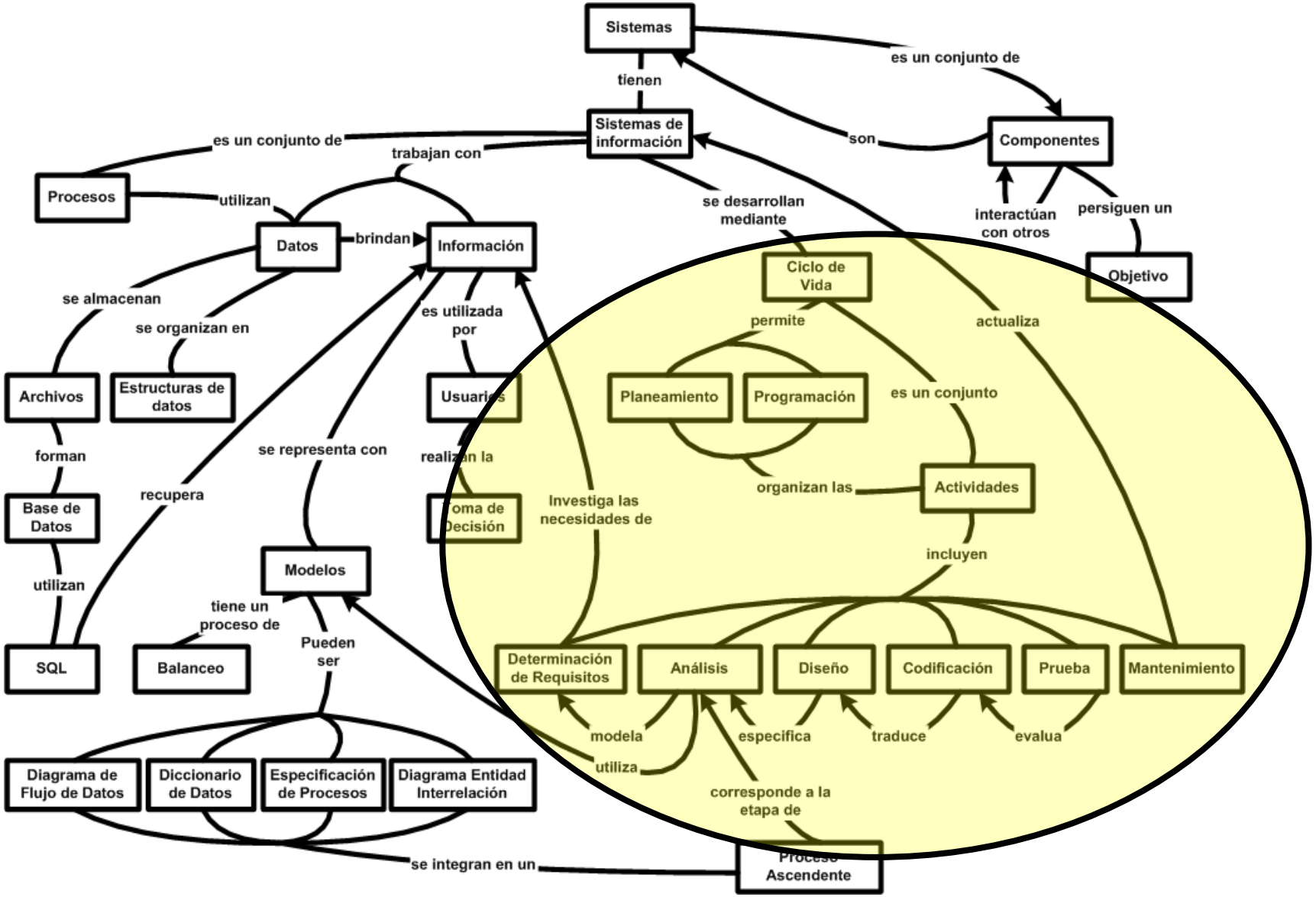
Unidad 5.2

PROCESO DE DESARROLLO DE SOFTWARE

UAI
Universidad Abierta Interamericana

UAIOnline Ultra

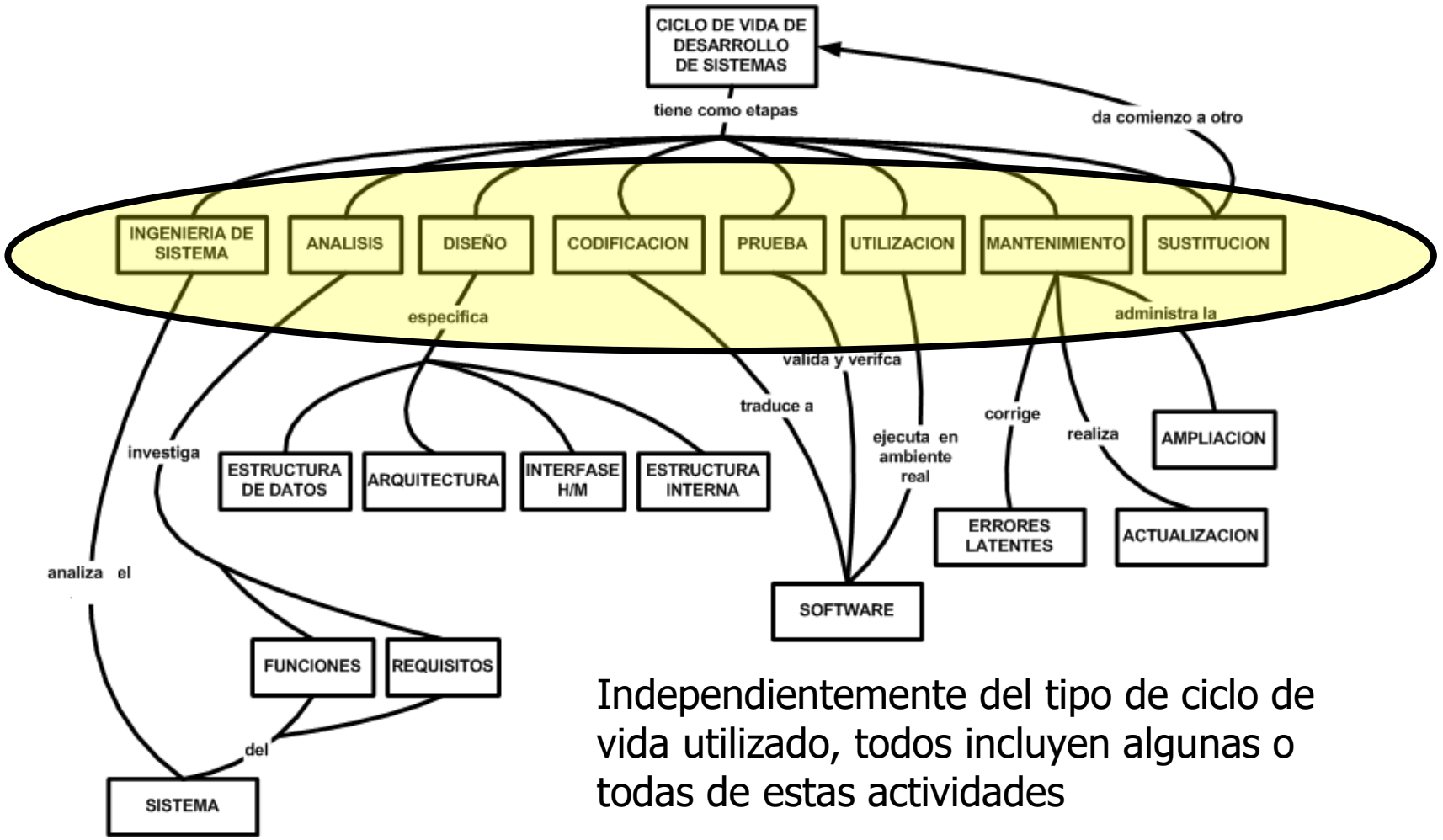

Sistemas
es un conjunto de
tienen
Sistemas de información
son
Componentes
es un conjunto de
trabajan con
Procesos
utilizan
Datos
brindan
Información
se desarrollan mediante
interactúan con otros
persiguen un
Objetivo
Ciclo de Vida
permite
actualiza
se almacenan
se organizan en
es utilizada por
Archivos
Estructuras de datos
Usuarios
Planeamiento
Programación
es un conjunto
forman
se representa con
realizan la
organizan las
Actividades
Base de Datos
recupera
Toma de Decisión
Investiga las necesidades de
Modelos
incluyen
utilizan
tiene un proceso de
Pueden ser
SQL
Balanceo
Determinación de Requisitos
Análisis
Diseño
Codificación
Prueba
Mantenimiento
modela
especifica
traduce
evalua
utiliza
Diagrama de Flujo de Datos
Diccionario de Datos
Especificación de Procesos
Diagrama Entidad Interrelación
corresponde a la etapa de
se integran en un
Proceso Ascendente

Unidad 5.1. Proceso de Desarrollo de Software

PRIMERA PARTE

Ciclo de Vida

CICLO DE VIDA



Independientemente del tipo de ciclo de vida utilizado, todos incluyen algunas o todas de estas actividades

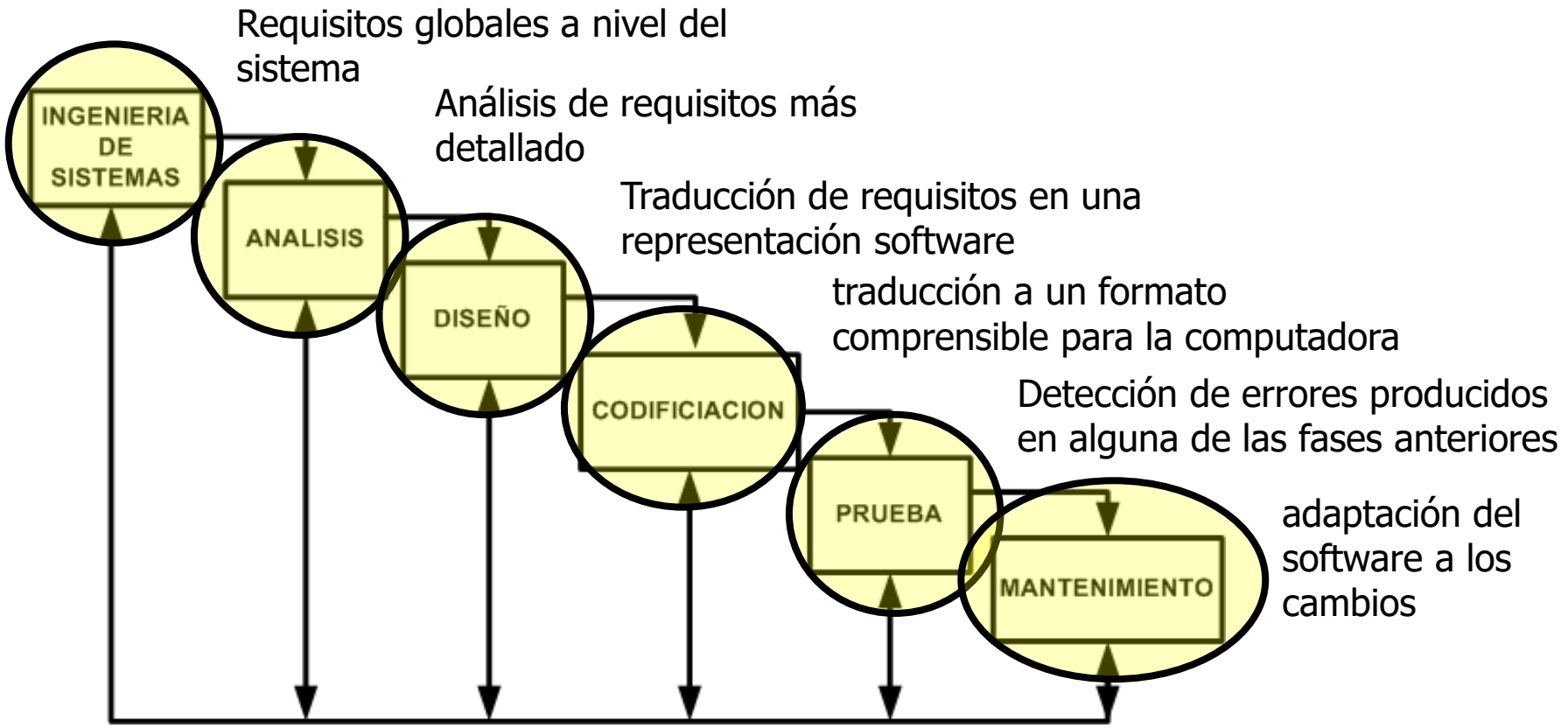
CICLO DE VIDA

- El ciclo de vida de desarrollo de sistemas es una sucesión de etapas por las que atraviesa el software desde que comienza un nuevo proyecto hasta que éste se deja de utilizar

- La elección de un modelo de ciclo de vida se realiza de acuerdo con la naturaleza del proyecto, los métodos a utilizar y los controles y entregas requeridos.

1. Ciclo de vida en cascada

CICLO DE VIDA EN CASCADA



PROBLEMAS DEL CICLO DE VIDA EN CASCADA

- Los proyectos de desarrollo de sistemas, en general, no siguen un ciclo de vida estrictamente secuencial, siempre hay iteraciones

- Dificultad para establecer, **inicialmente**, todos los requisitos del sistema

  - Los requisitos se van aclarando y refinando a lo largo de todo el proyecto

  - Hasta que se llega a la fase final del desarrollo, esto es, la codificación, no se dispone de una versión operativa del programa.

ÁMBITO DE APLICACIÓN DEL CICLO DE VIDA EN CASCADA

- Sistemas simples y pequeños, donde los requisitos sean fácilmente identificables

- Este modelo describe una serie de pasos genéricos que son aplicables a cualquier otro paradigma (espiral, UP)

2. HERRAMIENTAS 4 GL

HERRAMIENTAS 4GL

- Conjunto muy diverso de métodos y herramientas que tiene por objeto facilitar el desarrollo de software
  - Generación de código
  - Generación de pantallas y de informes
  - Gestión de entornos gráficos
  - Herramientas de acceso a bases de datos
- La ventaja principal de estas herramientas es la generación automática de código a partir de especificaciones de alto nivel de abstracción

CICLO DE VIDA 4GL

- El proceso comienza con la determinación de requisitos de información, que pueden ser traducidos directamente a código fuente usando un generador de código.
- El problema que se plantea es el mismo que en el ciclo de vida en cascada, es muy difícil que se puedan establecer todos los requerimientos desde el comienzo
- Si la especificación es pequeña, se puede pasar directamente del análisis de requisitos a la generación automática de código, sin realizar ningún tipo de diseño
- El resto de las fases del ciclo de vida es igual a las del modelo del ciclo de vida en cascada

PROBLEMAS DEL CICLO DE VIDA 4GL

- La mayoría de estas herramientas no logran prescindir totalmente de la codificación.
- Normalmente, el código generado es ineficiente.

ÁMBITO DE APLICACIÓN DEL CICLO DE VIDA 4GL

El ámbito de aplicación de esta técnica está restringido, casi exclusivamente, al software de gestión

- La mayoría de las herramientas de cuarta generación están orientadas a la generación de informes a partir de grandes bases de datos

- Esto es debido a que las mejores prestaciones de estos lenguajes está relacionada con la creación de interfaces hombre - máquina y con las consultas a bases de datos.

3. PROTOTIPOS

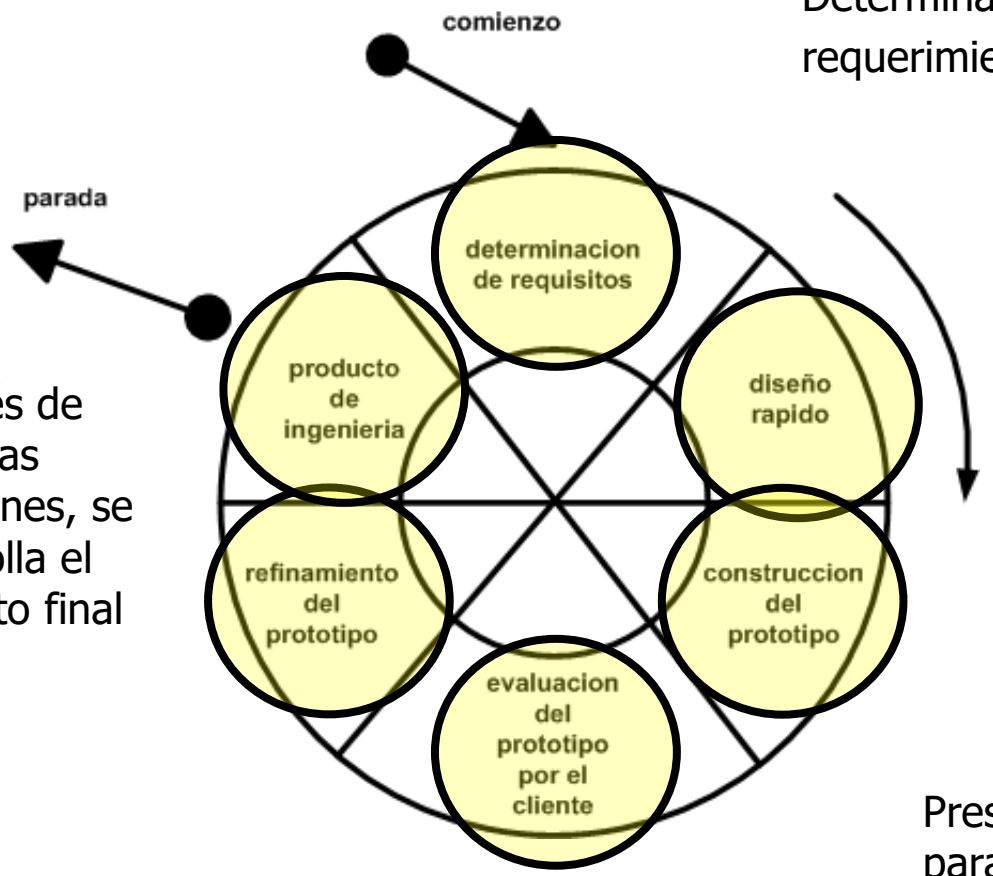
PROTOTIPOS

- En el ciclo de vida en cascada se dificulta la obtención clara de todos los requisitos del sistema al inicio del proyecto
- Un modelo de ciclo de vida basado en la construcción de prototipos puede disminuir estos inconvenientes
- La construcción de un prototipo comienza con la realización de un modelo del sistema, a partir de los requisitos que se conocen
- No es necesario realizar una definición inicial completa de los requisitos del usuario

ES UN PROCESO ITERATIVO E INCREMENTAL

PROTOTIPOS - CONSTRUCCIÓN

Determinación inicial de requerimientos



Diseño rápido, centrado sobre todo en la arquitectura del sistema y la definición de la estructura de las interfaces más que en aspectos procedimentales de los programas

Después de sucesivas iteraciones, se desarrolla el producto final

codificación rápida utilizando 4GL

Presentación al usuario para que lo pruebe y sugiera modificaciones

A partir de comentarios del usuario y los cambios necesarios en los requisitos, se procederá a construir un nuevo prototipo

PROBLEMAS CON LOS PROTOTIPOS

- Uno de los principales problemas con este método es que, con demasiada frecuencia, el prototipo pasa a ser parte del sistema final

- Se olvida aquí que el prototipo ha sido construido de forma acelerada, sin tener en cuenta consideraciones de eficiencia, calidad del software o facilidad de mantenimiento

ÁMBITO DE APLICACIÓN DEL PROTOTIPO

- En general, cualquier aplicación que presente mucha interacción con el usuario, o que necesite algoritmos que puedan construirse de manera evolutiva, yendo de lo más general a lo más específico es una buena candidata

- El prototipo provee una retroalimentación para evaluar y desarrollar nuevos requerimientos

- Cuando el sistema no requiere de especificación de grandes cantidades de detalles algorítmicos, ni de muchas especificaciones de procesos para describir algoritmos con los cuales se obtienen resultados.

4. CICLO DE VIDA EN ESPIRAL

CICLO DE VIDA EN ESPIRAL

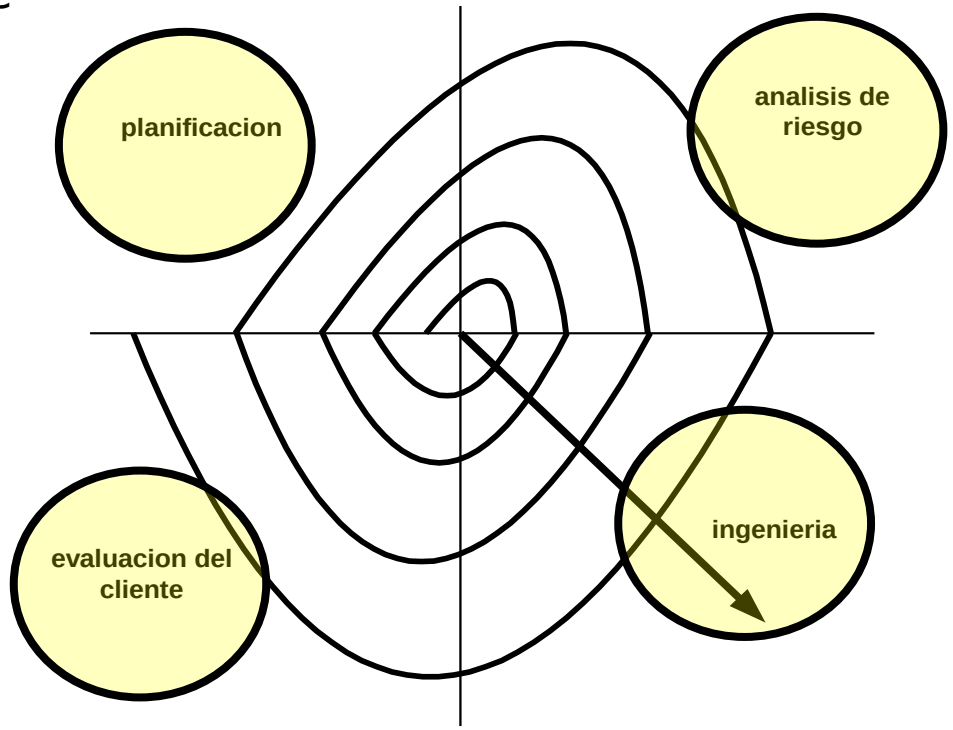
- Importantes proyectos de sistemas fallaron porque los riesgos del proyecto se despreciaron sin estar nadie preparado para algún imprevisto

- Barry Boehm reconoció esto y trató de incorporar el factor de "riesgo del proyecto" al modelo de ciclo de vida, agregándoselo a las mejores características de los modelos de Cascada y de Prototipo

- El modelo en espiral proporciona un modelo evolutivo para el desarrollo de sistemas de software complejos

CICLO DE VIDA EN ESPIRAL - CONSTRUCCIÓN/1

determinación de los objetivos del proyecto, las posibles alternativas y las restricciones.

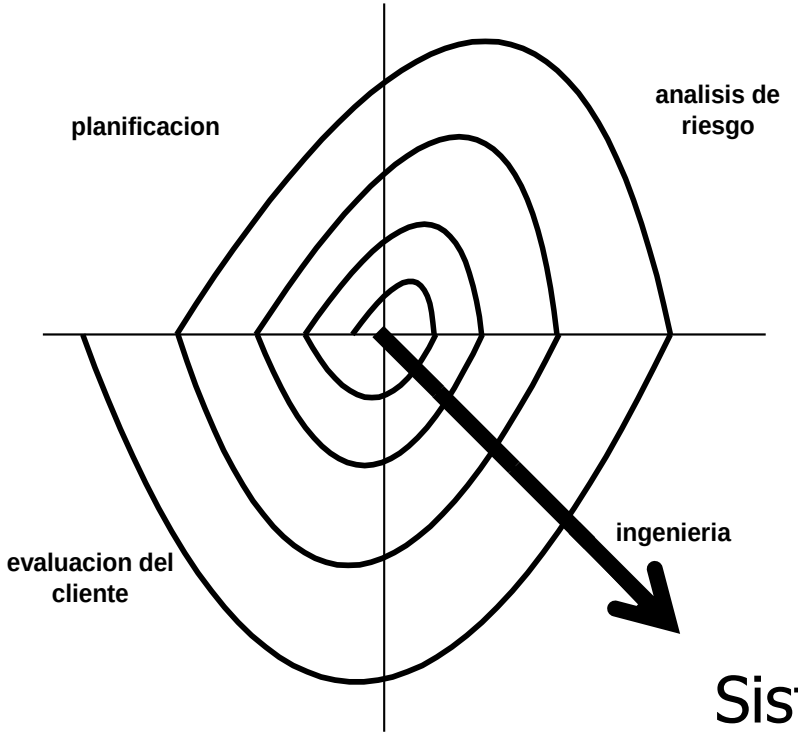


1. Identificación de los riesgos
2. Estimación de los riesgos
3. Evaluación de los riesgos
4. Gestión de riesgos

valoración, por parte del cliente, de los resultados de la ingeniería

Desarrollo del sistema o un prototipo del mismo.

CICLO DE VIDA EN ESPIRAL - CONSTRUCCIÓN/2



Sistema completo

Con cada iteración se construyen sucesivas versiones del software, cada vez más completas. Aumenta la duración de las operaciones del cuadrante de ingeniería, obteniéndose, al final, el sistema completo

PROBLEMAS CICLO DE VIDA EN ESPIRAL

- Falta de un proceso de guía explícito para determinar objetivos, limitaciones y alternativas.
- La determinación del riesgo no es una tarea fácil
- Es necesaria mucha experiencia en proyectos de software

ÁMBITO DE APLICACIÓN
CICLO DE VIDA EN ESPIRAL

- Evita las dificultades de los modelos existentes a través de un proceso conducido por el riesgo, intentando eliminar errores en las fases tempranas.

- Se adapta bien a proyectos complejos, dinámicos e innovadores

EL MANTENIMIENTO EN EL SOFTWARE

- Terminada la fase de pruebas, el software se entrega al cliente y comienza la vida útil del mismo

- El software sufrirá cambios a lo largo de su vida útil. Estos cambios pueden ser debidos a variadas causas.

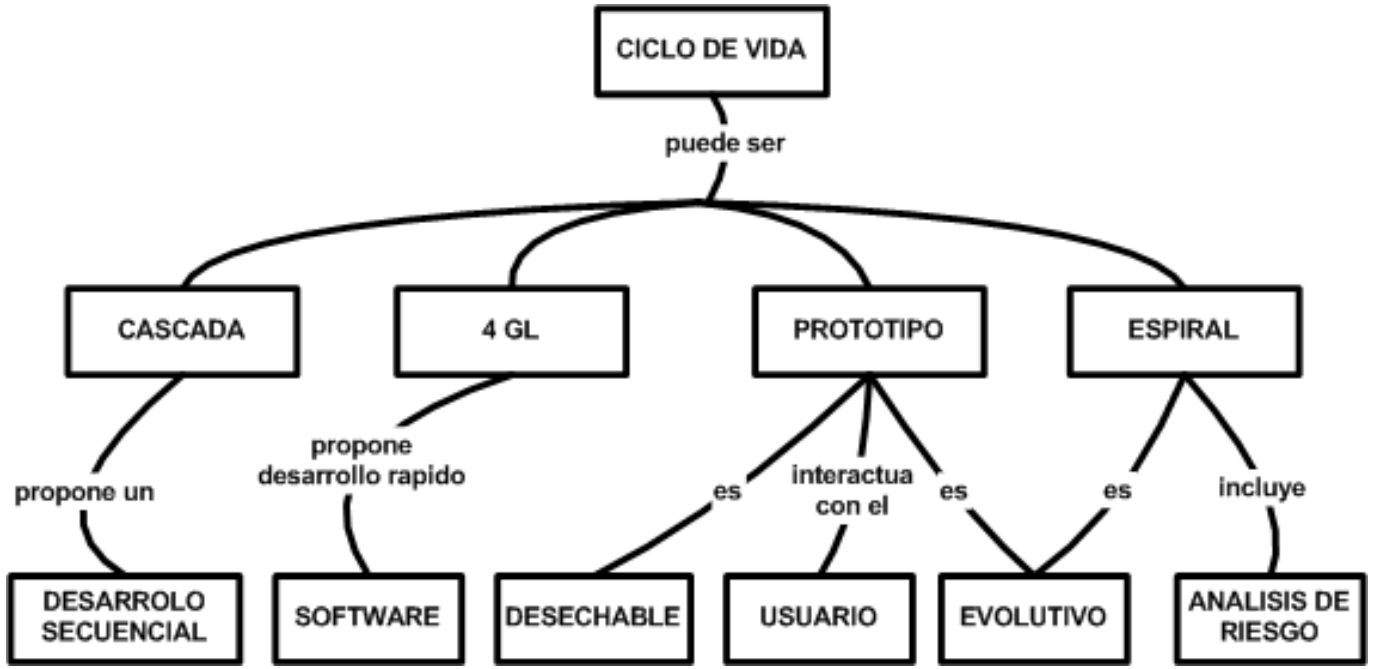
    1. Durante la utilización, el cliente puede detectar errores en el software, estos se denominan errores latentes

    2. Cuando se producen cambios en alguno de los componentes del sistema informático, por ejemplo cambios en la computadora, en el sistema operativo o en los periféricos, se debe adaptar el software a ellos.

    3. El cliente habitualmente requiere modificaciones funcionales, normalmente ampliaciones, que no fueron contempladas inicialmente en el proyecto.

- En cualquier caso, el mantenimiento supone volver atrás en el ciclo de vida, a las etapas de codificación, diseño o análisis dependiendo de la magnitud del cambio encarado.

CICLO DE VIDA - RESUMEN



SEGUNDA PARTE

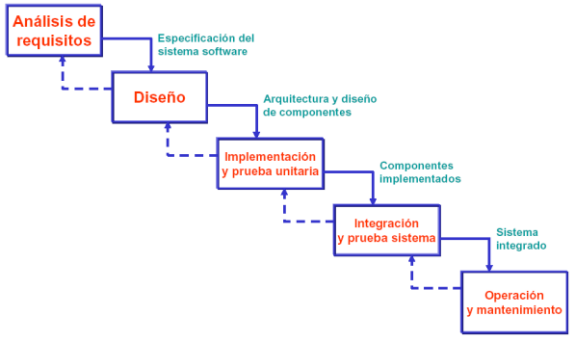
Proceso de Unificado de desarrollo

PROCESO UNIFICADO DE DESARROLLO

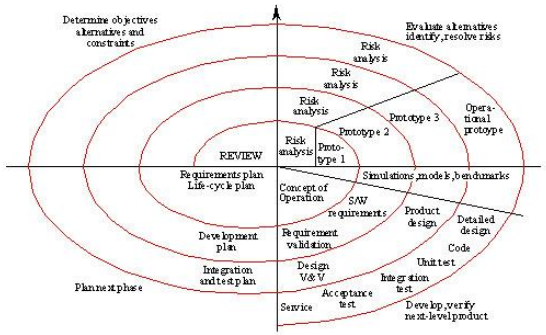
UML no es un proceso...

...UML es una notación

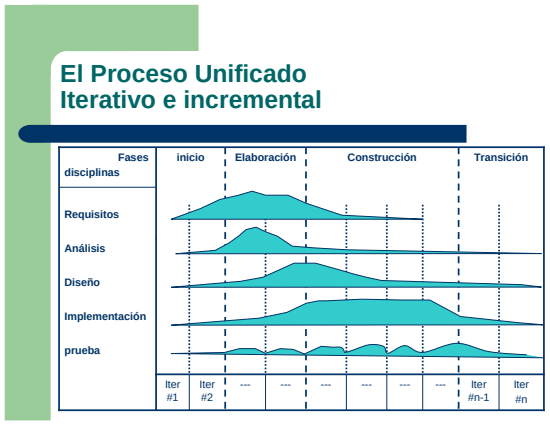
Por lo tanto precisa de un proceso de desarrollo (ciclo de vida) que especifique la secuencia de actividades que deben realizarse



Ciclo de vida en cascada



Ciclo de vida en espiral



Proceso Unificado

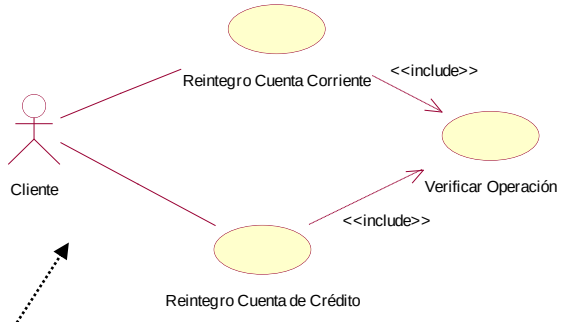
PROCESO UNIFICADO (PU)

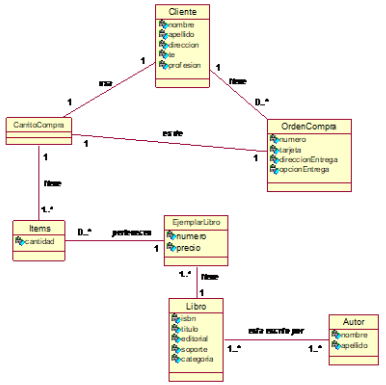
Proceso de Desarrollo de Software

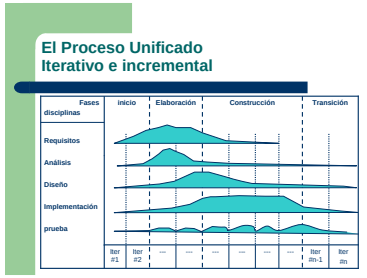
Conjunto de actividades para transformar los requisitos del usuario en un sistema software

Proceso Unificado (UP)

- *Dirigido por Casos de Uso*
- *Centrado en la Arquitectura*
- *Iterativo e Incremental*

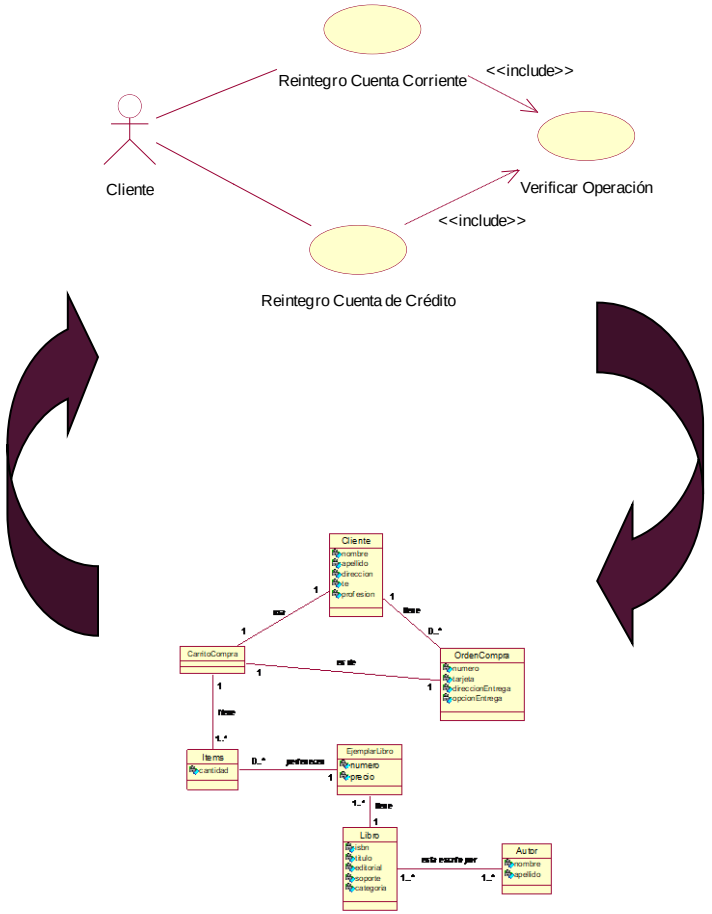




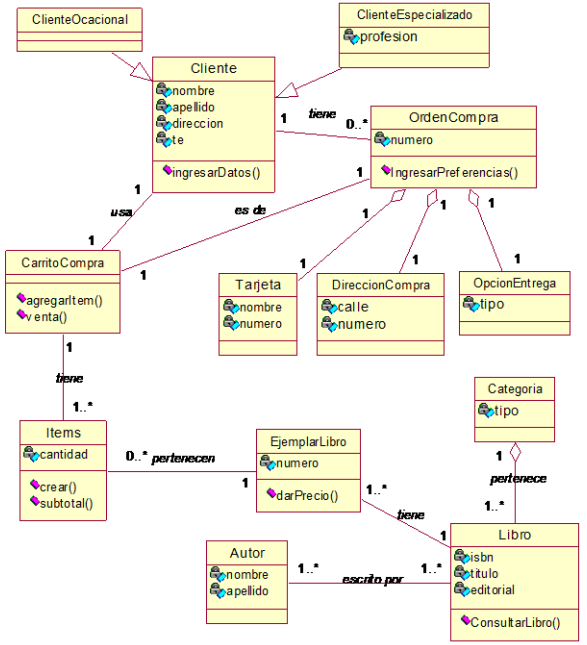


- *Los casos de uso representan los requisitos funcionales y guían el diseño, la implementación y la prueba*

- *Basándose en los casos de uso los desarrollares crean modelos de diseño e implementación que llevan a cabo los casos de uso*



- *La arquitectura es una vista de diseño con las características más importantes, dejando los detalles de lado. Describe diferentes vistas del sistema*
- *Constituyen la "forma del sistema", incluye aspectos estáticos y dinámicos*
- *La arquitectura y los casos de uso evolucionan en paralelo*



ITERATIVO E INCREMENTAL/1

- *Desarrollo iterativo:*

- *El desarrollo se organiza en una serie de mini proyectos de **duración fija**, llamados **iteraciones** (2 a 6 semanas)*

- *El resultado de cada uno es un sistema que puede ser **probado**, **integrado** y **ejecutado**.*

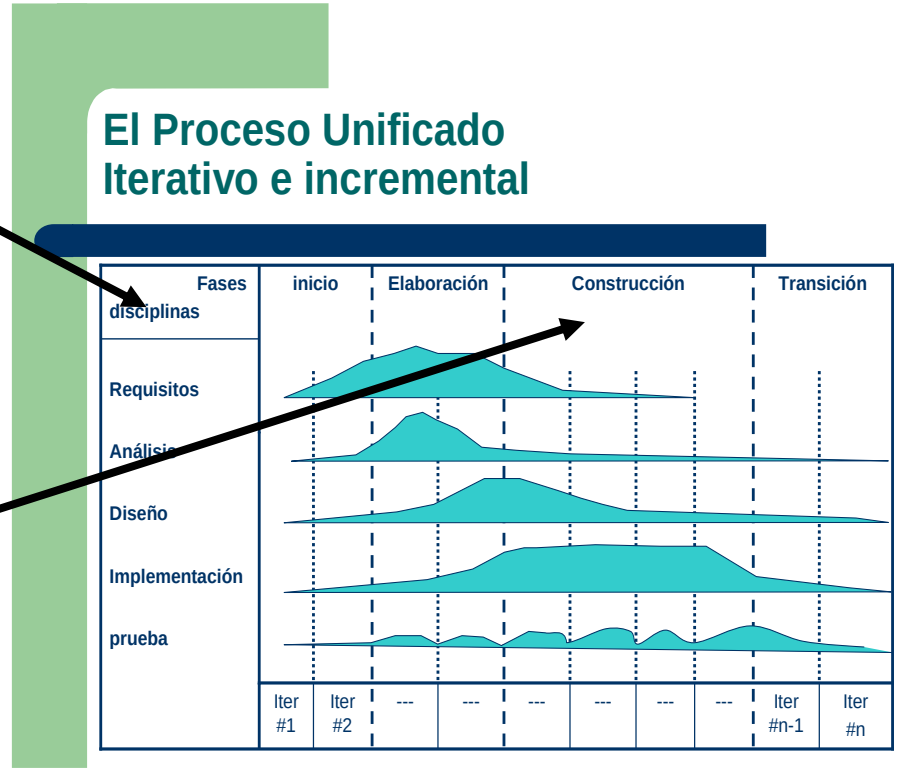
- *Cada iteración incluye sus propias actividades de: análisis de requisitos, diseño, implementación y prueba*

ITERATIVO E INCREMENTAL/2

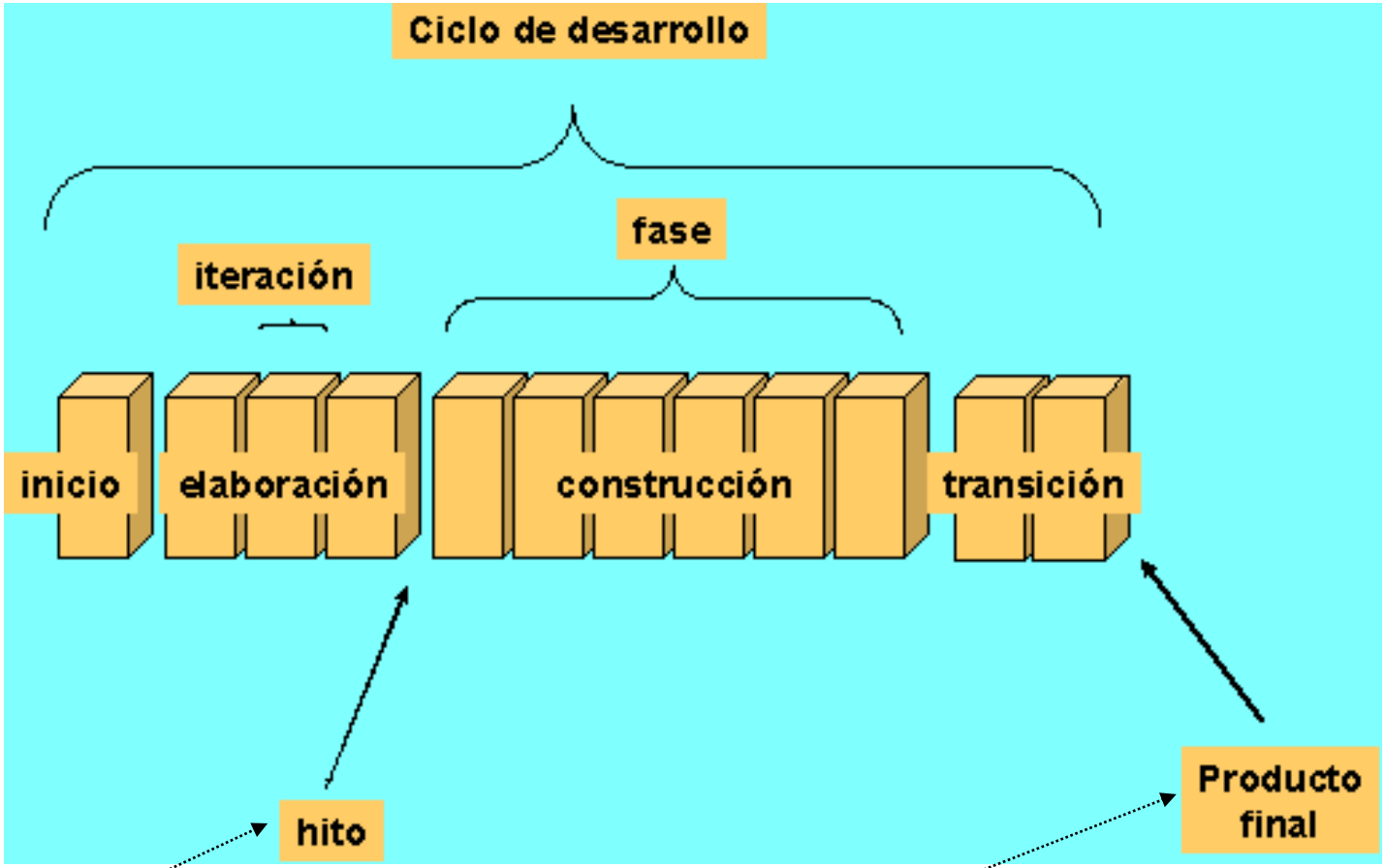
- *El sistema crece incrementalmente a lo largo del tiempo, iteración tras iteración.*
- *El resultado de cada iteración es un sistema* ***ejecutable****, pero incompleto*
- *En general, cada iteración aborda nuevos requisitos y amplia el sistema incrementalmente*
- *La salida de una iteración NO es un prototipo desechable, es un subconjunto de calidad del sistema final*

***Disciplina**: conjunto de actividades y artefactos vinculados en una área determinada: requisitos, diseño, implementación, etc.*

***Fases**: periodo de tiempo entre dos hitos principales de un proceso de desarrollo*



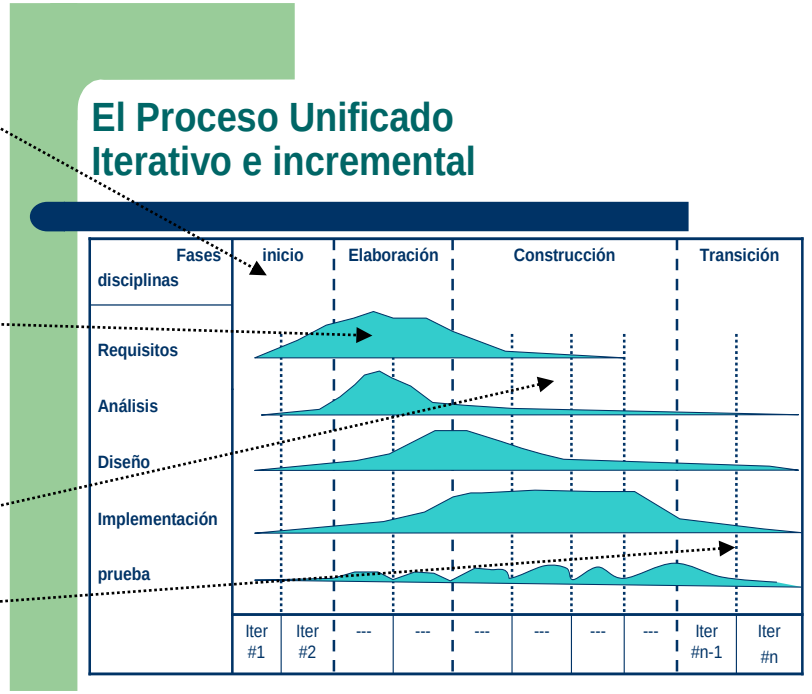
Ciclo de desarrollo



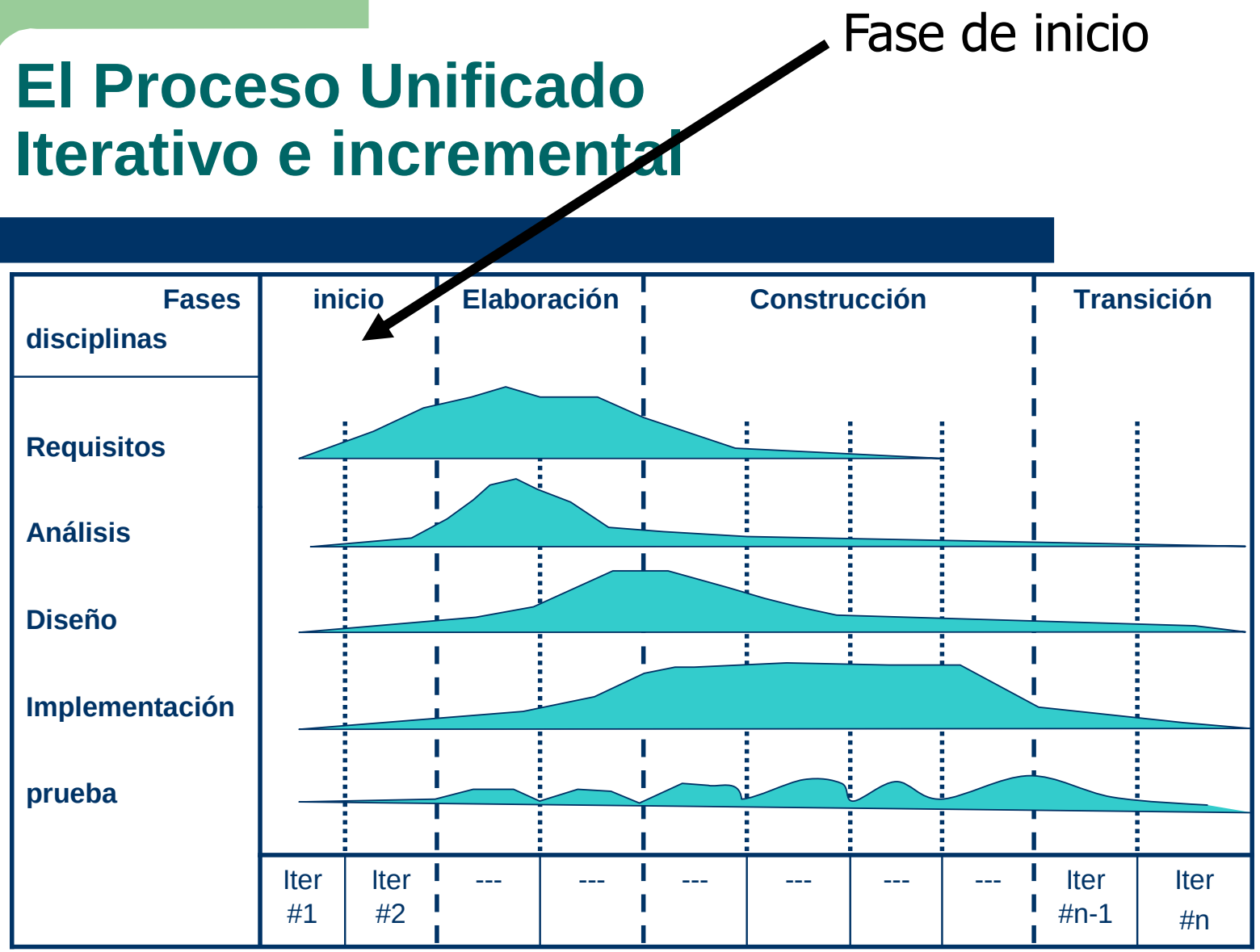
punto en donde han de tomarse decisiones importantes

Sistema en ambiente de producción

- ***Inicio***: *visión aproximada, incluye: análisis del negocio, alcance, estimaciones imprecisas*
- ***Elaboración***: *visión refinada, implementación iterativa del núcleo central de la arquitectura, resolución de riesgos altos, identificación de más requisitos*
- ***Construcción***: *implementación iterativa del resto de requisitos de menor riesgo*
- ***Transición***: *pruebas beta*



El Proceso Unificado Iterativo e incremental



FASE DE INICIO/ I

Actividades a realizar (una o algunas)

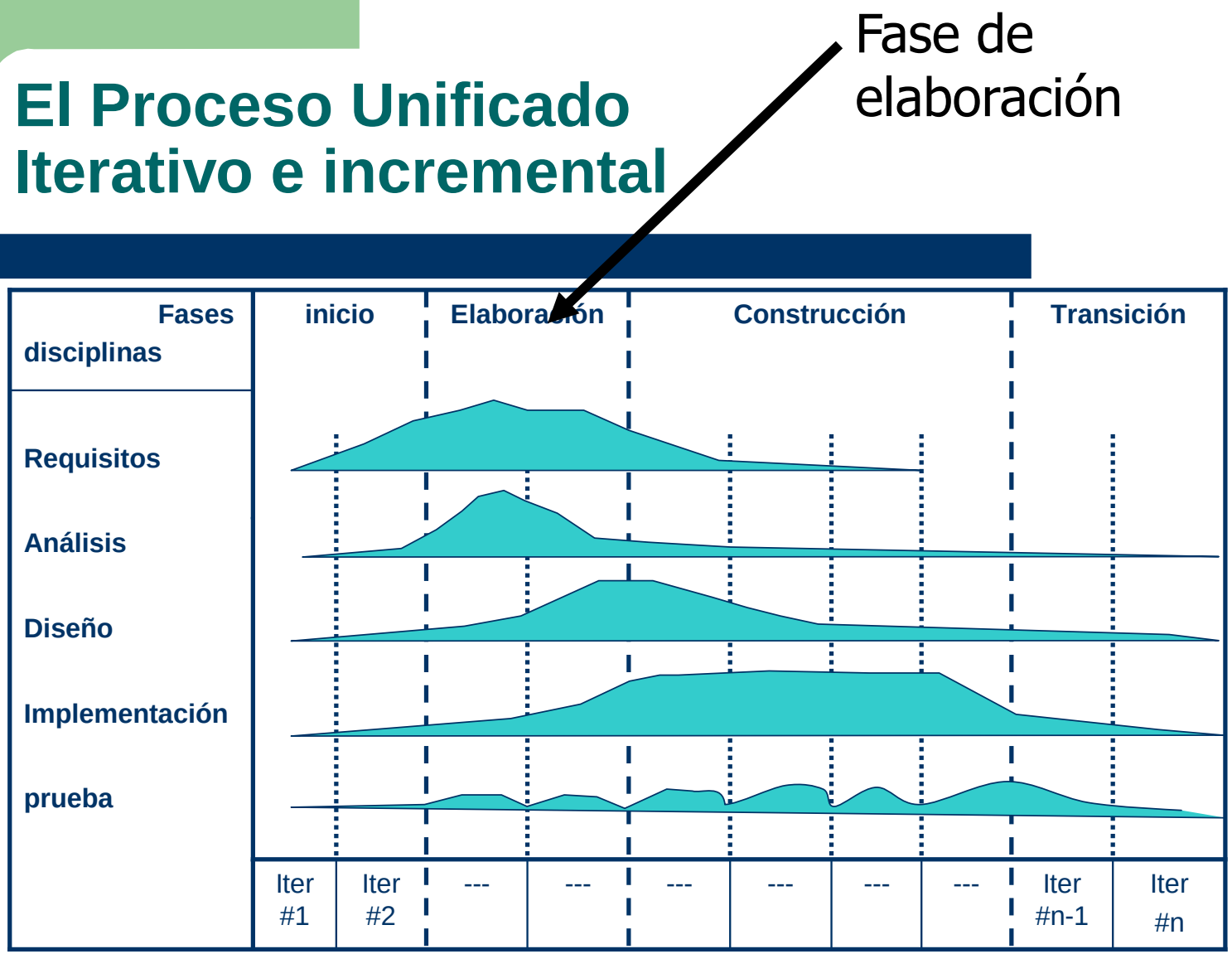
- ***Visión y análisis del negocio*** *(objetivos y restricciones de alto nivel)*
- ***Modelo de casos de uso*** *(requisitos funcionales y no funcionales relacionados)*
- ***Especificaciones complementarias*** *(otros requisitos)*
- ***Listas de riesgos*** *(del negocio, técnicos, etc.)*
- ***Prototipos*** *(para clarificar la visión)*
- ***Plan de iteración*** *(qué hacer en la primera iteración)*

FASE DE INICIO/2

***No** se entendió la fase de inicio si...*

- *La duración es mayor a unas pocas semanas*
- *Se intenta definir la mayoría de los requisitos*
- *Se espera que los planes y la estimación sea fiable*
- *Se define la arquitectura*
- *No se identifican la mayoría de de los nombres de los casos de uso y los actores*
- *Se escribieron todos los casos de uso en detalle*

El Proceso Unificado Iterativo e incremental



FASE DE ELABORACIÓN/1

Actividades a realizar

- *Se descubren y estabilizan la mayoría de los casos de uso*
- *Se reducen o eliminan los riesgos importantes*
- *Se implementan y prueban los elementos básicos de la arquitectura*
- *Duración: 2 a 4 iteraciones con una duración de 2 a 6 semanas (dependiendo de la duración del proyecto)*

FASE DE ELABORACIÓN/2

Artefactos de la fase de elaboración

- *Modelo del dominio (entidades del dominio)*
- *Modelo de diseño (diagramas de clases, de iteración. etc)*
- *Modelo de datos*
- *Modelo de pruebas*
- *Modelos de implementación*

*El producto resultante **no es** un prototipo desechable*

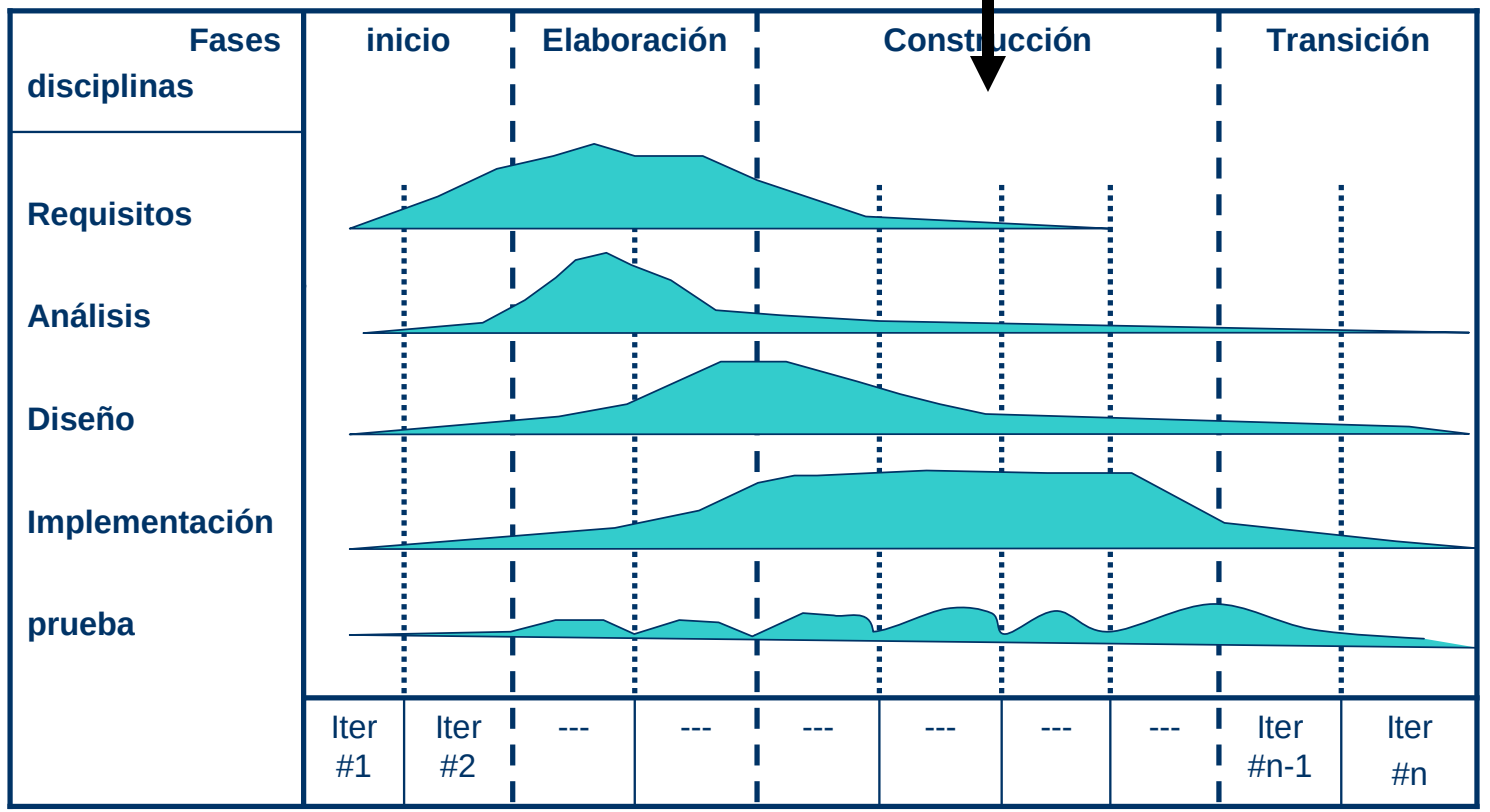
El código y el diseño son de calidad y se integran al sistema final

FASE DE ELABORACIÓN/3

***No** se entendió la fase de elaboración si...*

- *Sólo comprende una iteración*
- *La mayoría de los requisitos se definieron antes de la elaboración*
- ***NO** hay programación de código ejecutable*
- *Se intenta llevar a cabo un diseño completo antes de la codificación*
- *Los usuarios no se involucran en la evaluación*

El Proceso Unificado Iterativo e incremental

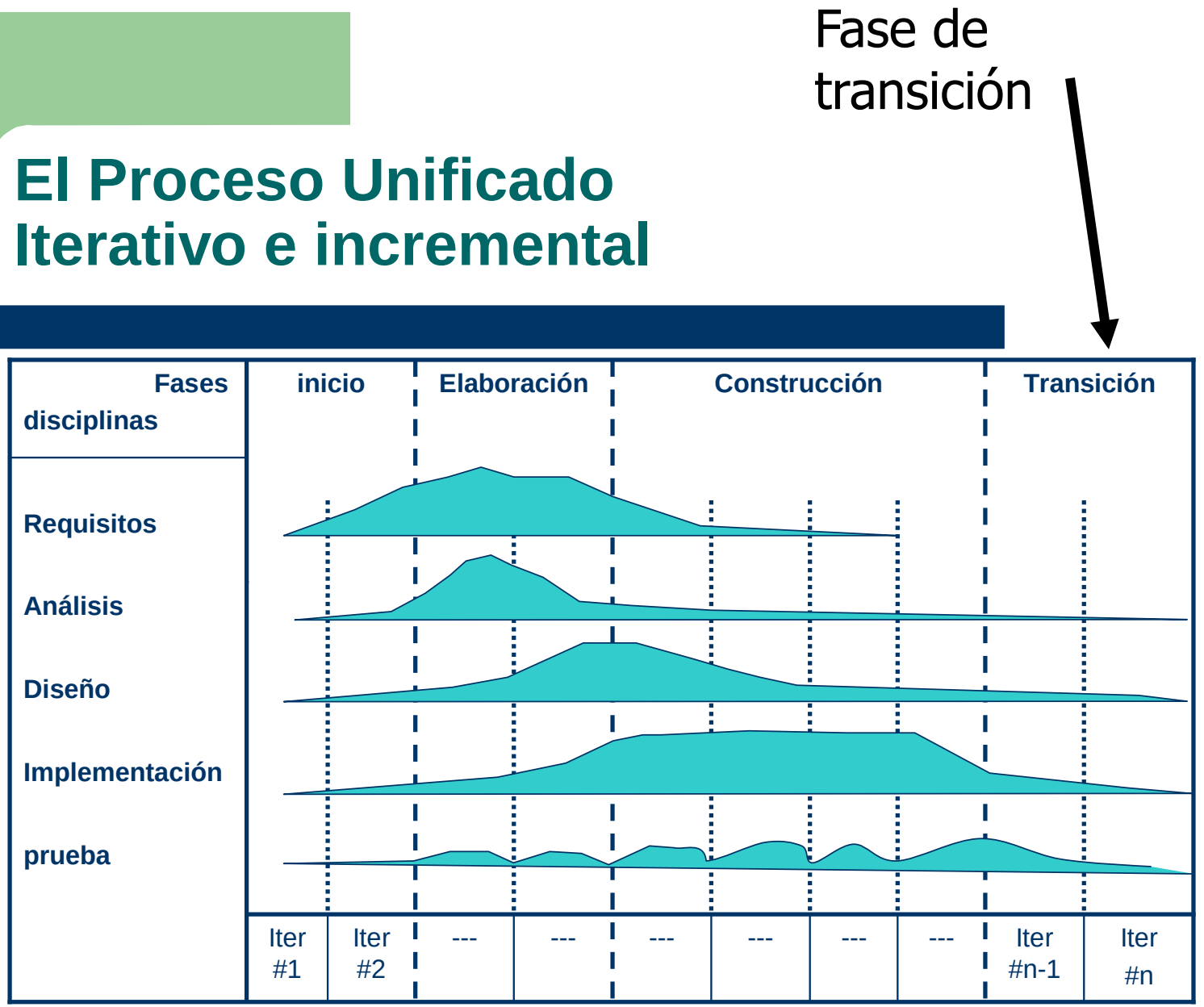


FASE DE CONSTRUCCIÓN

- *Objetivos*

  - *Terminar de construir la aplicación*
  - *Realizar pruebas alfa*
  - *Preparar pruebas beta (para la transición)*
  - *Preparación de guías de usuario*
  - *Preparación de materiales de aprendizaje*

El Proceso Unificado Iterativo e incremental



FASE DE TRANSICIÓN

Objetivo: *poner el sistema en producción*

Actividades

- *Realización de pruebas beta*
- *Reaccionar a la retroalimentación a partir de las pruebas beta*
- *Conversión de datos*
- *Cursos de entrenamiento*

SUGERENCIAS

- Recuerde que cada fase, si bien pasa por todas las disciplinas (análisis, diseño, etc.) pone énfasis en algunas de ellas más que en otras
- Valore la importancia de los casos de uso (unidad 3.1). A partir de ellos se crean todos los demás modelos del sistema
- Tenga en cuenta que después de cada iteración (que corresponde aproximadamente a un ciclo de vida en cascada) se va definiendo un sistema ejecutable, pero incompleto, que debe corroborar con el usuario
- Recuerde que inicio **no** es igual a requisitos, elaboración **no** es igual a análisis y construcción **no** es igual a diseño, etc
- Recuerde que cada iteración tiene un duración fija de 2 a 6 semanas aproximadamente dependiendo de la magnitud del proyecto

AUTO EVALUACIÓN PARTE I

Comprendí los conceptos más importantes de la unidad si puedo definir y dar ejemplos de:

- Ciclo de vida
- Análisis
- Diseño
- Codificación
- Prueba
- Mantenimiento (distintos tipos)

AUTO EVALUACIÓN PARTE 1

Comprendí los conceptos más importantes de la unidad 4.1 si:

- Identifico los características principales, problemas y ámbitos de aplicación de los distintos ciclos de vida presentados

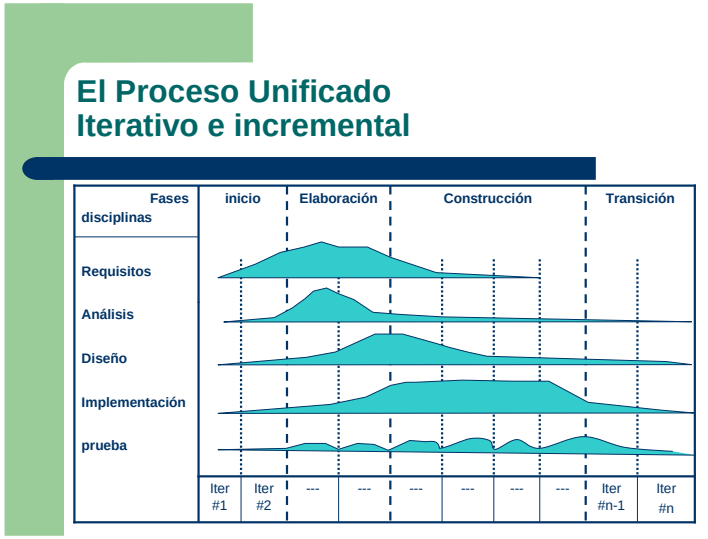
- Entiendo como relacionar entre si:
  - El ciclo de vida en cascada con el de prototipos
  - El ciclo de vida de prototipos con 4GL
  - El ciclo de vida es espiral con el de prototipos y cascada

AUTO EVALUACIÓN PARTE 2

- Comprendí los conceptos más importantes de la unidad si
- Conozco por qué usamos modelos y no directamente codificamos
- Entiendo en qué etapas del proceso de desarrollo utilizo UML
- Entiendo la diferencia entre el ciclo de vida en cascada y el UP
- Comprendo la relación que existe entre disciplinas de UP y el ciclo de vida en cascada
- Entiendo qué tienen en común el ciclo de vida en espiral y el UP
- Comprendo en qué hacen énfasis cada una de las fases del UP
- Entiendo cuál es el producto final de cada iteración

Comprendí los conceptos más importantes de la unidad si puedo definir y dar ejemplos de:

- Disciplinas / Fases
- Fase de inicio
- Fase de elaboración
- Fase de construcción
- Fase de transición
- Desarrollo iterativo e incremental
- Interpreto en el gráfico que significan las “montañitas”



TERCERA PARTE

Metodologías ágiles

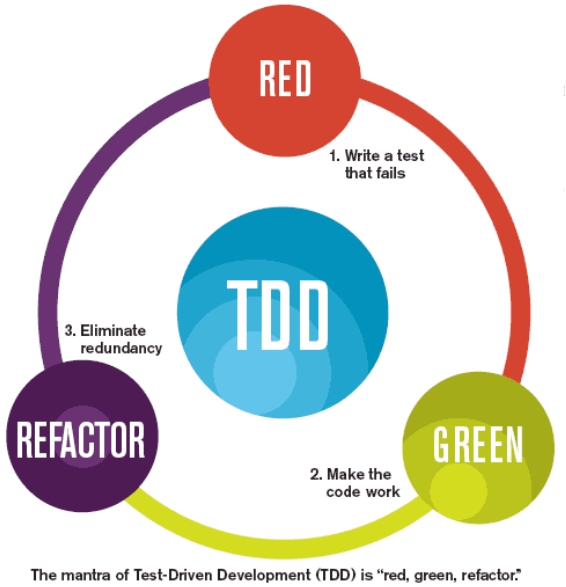
Desarrollo guiado por Test

¿CÓMO FUNCIONA?

Ciclo de Vida R-G-R

Red-Green-Refactor

- Escritura de código (Baby Steps)
- Obtener ROJO
- Escritura de código
- Obtener VERDE
- Refactor



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

Este es el ciclo de vida del desarrollo guiado por test, Rojo-Verde-Refactor y se debería repetir esto una y otra vez hasta que tengamos las características funcionales y completas.

CARACTERÍSTICAS

- La obtención de un buen resultado depende de la calidad de las pruebas.
- No solo se basa en pruebas, depende también del refactor del código (mejor código, limpio, mantenible).
- El refactor solo se aplica si es necesario.
- El test es escrito por el desarrollador. (Visión desde su punto de vista)
- El foco principal de TDD es testar las funcionalidad de bajo nivel

VENTAJAS

¿Por qué es mejor hacer las pruebas antes qué el código?

- Se condicionan las pruebas a lo implementado: pueden obviarse casos de prueba.
- Se realiza un profundo análisis funcional y se refinan los requisitos.
- Se implementa solamente código necesario:
  - Baja redundancia
  - Sin código muerto (“por si acaso”)
  - Código limpio

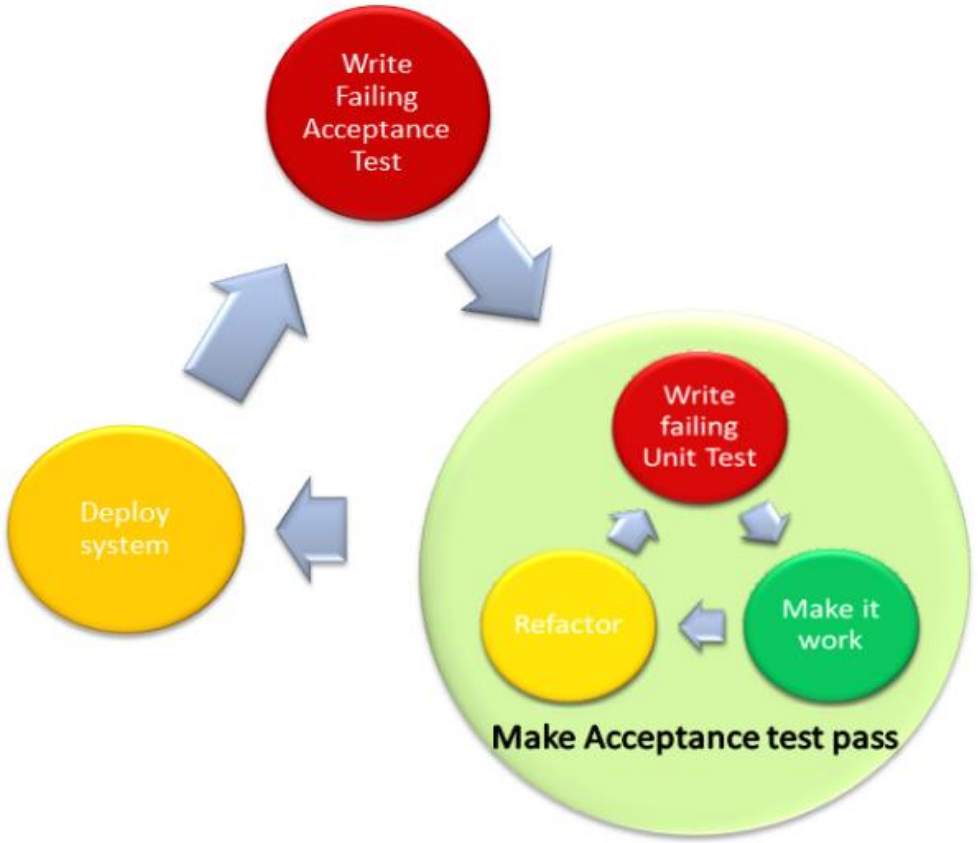
BUENAS PRÁCTICAS

- Tener el código separado de los tests, en carpetas diferentes.
- Los tests deben fallar la primera vez que son escritos.
- Los nombre de los tests deben ir acorde con la intención, deben ser nombres expresivos.
- Refactorizar para eliminar código duplicado después de pasar los tests.
- Repetir las pruebas después de cada refactorización.
- Solo se debe escribir nuevo código, cuando algún test falla. Cada test debe comprar un nuevo comportamiento, o diferente.
- Escribe primero el *assert*.
- Todos los tests deben ser pasados antes de escribir otro test.
- Solo se refactoriza cuando todos los tests pasan.
- Escribe el mínimo y simple código para pasar las pruebas.
- No usar las dependencias entre tests. Los tests deben pasar en cualquier orden.
- Los tests deben ser rápidos.
- Crear solo un test por iteracion

BDD Y ATDD

- **ATDD (Acceptanec Test Driven Develpment)**
  - **Los criterios de aceptación son definidos antes del proceso de desarrollo y van a guiar al proceso subsecuente.**
  - **Ejercicio colaborativo que involucra al product owner, analistas de negocio, testers y desaarrolladores, y ayuda a asegurar que todos los miembros del proyecto entendieron que se necesita para que el proyecto este correcto**
  - **los equipos crean uno o mas test de nivel de aceptación para una característica antes de comenzar a trabaja**
  - **Los test unitarios aseguran que la aplicación se construyo correctamente**
  - **Los test de aceptación aseguran que se ha desarrollado la funcionalidad esperada**

ATDD


Write Failing Acceptance Test
Write failing Unit Test
Make it work
Refactor
Make Acceptance test pass
Deploy system

Fin de la clase

UAI
Universidad Abierta Interamericana