Procesos iterativos incrementales

IIC2143: Ingeniería de Software

Cinco modelos de procesos principales

Cascada [W. Royce, 1970]

(**Mejoramiento Iterativo** [V. Basili & A. Turner, 1975])

Espiral [B. Boehm, 1986]

RUP [I. Jacobson, G. Booch & J. Rumbaugh, 1996] --> OpenUP [IBM, 2006]

Ágil, implementado por diversos métodos:

- **Scrum** [K. Schwaber & J. Sutherland, 1996]
- **XP** [K. Beck, 1999]
- **Kanban** [D. Anderson, 2010]

Prototipos

Reconoce el desajuste entre un sistema de software recién construido y las expectativas de los usuarios

…

…

Como solución, construye un prototipo del sistema y lo usa para obtener y validar requisitos:

- también para hacer estudios de factibilidad o para validar un diseño
- pueden ser simples —muestra solo el *look and feel* y una secuencia de pantallas— o sofisticados —puede llegar a implementar varias de las funciones del sistema
- pueden ser desechables —construidos de manera rápida y económica, y posiblemente reusados para capacitar a los usuarios o incluso en pruebas unitarias o de integración—
- … o evolutivos

Prototipos evolutivos

Los prototipos desechables a veces implican que se desperdicie mucho esfuerzo:

- prototipos sofisticados para estudios de factibilidad o validación de diseños

…

…

Como solución, los prototipos evolutivos (naturalmente) evolucionan:

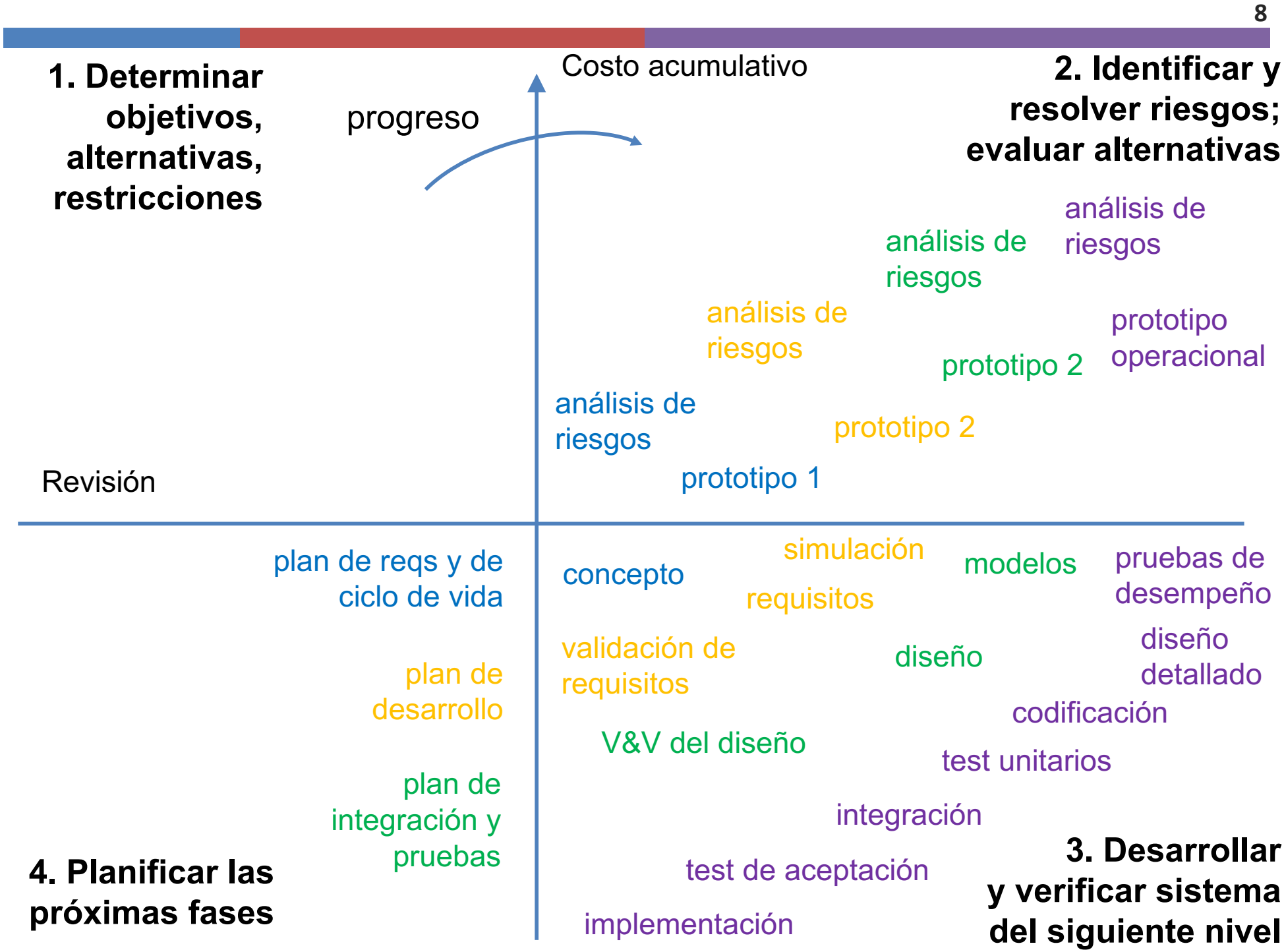
- los usuarios experimentan con una versión inicial, según un conjunto de requisitos preliminares
- la realimentación de los usuarios se usa repetidamente para ir mejorando y completando el prototipo
- como no se va a desechar, se lo construye tanto con funcionalidad operacional como de manera robusta
- son más apropiados para proyectos de tipo exploratorio, en que los requisitos y algoritmos van siendo descubiertos a medida que se trabaja con el sistema

El modelo de desarrollo espiral

Incluye explícitamente gestión de riesgos

Cada ciclo (actividades de un mismo color en tres cuadrantes) apunta a mejorar un determinado aspecto —el sistema evoluciona incremental-mente:

- cuadrante 1: Identifica y prioriza los riesgos del proyecto
- cuadrante 2: Evaluación de alternativas; análisis y resolución de los riesgos de no lograr los objetivos; puede ser seguido por 4 (si quedan riesgos importantes), 3 (si los riesgos principales han sido resueltos), 2 a la derecha (si ya tenemos un prototipo operacional robusto)
- cuadrante 3: Procede como el modelo de cascada
- cuadrante 4: Define requisitos, actividades del ciclo de vida, y el plan de integración y pruebas para la siguiente fase


1. Determinar objetivos, alternativas, restricciones
Costo acumulativo
progreso
2. Identificar y resolver riesgos; evaluar alternativas
análisis de riesgos
análisis de riesgos
prototipo operacional
análisis de riesgos
prototipo 2
análisis de riesgos
prototipo 2
prototipo 1
Revisión
plan de reqs y de ciclo de vida
simulación
modelos
pruebas de desempeño
concepto
requisitos
validación de requisitos
diseño
diseño detallado
plan de desarrollo
codificación
V&V del diseño
test unitarios
plan de integración y pruebas
integración
test de aceptación
implementación
4. Planificar las próximas fases
3. Desarrollar y verificar sistema del siguiente nivel

Aclaraciones

No es una secuencia de incrementos del modelo de cascada

No todo lo que se hace en un proyecto sigue una única secuencia espiral

No es necesario que todos los elementos del modelo, tal como aparecen en el diagrama, sean “visitados” en el orden indicado

Es válido “retroceder” y volver a plantearse decisiones ya tomadas

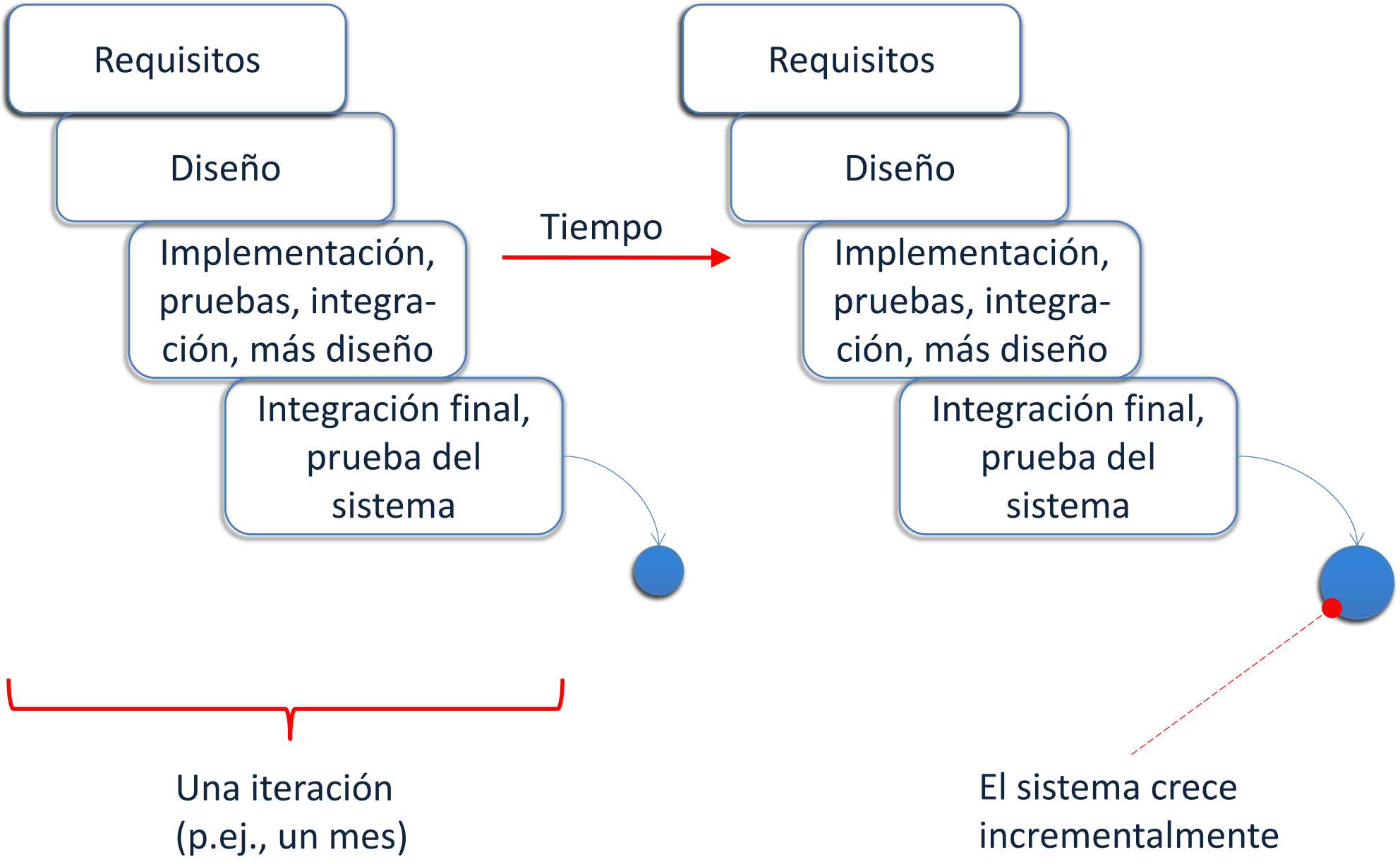
El desarrollo iterativo es una práctica clave en los procesos modernos

El desarrollo es organizado en una serie de mini proyectos cortos — **iteraciones**

Cada iteración incluye sus propias actividades de análisis de requisitos, diseño, implementación y testing

El resultado de cada iteración es un *sistema parcial*:

- puede ser ejecutado, está validado e integrado
- **no es** un prototipo experimental o desechable


Requisitos
Diseño
Implementación, pruebas, integra-ción, más diseño
Integración final, prueba del sistema
Tiempo
Requisitos
Diseño
Implementación, pruebas, integra-ción, más diseño
Integración final, prueba del sistema
Una iteración (p.ej., un mes)
El sistema crece incrementalmente

El desarrollo iterativo (incremental y evolutivo) se contrapone al ciclo de vida de cascada

Incluye programación y testing desde temprano de un sistema parcial, en ciclos que se repiten

El desarrollo mismo comienza antes de que todos los requisitos estén definidos en detalle

Emplea realimentación para clarificar y mejorar especificaciones que van siendo desarrolladas gradualmente

En cada iteración elegimos un subconjunto pequeño de requisitos …

… y diseñamos, implementamos y verificamos un subsistema que responda a esos requisitos:

- los requisitos y el diseño de las iteraciones iniciales pueden no ser exactamente lo que finalmente necesitamos
- … pero el dar pasos pequeños rápidamente permite realimentación igualmente rápida de los usuarios, desarrolladores y a partir de las pruebas
- en lugar de especular sobre los requisitos o el diseño, se aprovecha la realimentación proveniente de construir y probar de manera realista
- la actitud clave es acoger el cambio y adaptarse

Cambio y adaptación son inevitables y hasta esenciales

… con los beneficios enumerados en la próxima diapositiva

Gestión de riesgos:

- podemos validar/invalidar requisitos y suposiciones de diseño implementando incrementalmente partes del sistema, comenzando por las más riesgosas

Economía:

- podemos revisar las prioridades frecuentemente y tratar las decisiones como opciones —negocios inciertos— que se vuelven más valiosas si hay más flexibilidad gracias a resultados tempranos y verificaciones frecuentes

Foco:

- al agrupar el trabajo en iteraciones cortas, los integrantes del equipo se enfocan mejor en ese trabajo

Motivación:

- es motivante para un equipo de software ver entregas tempranas funcionando

Control:

- las iteraciones cortas ayudan a reducir el error de las estimaciones y dan realimentación rápida sobre los planes

Involucramiento de interesados:

- clientes, usuarios, etc., ven resultados antes y se involucran más, dedicando más tiempo, perspicacia y recursos

Aprendizaje continuo:

- el equipo aprende acerca del producto en cada iteración

El RUP combina varias prácticas eficaces de la ingeniería de software moderna

Idea central:

- desarrollo iterativo adaptable —iteraciones cortas de duración fija

Iteraciones iniciales:

- abordamos aspectos de alto riesgo y valor
- construimos una arquitectura central cohesiva

Prácticas continuas:

- involucramos a los usuarios en la evaluación, realimentación y descubrimiento de requisitos
- verificamos la calidad, validando desde un comienzo, a menudo y de manera realista
- administramos los requistos, los cambios y la configuración

También:

- aplicamos casos de uso —para encontrar y describir los requsitos funcionales
- modelamos el software visualmente —con UML

El RUP organiza el trabajo y las iteraciones en cuatro *fases*

1) **Inicio**:

- Visión aproximada, caso de negocio, alcance, estimaciones preliminares

2) **Elaboración**:

- Visión refinada, implementación iterativa de la arquitectura central, resolución de los mayores riesgos, identificación de la mayoría de los requisitos, estimaciones más realistas

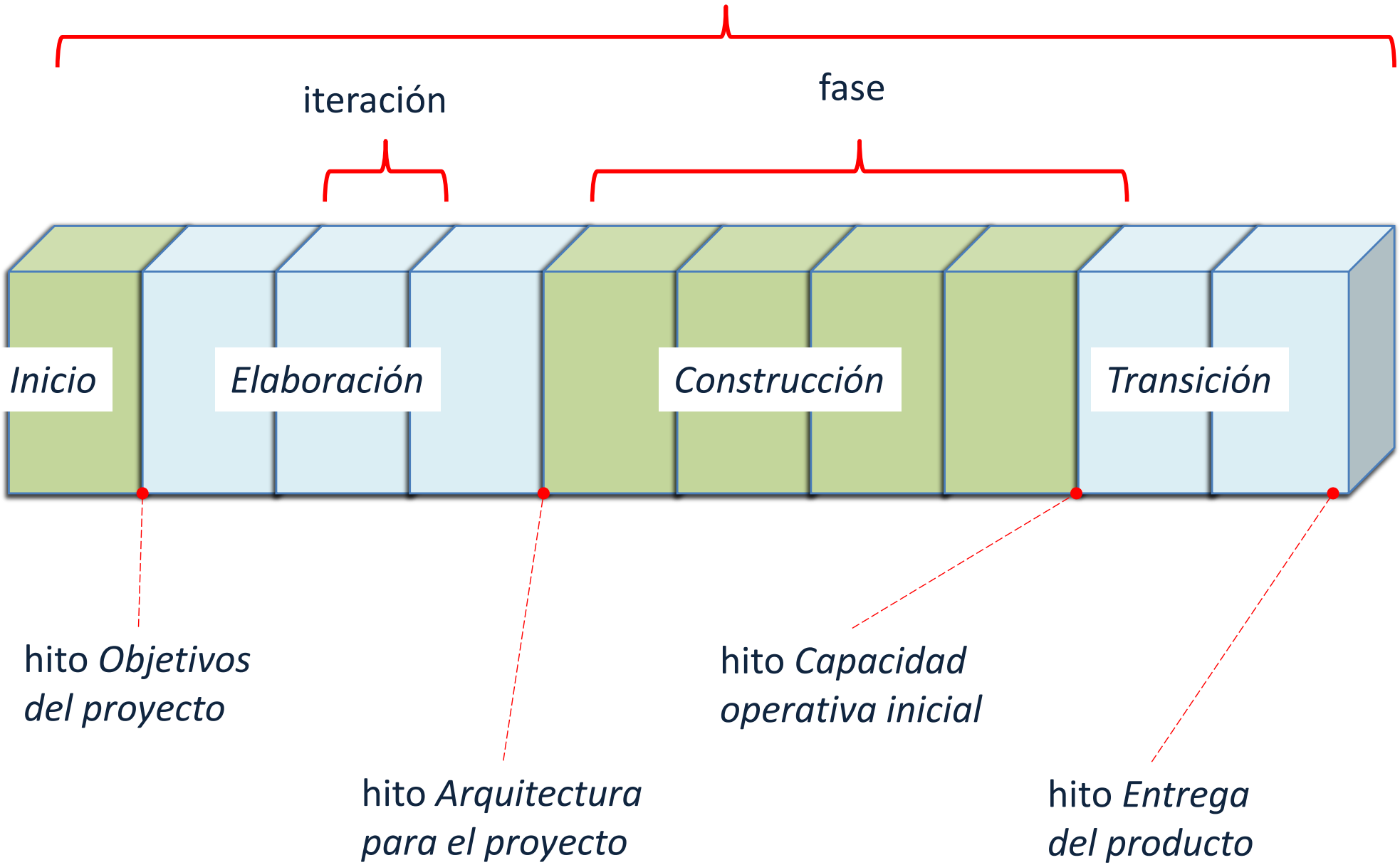
3) **Construcción**:

- Implementación iterativa de los elementos de menor riesgo

4) **Transición**:

- Pruebas beta, instalación

ciclo de desarrollo
(el número de iteraciones es solo un ejemplo)



El RUP no es una nueva versión del proceso secuencial

Los nombres de las fases reflejan el **estado** del proyecto,

… no las actividades que se llevan a cabo

Inicio es una fase de *análisis de factibilidad*:

- se investiga lo suficiente para tomar la decisión de continuar o parar
- no es una fase de requisitos —en el RUP no hay una fase de requisitos

Elaboración es una fase de *mitigación de riesgos*:

- la arquitectura central es implementada iterativamente
- no es una fase de requisitos o diseño —en el RUP tampoco hay una fase de diseño

El RUP describe las actividades de un proyecto dentro de *disciplinas*

Una **disciplina** es un conjunto de actividades, y los artefactos relacionados, en un cierto ámbito; p.ej.,

- Gestión del proyecto
- Requisitos (artefacto: Modelo de Casos de Uso)
- Análisis y Diseño (artefacto: Modelo de Diseño)
- Implementación (código y pruebas unitarias)
- Evaluación
- Instalación
- Gestión de configuración y cambios
- Entorno

Durante una iteración, en cualquier fase, se trabaja en todas, o en la mayoría, de las disciplinas

El esfuerzo relativo puesto en una disciplina cambia a lo largo de las fases:

- en las iteraciones iniciales (p.ej., *Elaboración*), el énfasis se pone en requisitos y diseño, aunque también hay implementación
- en las iteraciones centrales (p.ej., *Construcción*), el énfasis se pone en la implementación —los requisitos se han estabilizado gracias a la retroalimentación y adaptación

¿Qué se hace en cada iteración, a lo largo de las fases?

Abordar nuevos requisitos:

- se elige un subconjunto pequeño de requisitos

Extender el sistema incrementalmente:

- se diseña, implementa y prueba rápidamente

Ocasionalmente, se puede revisar y mejorar software existente:

- p.ej., mejorar el desempeño de un subsistema en lugar de agregarle nuevas capacidades

¿Cuál es el resultado de cada iteración?

Es un *sistema ejecutable* pero incompleto:

- el sistema sólo estará listo para ser puesto en producción después de varias iteraciones, p.ej., 10

No es un prototipo experimental o desechable, sino un subconjunto del sistema final:

- el desarrollo iterativo e incremental **no es** desarrollo de prototipos desechables

El RUP debe ser *instanciado* para cada organización (y proyecto)

El RUP es un proceso genérico de desarrollo de software:

- algunas de sus prácticas y principios son invariantes, p.ej., desarrollo iterativo dirigido por los riesgos, verificación continua de la calidad
- … pero todas sus actividades y artefactos son opcionales

La **instanciación** consiste en elegir las prácticas y artefactos apropiados para un proyecto particular y documentarlos en el caso de desarrollo

Podemos aplicar el RUP *ágilmente*

Las actividades y los artefactos son opcionales —elijamos sólo los que agreguen valor al proyecto

Enfoquémonos en programar más que en documentar desde un comienzo

Empleemos prácticas de modelamiento ágil con UML

Los requisitos y el diseño van apareciendo y adaptándose a lo largo de varias iteraciones

… no se espera que estén completos antes de iniciar la implementación

No hay un plan detallado para todo el proyecto, sólo para la próxima iteración

… el plan del proyecto es sólo de alto nivel

Vieja escuela	Nueva escuela
Varios roles —*product manager, project manager*— comparten la responsabilidad de darle vida al producto	Una persona —el dueño del producto— está a cargo del producto y lidera el proyecto
Los *product managers* están separados de los equipos de desarrollo, ya sea por proceso o por departamento (u otro tipo de separación)	El dueño del producto es un integrante del equipo Scrum y trabaja continuamente cerca del ScrumMaster y del equipo
Se hace mucha investigación de mercado, planificación del producto y análsis de negocio por adelantado	Se realiza un trabajo por adelantado mínimo para crear una visión que describa aproximadamente cómo se verá y qué hará el producto
Descubrimiento y definición del producto por adelantado: se detallan y se congelan los requisitos temprano	El descubrimiento del producto es un proceso continuo; los requisitos emergen. No hay una fase de definición ni una especificación de requisitos. El backlog del producto es dinámico y evoluciona a partir de la retroalimentación del cliente y del usuario
La retroalimentación del cliente se recibe tarde, en pruebas de mercado y después del lanzamiento del producto	Entregas desde temprano y frecuentes junto a revisiones del sprint producen retroalimentación valiosa del cliente y del usuario, que ayuda a crear un producto que le encanta al cliente