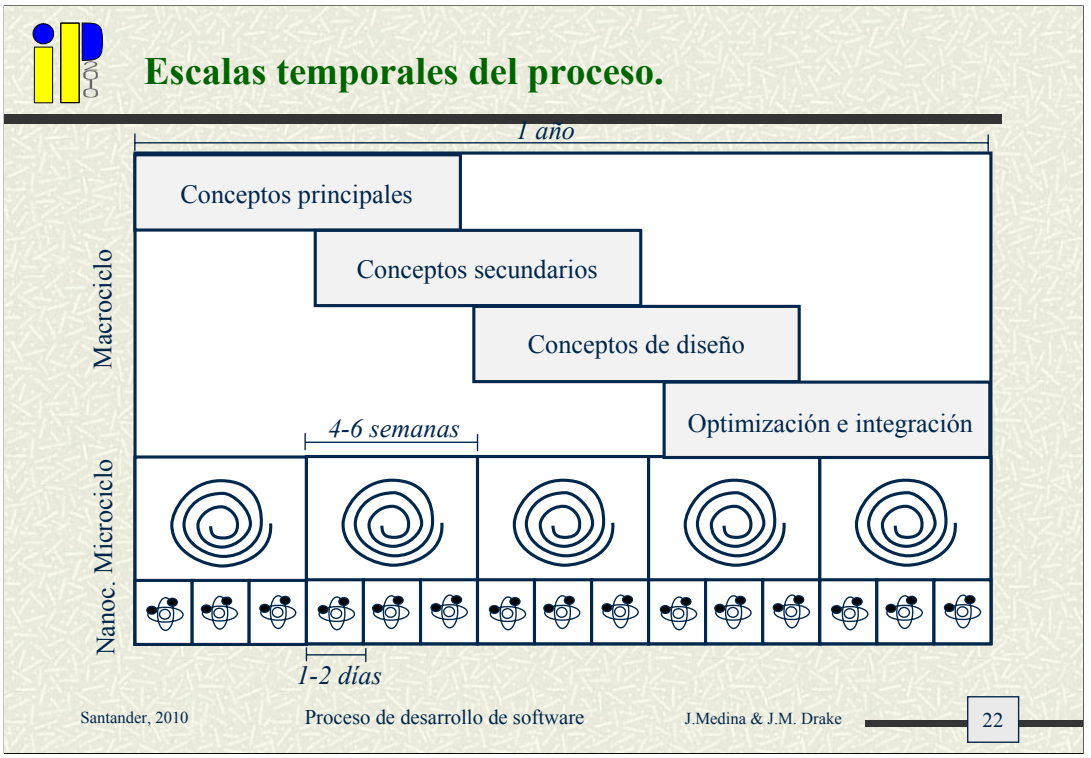
La arquitectura se preocupa de la integridad, uniformidad, simplicidad, reusabilidad y estética de la aplicación.
La arquitectura es la estructura maestra que soporta la aplicación y su formulación estable debe hacerse al principio del proceso de desarrollo.
No existe una arquitectura universal válida para cualquier aplicación, pero si existen arquitecturas reutilizables dentro de un ámbito concreto.
La arquitectura se ha definido como la unión de Componentes, Formas y Motivaciones. Los componentes son las clases y objetos, las formas son las agrupaciones de clases y objetos (patterns), y las motivaciones explican por qué cada agrupación es la adecuada en cada contexto.
Características que debe ofrecen una buena arquitectura son:
•Simplicidad: Entendida como una estructura basada en pocos y bien definidos criterios.
•Elegancia: Su estructura se basa en líneas bien definidas que se mantienen uniformemente a lo largo de toda la aplicación y en la ausencia de atajos y casos especiales que la oscurecen.
•Inteligibilidad: Debe ser fácil de comprender, ya que va a ser utilizada por todos los que intervienen en el desarrollo de la aplicación y que la utilizan como guía de sus actividades.
•Niveles de abstracción bien definidos: La abstracción correcta de los componentes que se propongan a nivel de la arquitectura es la principal herramienta para simplificar la estructura y facilitar su evolución y reusabilidad.
•Separación entre interfaz e implementación: La separación sistemática entre interfaces e implementaciones, ayuda al mantenimiento de la aplicación. Si en el mantenimiento se respetan las interfaces, se pueden modificar las implementaciones de un componente sin efectos secundarios sobre otros.



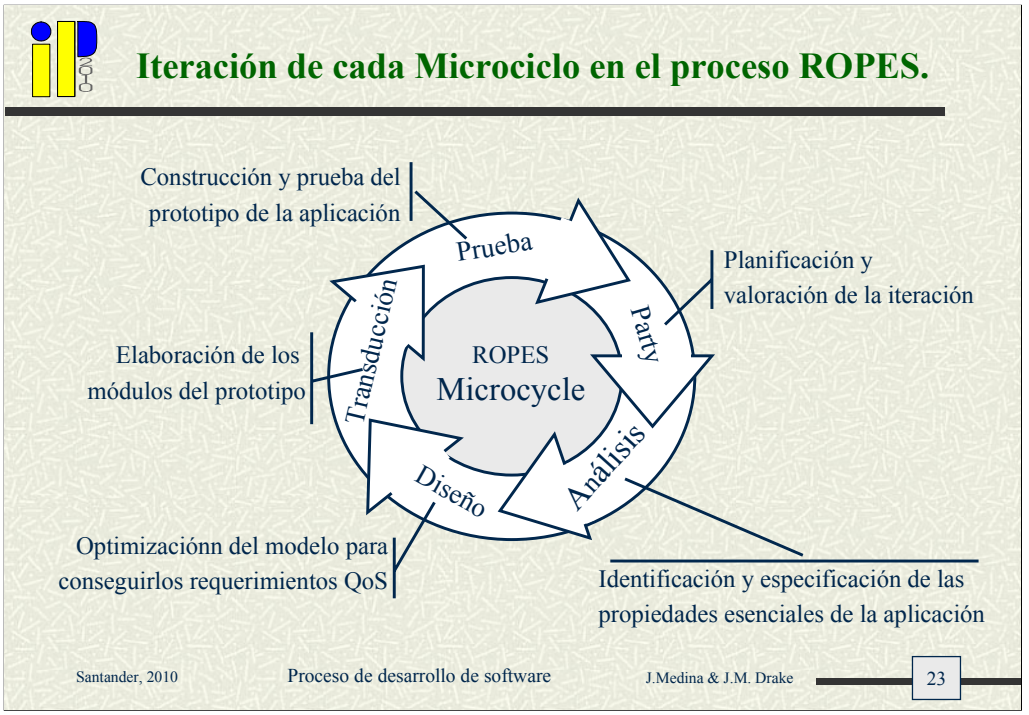
El proceso ROPES (Rapid Object-Oriented Process for Embedded Systems) es un proceso de desarrollo de aplicaciones software de propósito general (aplicable a cualquier tipo de aplicación) que enfatiza los aspectos de ingeniería software, integración de los sistemas y sus pruebas. Se ha venido utilizando en la empresa Artisan durante los últimos 25 años, aunque ha evolucionado intensamente en función de las tecnologías que van estando disponibles, y siguiendo en la evolución una estrategia darwiniana (lo que va bien se mantiene y lo que resulta inútil se mantiene).

El proceso ROPES es altamente escalable, y ha demostrado su eficacia tanto para proyectos pequeños realizados por dos o tres personas como por proyectos grandes realizados por equipos de cientos de personas..

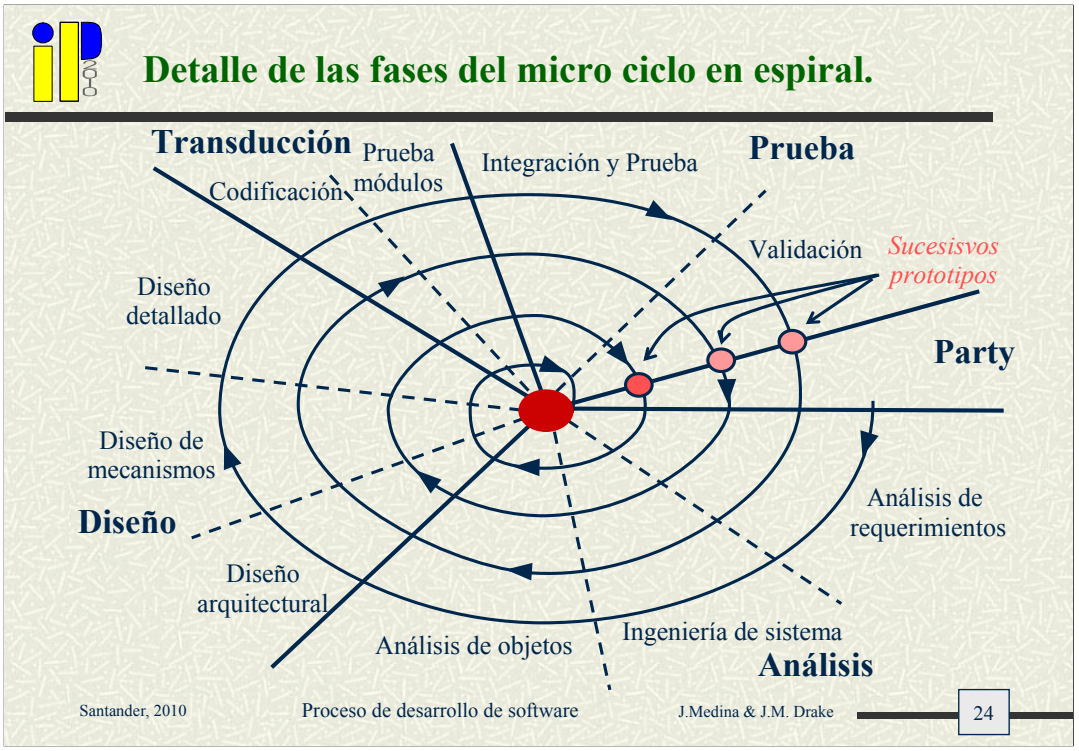
ROPES se desarrolla con un único objetivo, producir aplicaciones software con un mínimo esfuerzo, sin fallos y con la máxima predecibilidad de su desarrollo.



El proceso ROPES utiliza tres diferentes escalas de tiempo simultáneas.
El Macrociclo hace referencia al desarrollo completo del proceso, desde que se inicia su especificación hasta que se entrega el producto. Dentro de él se desarrollan solapadamente las cuatro fases de un proceso tipo cascada, y su objetivo es dirigir la evolución de los prototipos que se programan, de forma que sucesivamente traten de resolver los problemas sobre conceptos claves, requerimientos, arquitectura y tecnología . En nuestro caso, se toma como referencia un proyecto de un año de duración.
El Microciclo tiene un ámbito mas reducido y su objetivo es el desarrollo de un nuevo prototipo con una funcionalidad limitada que resuelva actividades consideradas de riesgo en el proceso de desarrollo. La duración adecuada para cada una de estas fases es de entre 4 y 6 meses.
El Nanociclo es el que tiene el ámbito mas reducido y su objetivo es modelar ejecutar o establecer ideas que se necesitan. Su duración no debe exceder los 2 días. La razón por la que se introduce esta fase es la conveniencia de probar y estar seguro de forma continuada todas las ideas que se incorporen al proyecto.

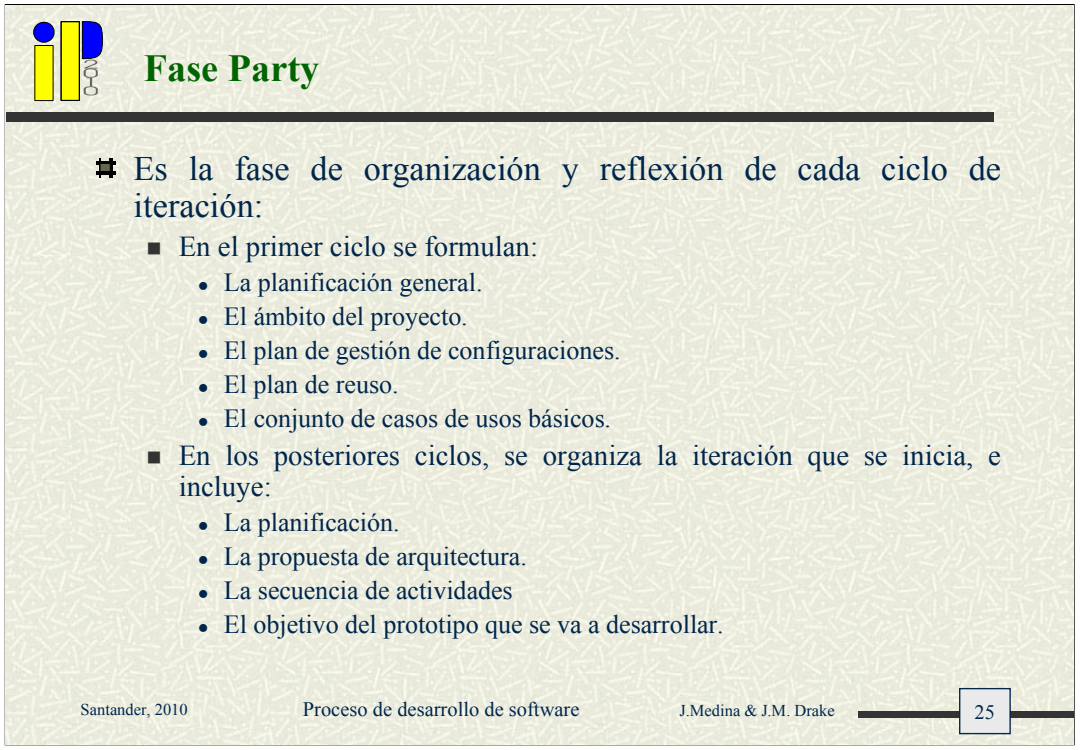


En el proceso ROPES el microciclo tiene el objetivo de producir un nuevo prototipo que implemente algunos casos de uso parciales y que resuelva problemas o dudas que introduzcan riesgos.
La fase "Party" es donde se establece el objetivo del microciclo y se planifican las actividades a ser realizadas.
La fase "Análisis" define los aspectos esenciales del prototipo que se desarrolla.
La fase "Diseño" optimiza el modelo de análisis eligiendo la arquitectura de diseño adecuada, seleccionando la tecnología y aplicándola al modelo de análisis.
En la fase "Transducción" se genera manualmente o automáticamente el código, se compila y enlaza las nuevas piezas para constituir un prototipo ejecutable.
La fase "Prueba" integra las piezas generadas, aplica los vectores de pruebas previos y los generados específicamente en este microciclo y genera realiza un análisis del estado actual del desarrollo.



Cada iteración del microciclo del proceso en espiral ROPES supone realizar las 4 fases que ya se han descrito. En este gráfico se detallan sus subfases:

- Fase Party: Se evalúan las características del proyecto que convienen abordar, se establece el objetivo del microciclo y se planifican las actividades a ser realizadas.
- Fase Análisis: Define los aspectos esenciales del prototipo que se desarrolla. Por esencial se entienden "el que" es el sistema, esto es, aquellas propiedades que en su ausencia se considera el sistema erróneo o incompleto. Típicamente es una vista de caja negra en la que los detalles sobre sobre la estructura interna detallada es irrelevante.
- Fase Diseño: Se optimiza el modelo de análisis eligiendo la arquitectura de diseño adecuada, seleccionando la tecnología y aplicándola al modelo de análisis. En el diseño se justifica y describe "el como" las características establecidas en el análisis son implementadas. Corresponde a una vista mas detallada, en la que los detalles son relevantes en esta fase. Un diseño es una solución particular de las muchas posibles, y lo mas relevante de esta fase es establecer y satisfacer las características de QoS que deben ser optimizadas y en función de las cuales se justifica la solución.
- Fase Transducción: Concierne a la construcción de la implementación correcta de los elementos estructurales propuestos en el diseño. Se incluye la generación manual o automática del código, se compila y enlaza las nuevas piezas para constituir un prototipo ejecutable.
- Fase Prueba: Se integra el prototipo a partir de las piezas y módulos generadoas, aplica los vectores de pruebas previos y los generados específicamente en este microciclo y genera realiza un análisis del estado actual del desarrollo.



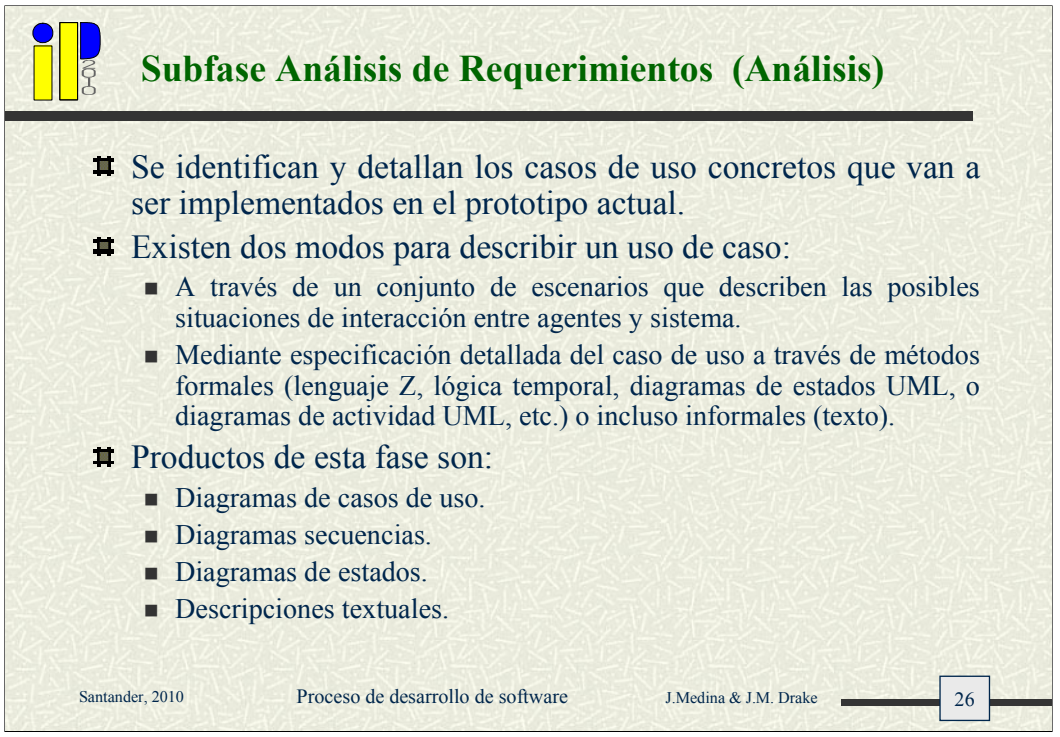
Uno de los errores mas serios que se pueden cometer en el desarrollo de un proyecto es que no se realice la valoración del estado del proyecto y no se ajuste su ejecución de acuerdo con sus estado. Esto supone dos aspectos igualmente importante: tener capacidad de evaluación del estado, como capacidad de reconducir el proceso de acuerdo con el resultado de la evaluación, a través de reasignar recursos, replanificar actividades, reducir o incrementar el ámbito, etc..

Debido a que la selección e implementación de una arquitectura correcta es de gran importancia para el éxito global del proyecto, es muy importante evaluar al inicio de cada ciclo la correctitud de la arquitectura propuesta y si debe ser cambiada. Para ello es necesario estimar:

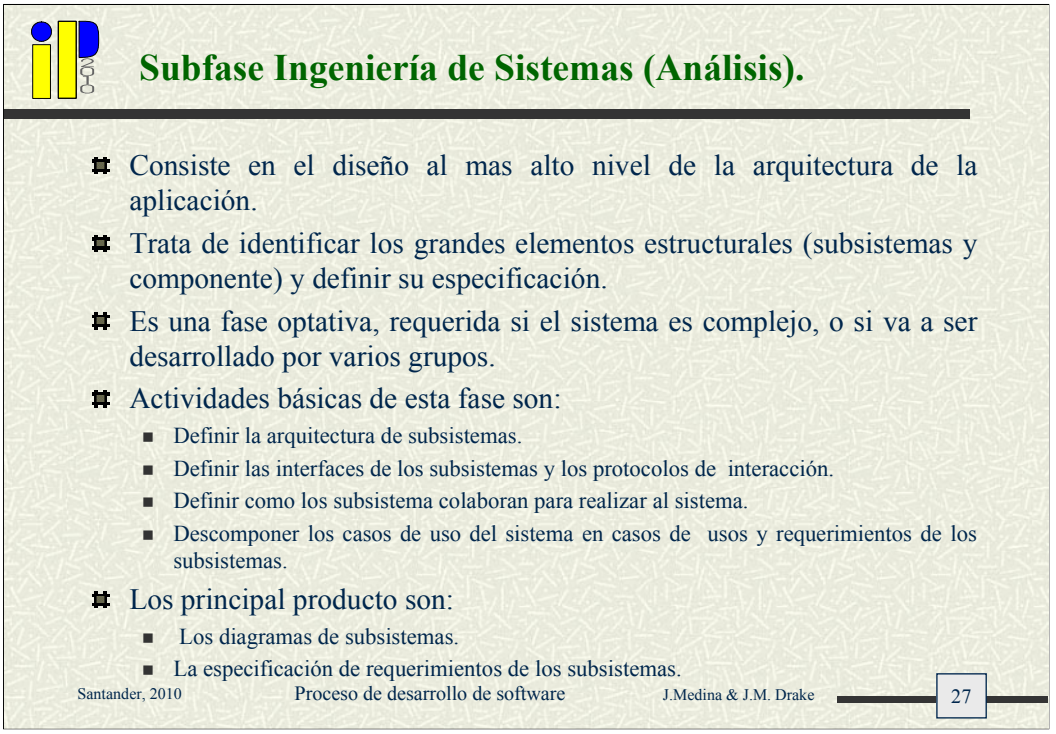
•Si la arquitectura permite conseguir las características QoS que se requieren.

•Si la arquitectura soporta razonablemente la evolución y crecimiento de la aplicación.

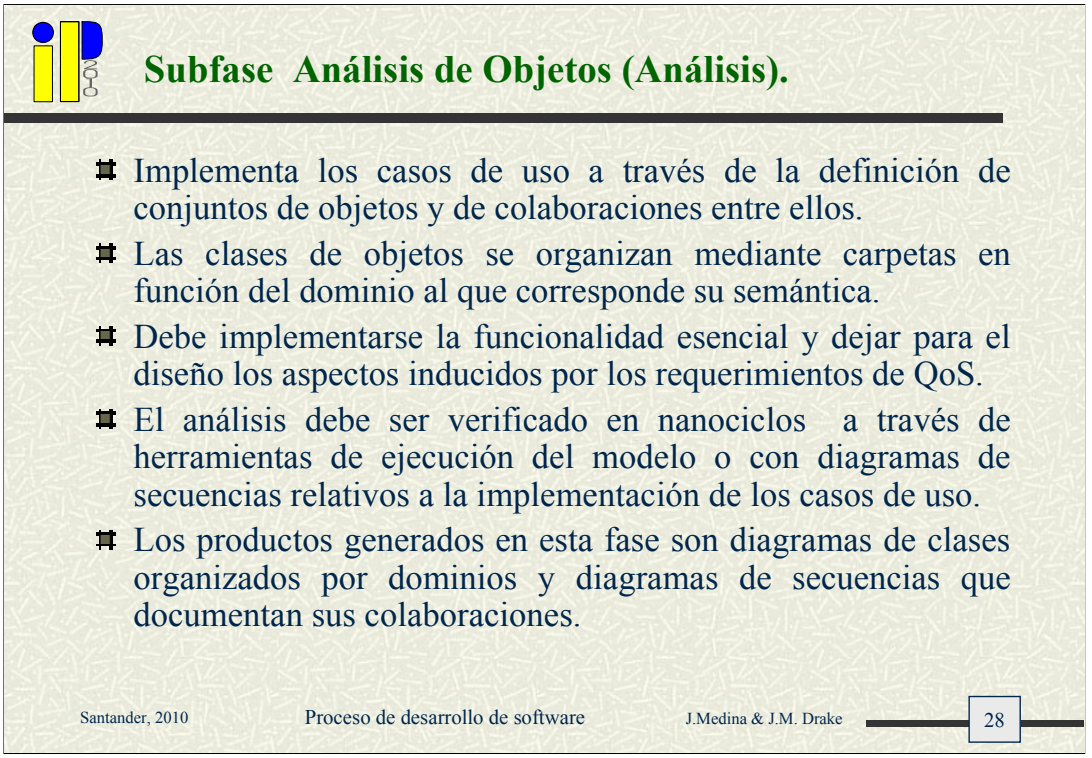
Si en cada ciclo se requiere continuos cambios estructurales de los elementos ya desarrollados, es razonable pensar que la arquitectura elegida no es la adecuada para la aplicación.



En la fase de Requirements analysis, los requerimientos del prototipo que se va a desarrollar en la iteración, se identifican y se describen con detalle. Los casos de uso se ha seleccionado en la fase Party, pero hasta esta fase no se detallan en todos sus aspectos.
Hay dos modos básicos de describir un caso de uso:
•Por medio de ejemplos, a través de un conjunto (posiblemente amplio) de escenarios que describen la posibilidades de interacción y todas su variantes. Los escenarios son habitualmente descritos mediante diagramas de secuencias. La ventaja de este método es que es muy accesible a los clientes no técnicos. La principal dificultad es que a menudo se requiere un número excesivo de escenarios para describir de forma completa un caso de uso, y que también es difícil de describir requerimientos negativos.
•Mediante una especificación sistemática utilizando técnicas formales (lenguaje Z, lógica temporal, diagramas de estados, diagramas de acrtividad, etc) o descripciones textuales. La ventaja de este método es mas preciso y conciso. El inconveniente es que suelen ser difíciles de comprender por los usuarios no técnicos.
La solución mas adecuada es el uso mixto de ambos métodos.



La fase de Ingeniería de sistemas es el diseño arquitectural al máximo nivel del prototipo que se desarrolla en la iteración actual. Es una subfase opcional que solo se ejecuta en sistemas complejos en los que van a ser desarrollados por varios grupos de trabajo. El propósito de la fase es descomponer el sistema en módulos independientemente especificados (subsistemas) que puedan ser desarrollados independientemente por los diferentes equipos.
Las actividades primarias de esta fase son:
•Definir la arquitectura de subsistemas.
•Definir las interfaces y los protocolos de interacción de los subsistemas definidos.
•Definir como los subsistemas colaboran entre sí para implementar los casos de uso del sistema, especificando los papeles que juegan cada subsistema en la colaboración, pero sin detallar su estructura interna.
•Descomponer los casos de uso y los requerimientos del sistema en casos de uso y en requerimientos de los subsistemas.
El producto primario usado en esta fase son los diagramas de subsistema. Estos son diagramas de clase UML en los que los elementos primarios son subsistemas, controladores, interfaces, actores y diagramas de secuencia de alto nivel. Sobre cada subsistema definido en esta fase debe ser aplicado y generar los mismos productos que se desarrollaron en la fase de análisis de requerimiento para el sistema completo.



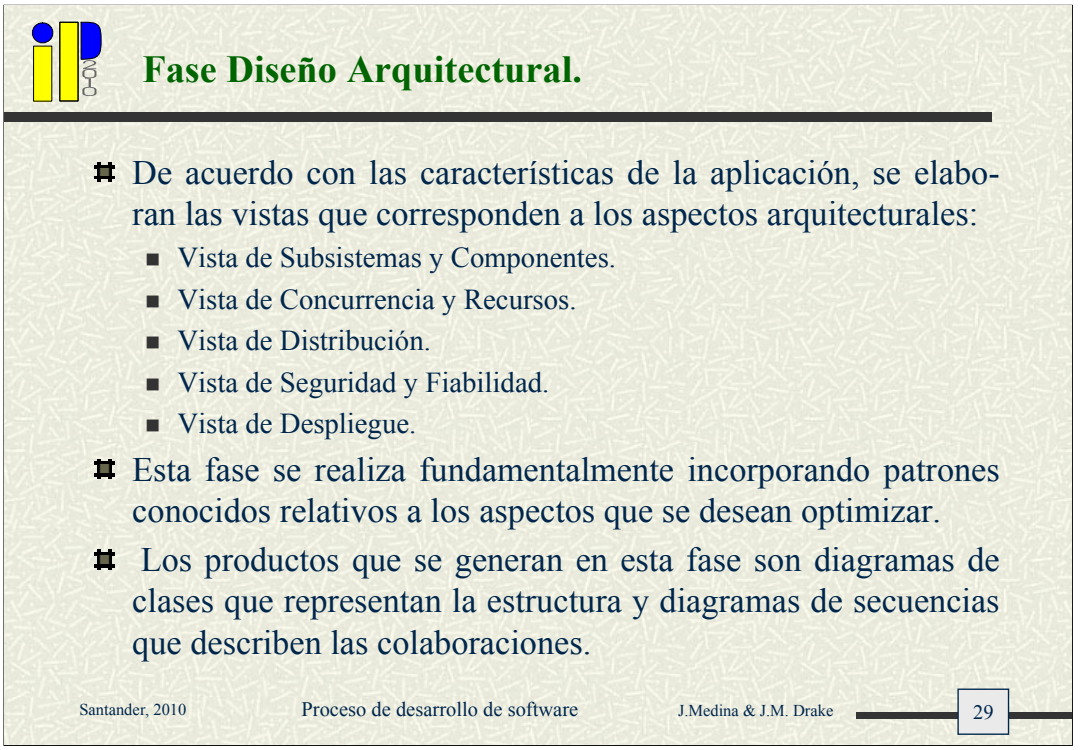
Un caso de uso es una caja negra que representa la funcionalidad del sistema a través de las interacciones de los actores y el sistema. En esta fase se establece la estructura interna del sistema. Cada caso de uso se concibe como la colaboración de un conjunto de objetos que responden al dominio o dominios de la aplicación,que deben ser identificados y caracterizada sus interfaces en esta fase.

Si la fase previa de ingeniería de sistemas está presente, esta fase debe realizarse separadamente por subsistema.

Debe tenerse cuidado de minimizar la introducción de aspectos relativos al diseño. Solo debe capturarse la funcionalidad esencial, y dejar los aspectos relativos a los requerimientos de QoS para próximas fases.

Debe plantearse si el esquema de objetos que resulta del análisis es correcto, y para ello, en sucesivas fases de nanociclos, deben validarse la consistencia y completitud de la estructura que se propone. Si se disponen herramientas de ejecución del modelo, esto puede automatizarse, en caso contrario, pueden realizarse a través de familias de diagramas de secuencias que ilustren los mecanismos de colaboración para los escenarios contemplados en los casos de uso.

Los productos que se generan en esta fase son los diagramas de clases que modelan los objetos definidos, y los diagramas de secuencias que ilustran sus colaboraciones.



Hay muchas formas de definir el termino arquitectura. En este curso la entendemos como el conjunto de estrategias y decisiones de diseño que afectan globalmente al sistema. La arquitectura se refiere principalmente a la organización estructural del sistema, y solo implícitamente como objetivo final tiene en cuenta los aspectos de comportamiento y funcionales del sistema.

La arquitectura se puede diseñar a base de estudiar cinco vista complementarias que hacen referencia a aspectos estructurales típicos en los sistema:

•Vista de subsistemas y componentes: Hace referencia a los grandes módulos (subsistemas y componentes) con que se construyen la aplicación.

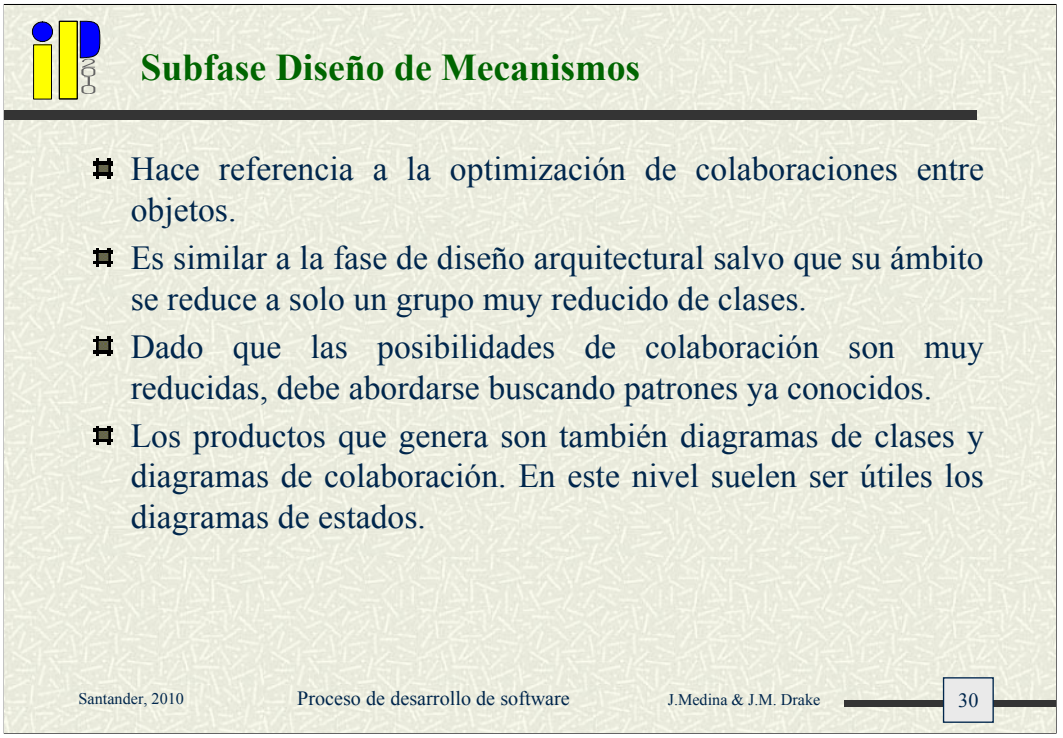
•Vista de Concurrencia y Recursos: Hace referencia a los niveles de concurrencia que se deben incorporar en la aplicación y en los recurso necesarios para una operación concurrente segura.

•Vista de Seguridad y Fiabilidad: Hace referencia a los niveles de redundancia necesarios para hacer fiable el sistema y a la gestión de los fallos que puedan producirse en la aplicación.

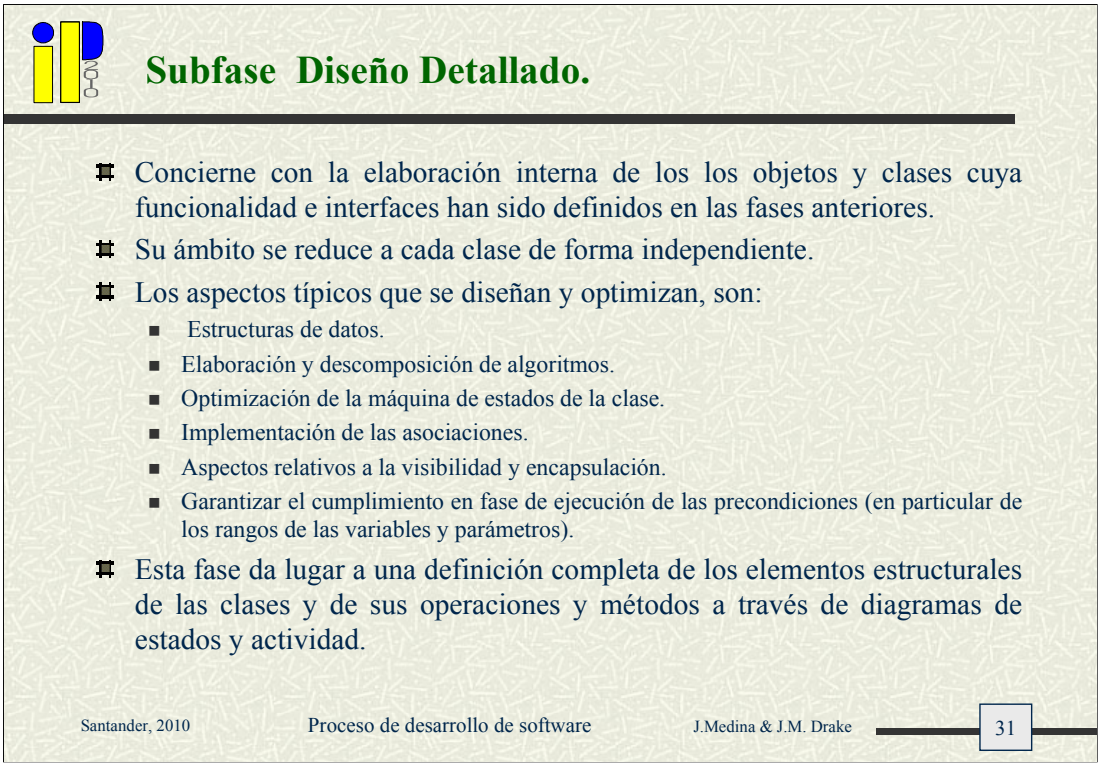
•Vista de Distribución: Hace referencia a como se establecen diferentes espacios de direcciones, como se distribuyen los módulos software entre ellos, y los mecanismos, formatos y protocolos de colaboración entre ellos.

•Vista de Despliegue: Hace referencia a la forma de mapear los componentes software en los diferentes elementos físicos (procesadores, discos, dispositivos, etc.) que constituyen la plataforma hardware del sistema.

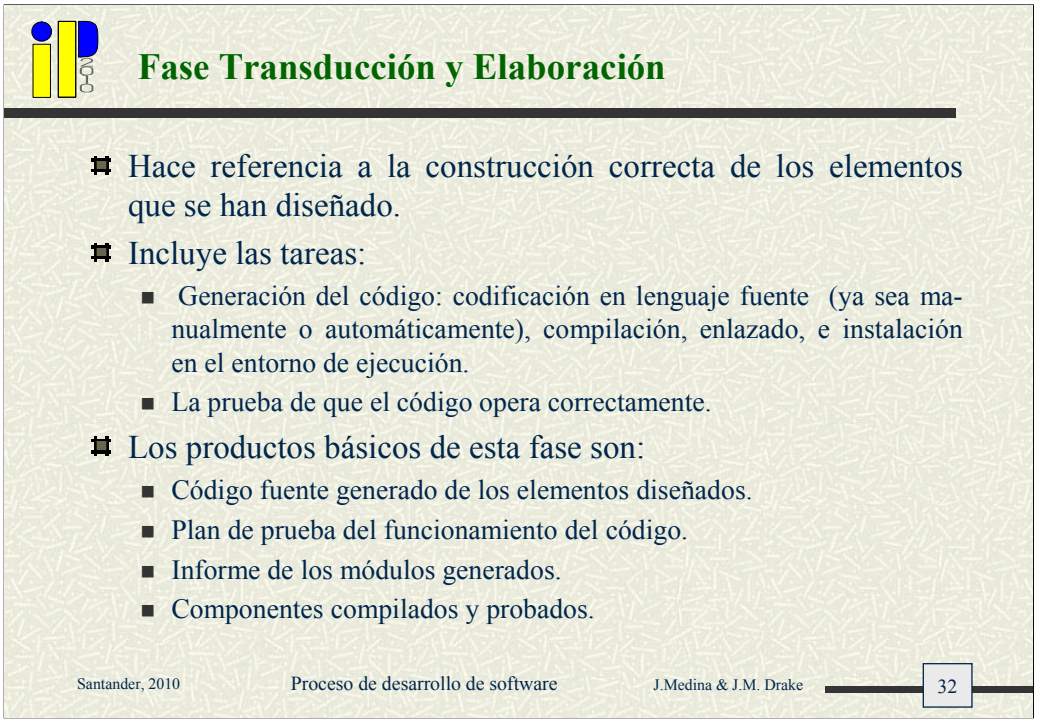
La representación del diseño arquitectural utiliza los tipos de vistas propios de un descripción estructural, esto es, diagramas de clases para definir los objetos y diagramas de secuencia para describir las colaboraciones entre ellas.



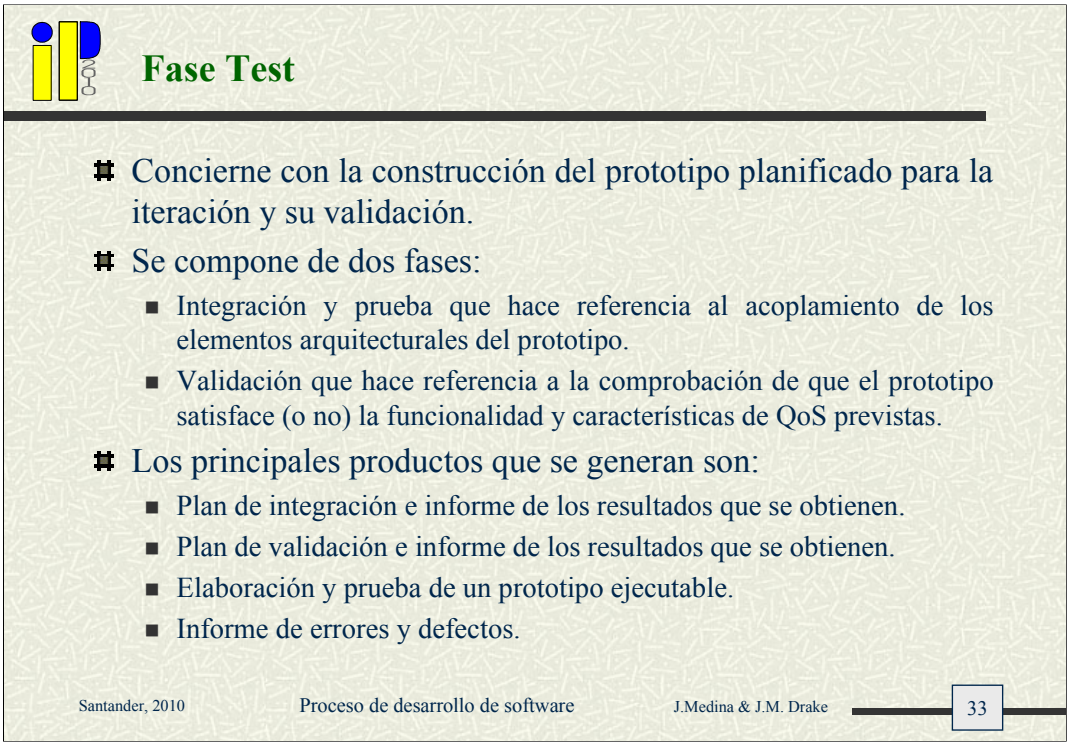
La fase de diseño de mecanismos hace referencia a la optimización de colaboraciones en grupos individuales reducidos. Es un tipo de tarea similar a la realizada en la fase de diseño de la arquitectura, salvo que se reduce localmente a un mecanismo de colaboración.
Al igual que en el diseño de la arquitectura, el diseño de los mecanismos debería basarse en patrones de diseño, cuyos beneficios y efectos sean bien conocidos, sin embargo estos patrones son diferentes de los patrones estructurales.
La vista que se genera en esta fase de diseño de mecanismos se compone de diagramas de clases que completan la definición de los objetos que intervienen, así como diagramas de descripción de las interacciones tales como diagramas de secuencias, de colaboración y diagramas de estados. A este nivel, estos dos últimos tipos de diagramas son muy útiles.



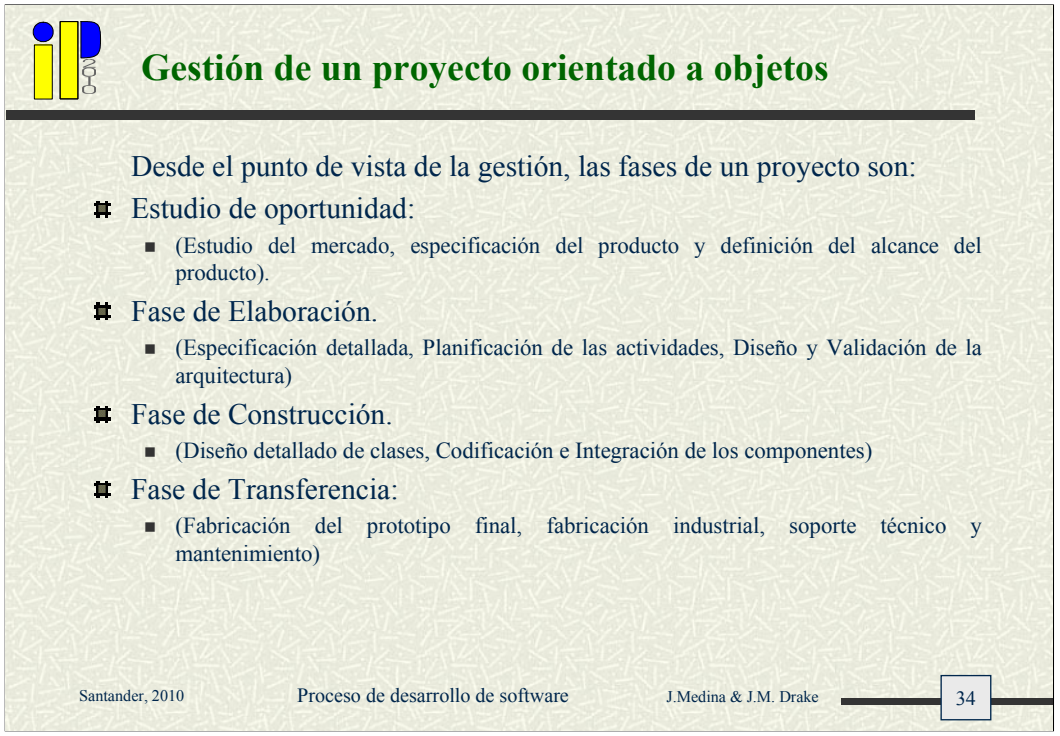
La fase de diseño detallado elabora los aspectos internos de los objetos y clases, y su ámbito se reduce a cada clase de forma independiente.
Los aspectos de la clases que son habitualmente tratados en esta fase son:
•La estructuras de datos internas de las fases.
•La implementación y optimización de los algoritmos y su descomposición cuando son complejos.
•La optimización de la máquina de estados de los objetos.
•La estrategias de elaboración de los objetos.
•Las estructuras con las que se elaboran las asociaciones, en particular cuando son múltiples y requieren mecanismos contenedores.
•La implementación de las visibilidades, en especial cuando son cruzadas y no son directamente implementables, así como los criterios de modularización y encapsulamiento de las clases.
•Estudio de los mecanismos de garantizar en tiempo de ejecución que las precondiciones son satisfechas y como responder cuando no se satisfacen.
Hay muchos criterios y guías para realizar el diseño de las clases, y suele corresponder con las estrategias de programación básica. No suelen basarse en patrones sino en bloques de sentencias proporcionadas por el lenguaje.
El resultado de esta fase son clases completamente definidas, en las que su estructuras de datos internas quedan perfectamente definidas, así como los diagramas de estados y actividades que especifican las operaciones y métodos de su interfaces.



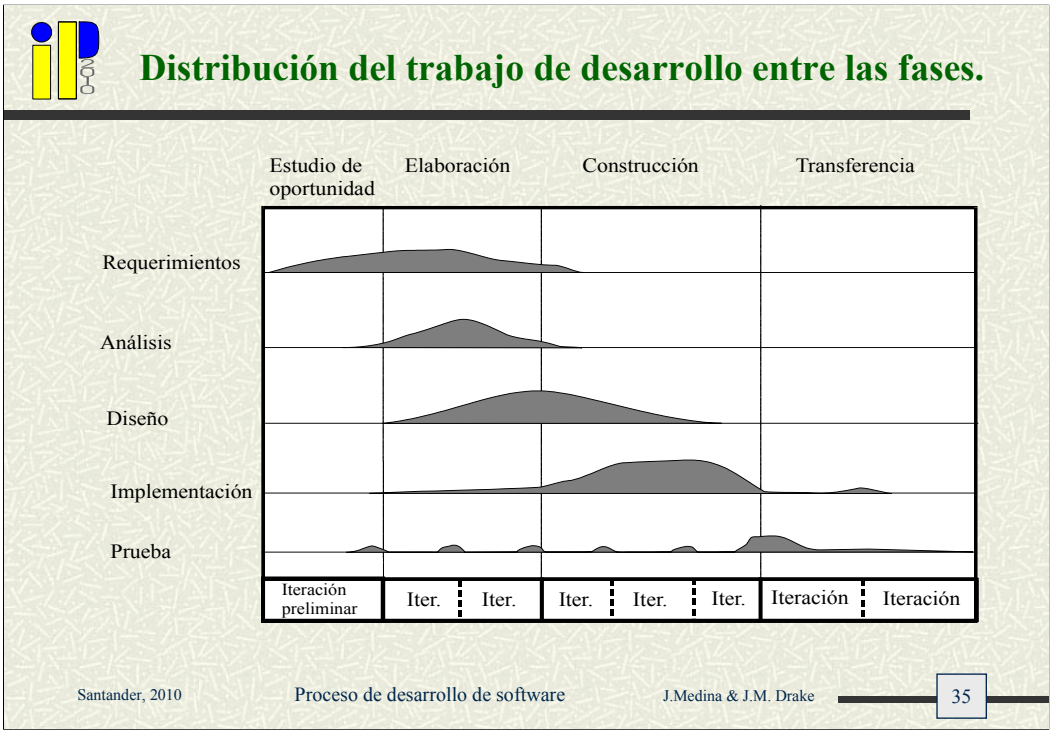
La fase de transducción concierne con la correcta construcción de los elementos arquitecturales que trabajen correctamente.
Esta fase incluye la generación del código (que puede generarse automáticamente por herramientas o manualmente, o por combinación de ambas que es lo mas habitual, la prueba a nivel de código fuente de los algoritmos e interacciones, la compilación y compatibilidad de enlazado de los módulos compilados y la instalación y ejecución de del código ejecutable generado. Así mismo, debe establecerse un plan de verificación de los elementos construidos.
Los principales productos a que da lugar la fase de traducción y elaboración son:
•El código fuente de los elementos definidos en el diseño.
•Un plan de prueba , de las asociaciones, de las operaciones y de las interacciones.
•Un informe sobre los módulos elaborados y su forma de utilización.
•Los componentes de software compilados.



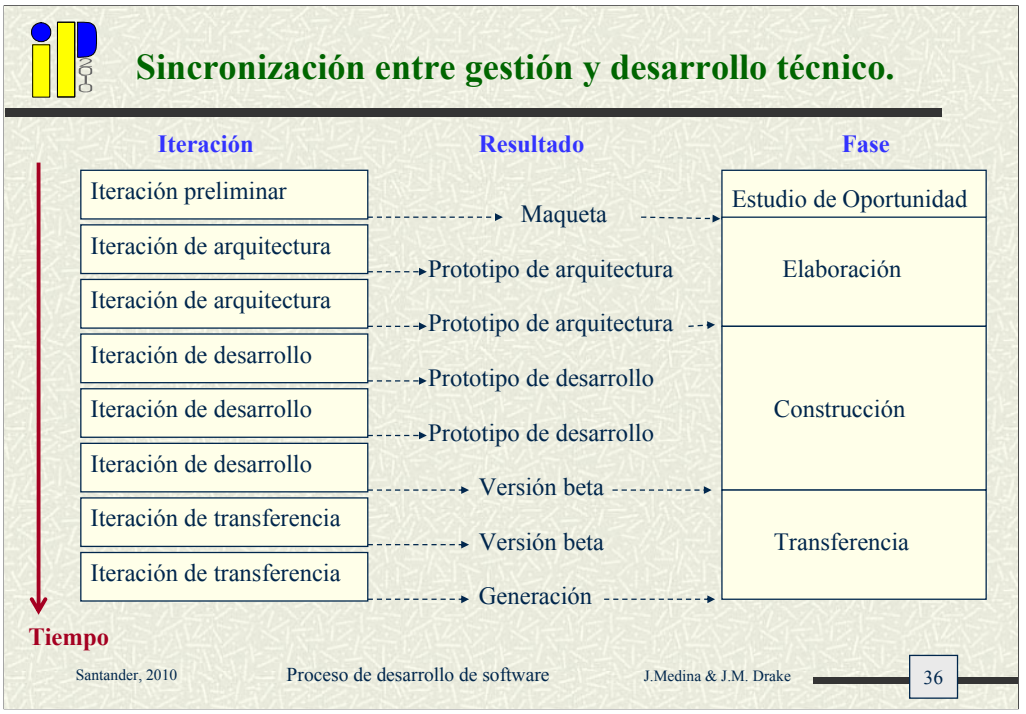
La fase de Test construye (Integración y prueba) el prototipo con los elementos diseñados y verifica si el prototipo satisface o no (Valida) los requerimiento funcionales y de QoS establecidos para él.
La prueba se limita a demostrar que las interfaces de los elementos estructurales operan correctamente y satisfacen las restricciones establecidas. El proceso suele se incremental, se van integrando sucesivos elementos estructurales y se va verificando que operan correctamente.
Si se encuentran defectos y problemas durante esta fase, deben ser reportados y analizados ya que son el objetivo final de la iteración, y deben ser considerados como la base de la siguientes iteraciones.



La gestión del desarrollo de un sistema software trata los aspectos financieros, estratégicos, comerciales y humanos:
Se puede descomponer cuatro fases:
•Estudio de la Oportunidad: que comprende el estudio del mercado,la especificación del producto final y la definición del alcance del proyecto.
•Fase de Elaboración:que corresponde a la especificación de las particularidades del producto, a la planificación de las actividades, a la determinación de los recursos, al diseño, y a la validación de la arquitectura.
•Fase de Construcción: agrupa la realización del producto, la adaptación de la arquitectura, la codificación y la integración)
•Fase de Transferencia: que corresponde a la fabricación del prototipo final, de la fabricación industrial, distribución entre usuarios,, soporte técnico y mantenimiento.



La definición de los requerimientos se realiza básicamente entre las fases de Estudio de oportunidad y de Elaboración. Durante el Estudio de viabilidad se identifican los casos de uso básicos y los restantes secundarios se establecen en la fase de Elaboración.
El análisis está claramente centrado en la fase de Elaboración. El problema se comprende y su estructura se formula mediante un modelo abstracto.
El diseño se realiza entre las fases de Elaboración, en la que se obtienen un modelo concreto de la solución y se formula y estabiliza la arquitectura.Durante la fase de Construcción se definen los detalles de los componentes, se organizan los módulos y se despliegan sobre la plataforma.
La implementación se desarrolla básicamente durante la fase de construcción en la que se codifican e integran la mayoría de los componentes del sistema.
Por último, las pruebas se distribuyen a lo largo de todas las fases coincidiendo con las finalizaciones de las iteraciones.



El proceso de desarrollo técnico y las fases de gestión del proyecto se sincronizan al final de cada fase, sobre el resultado tangible de una iteración.
El estudio de oportunidad puede necesitar un prototipo para estudiar la viabilidad técnica del proyecto y acabar de definir su especificación. La toma de decisión de proseguir o no con el proyecto se toma teniendo en cuenta la evaluación de la maqueta entregada por el equipo de desarrollo.
La fase de Elaboración debe concluir con la definición de una arquitectura estable, y esto suele requerir varios prototipos. La fase de Elaboración concluye con una arquitectura validada por un prototipo.
La fase de Construcción progresa según se van desarrollando los sucesivos prototipos de desarrollo. Cuyos objetivos son afianzar los progresos y garantizar la estabilidad del proyecto. La fase de Construcción termina con el prototipo definitivo en su versión beta, que se entrega a los usuarios para que lo prueben bajo condiciones reales.
La fase de Transferencia comienza con la entrega de la versión beta y prosigue con los prototipos que corrigen los defectos detectados por los usuarios. La fase de Transferencia termina cuando el producto llega a la versión de producción.