Objeto de Estudio de Ingeniería de SW

Proceso de desarrollo de Software

Proceso de Desarrollo de Software

¿Qué es el PDSW?

Conjunto de etapas con la intención de lograr un objetivo:

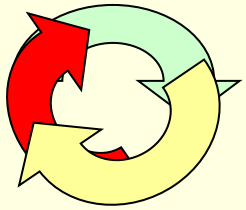
Obtener un software de calidad

en tiempo y presupuesto definido

Proceso de Software

Otra denominación del Proceso de Software

Al proceso de software también se le conoce como Ciclo de Vida del Software



Proceso de Desarrollo de Software

Fases Genéricas

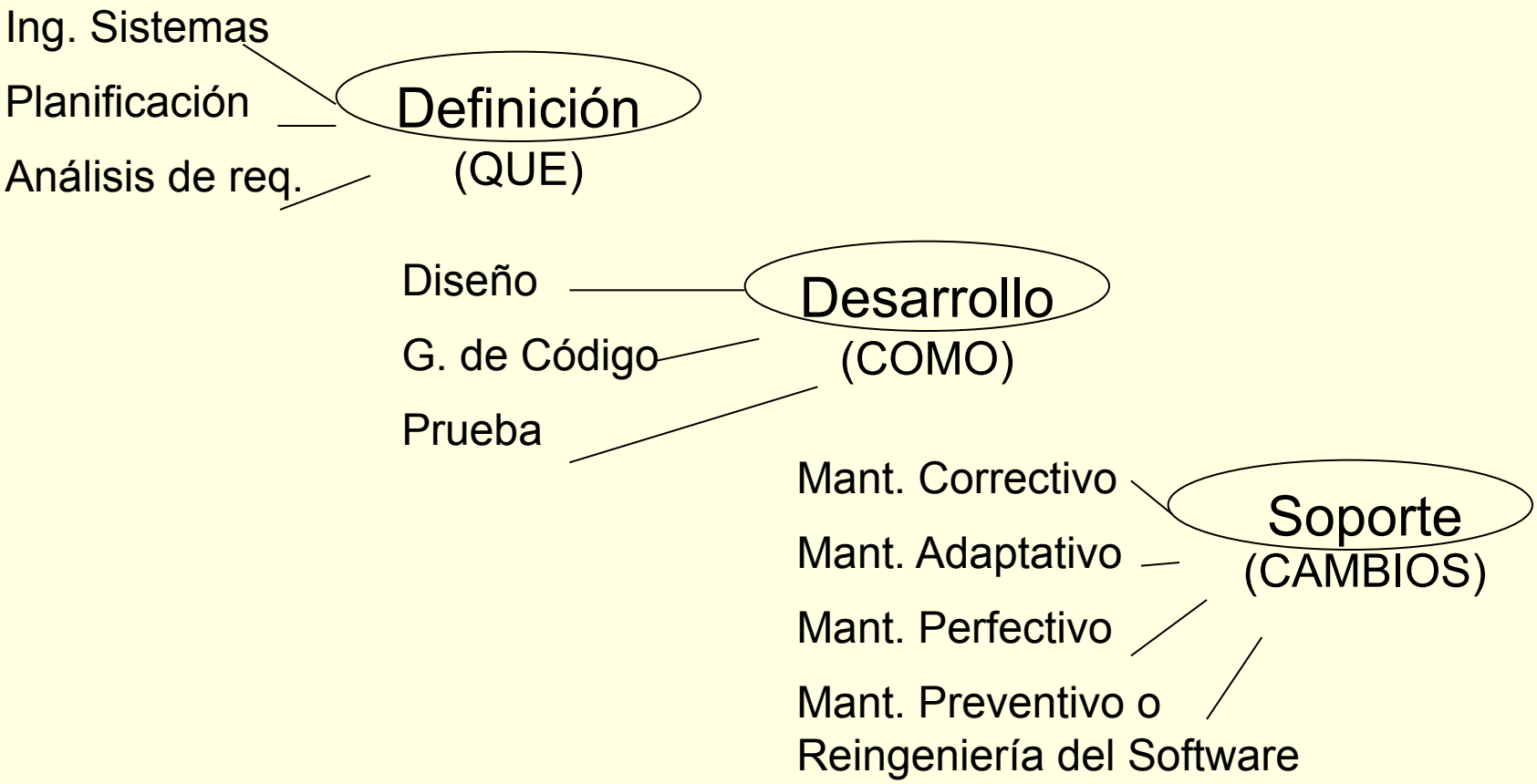
- La Fase de Definición ¿Qué?
- La Fase de Desarrollo ¿Cómo?
- La Fase de Mantenimiento - Cambio

Proceso de Desarrollo de Software

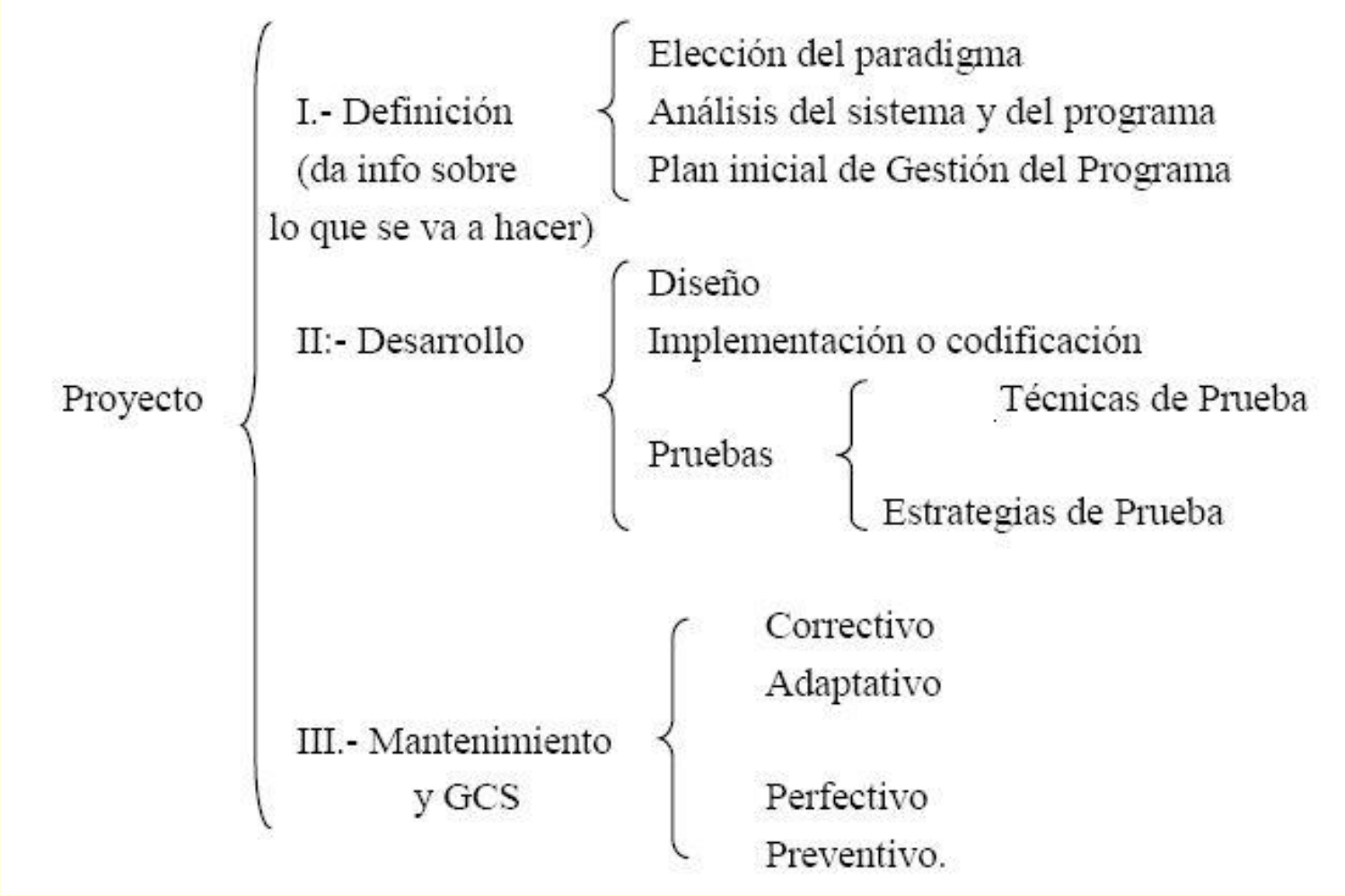
Fases y Actividades Genéricas o estructurales

- La Fase de Definición ¿Qué?
  - Análisis
  - Planificación
- La Fase de Desarrollo ¿Cómo?
  - Diseño
  - Codificación
  - Pruebas
- La Fase de Mantenimiento – Cambio
  - Preventivo
  - Correctivo
  - Adaptativo
  - Perfectivo

El Proceso – Visión Genérica



El Proceso – Visión Genérica



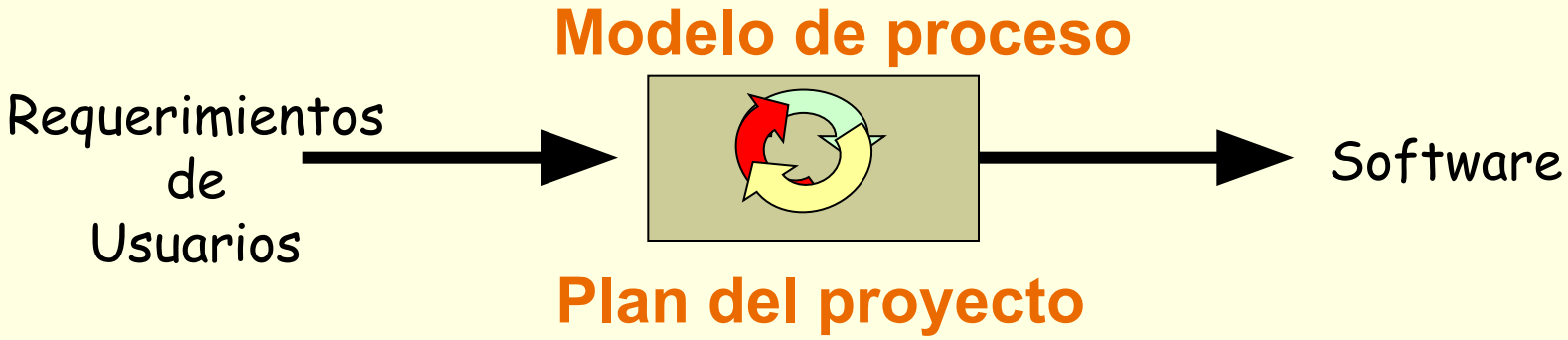
Modelo de Proceso de Software

DEFINICION:

¿Qué es un Modelo de Proceso de Software?

Es una estrategia de desarrollo que los ingenieros de software deben emplear para resolver problemas de la industria de software

Modelo de proceso & Planificación



Modelos de Procesos de Software

SCRUM

RUP

Lineal Secuencial

DRA

Desarrollo Concurrente

Construcción de Prototipos

Incremental

Espiral

XP

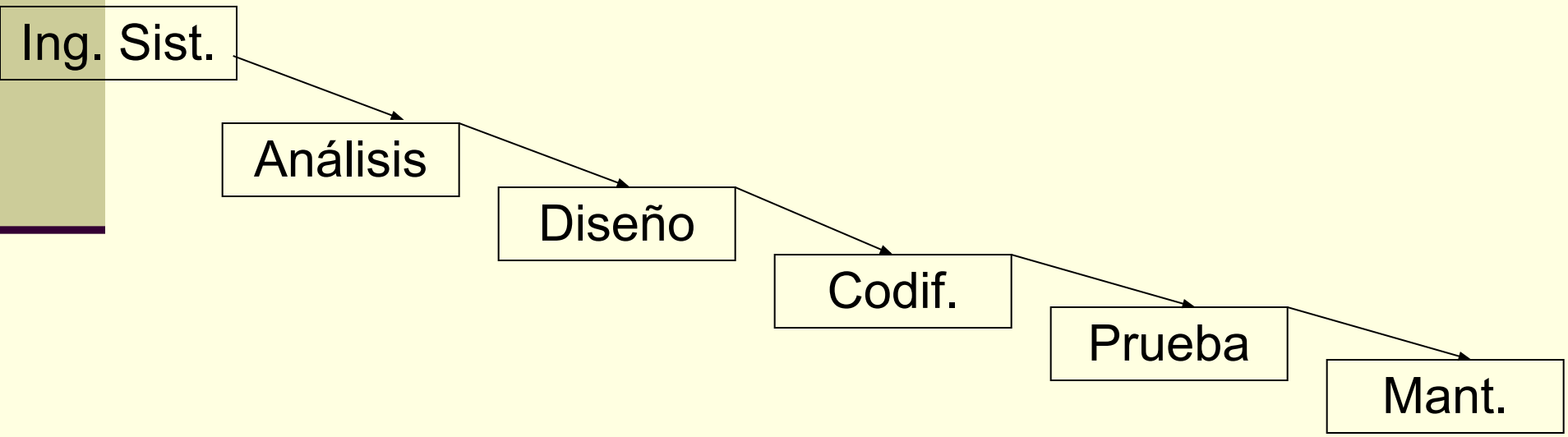
Ensamblaje de Componentes

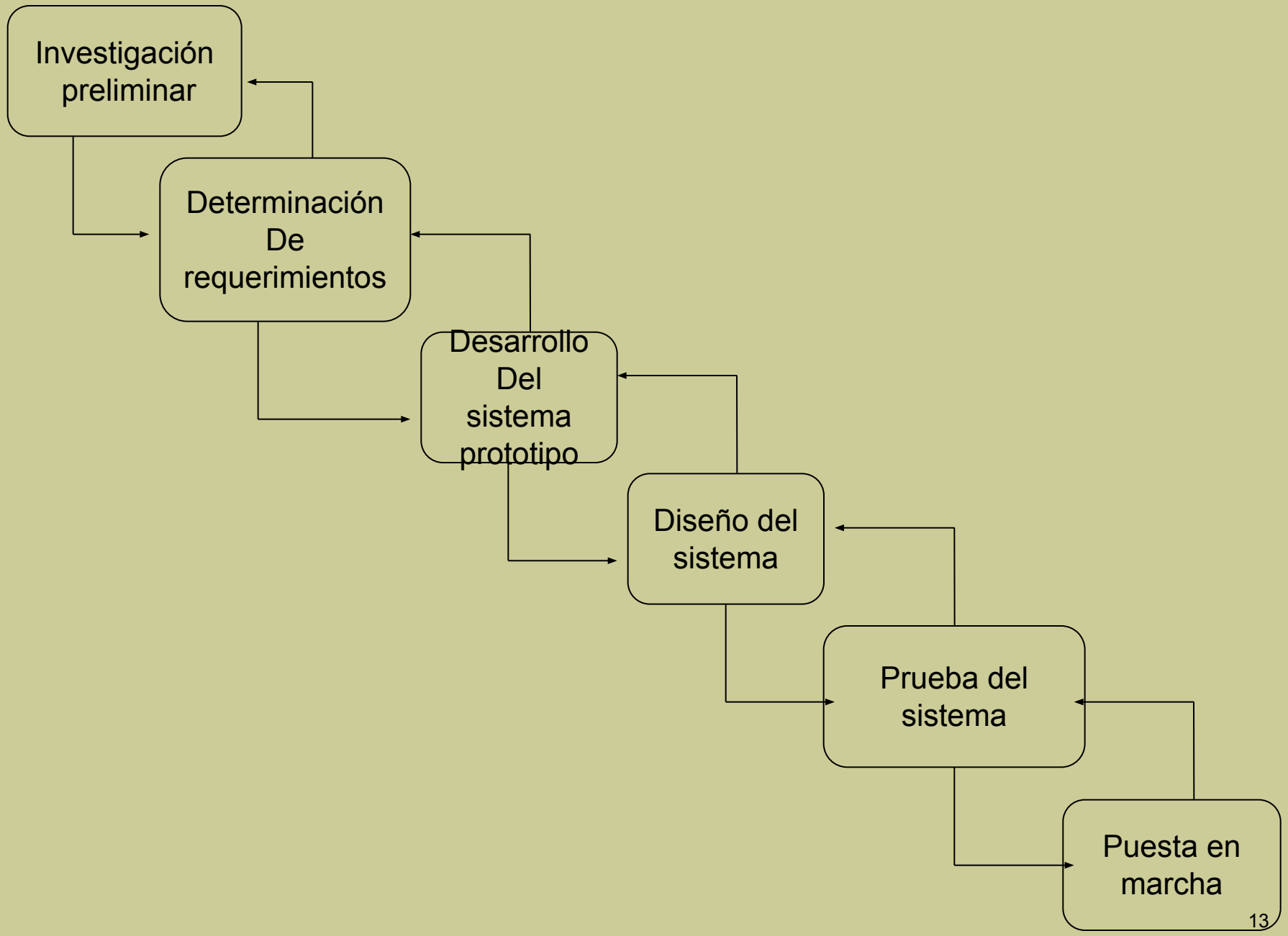
Modelos de Procesos de Software

El problema es seleccionar el modelo de proceso de software apropiado que debe aplicar el equipo de proyecto

Modelo Lineal Secuencial

- Ciclo de vida clásico, modelo en cascada
- + antiguo, + usado
- Enfoque sistemático secuencial
- Dirigido por documentos




Investigación preliminar
Determinación De requerimientos
Desarrollo Del sistema prototipo
Diseño del sistema
Prueba del sistema
Puesta en marcha

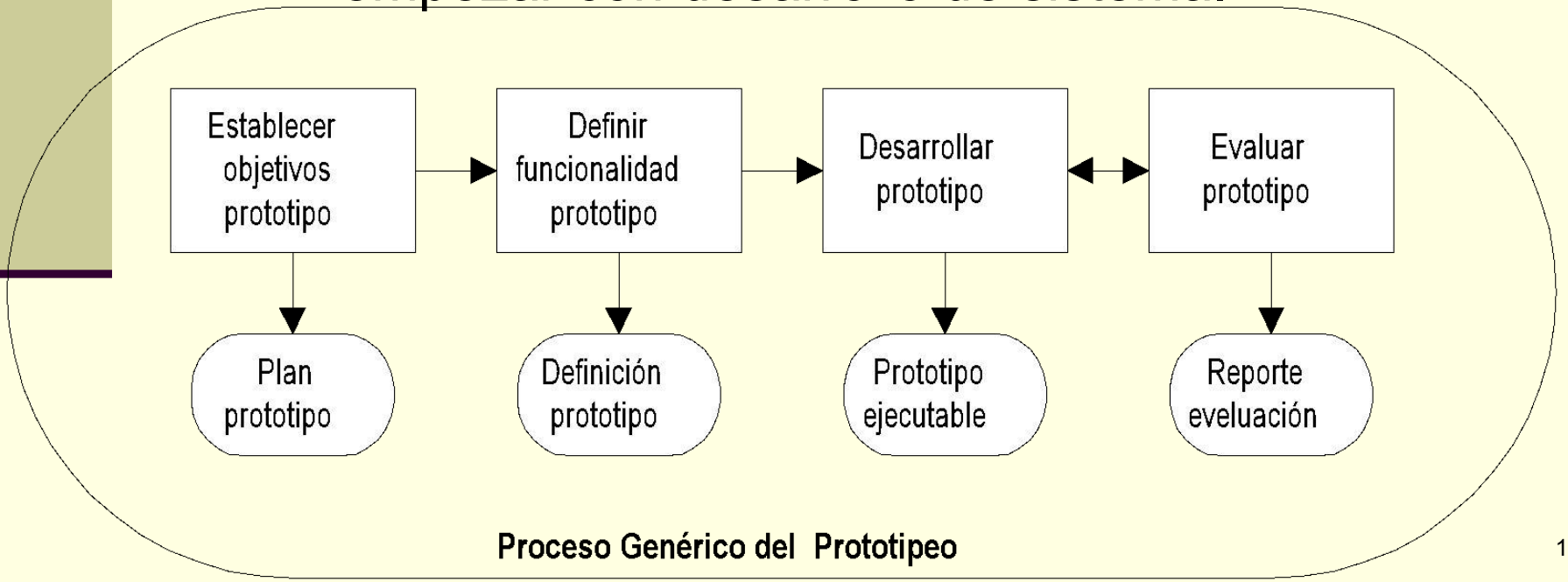
Modelo Lineal Secuencial

- Críticas:
  - Proyectos reales raras veces se ajustan.
  - Raras veces cliente expone todos los req. de entrada.
  - Producto operativo al final => Paciencia (cliente) alta.
- Ventajas
  - Fácil administrar, comprender
  - Todos lo conocen
- Consejo: ¿Cuándo usar?

Usar cuando todos los requerimientos han sido establecidos claramente de entrada.

Modelo de Construcción de Prototipos

- No están claros los reqs. de entrada
- Iterativo. Hasta cuando se itera?
- Working prototype, desechar y empezar con desarrollo de sistema.

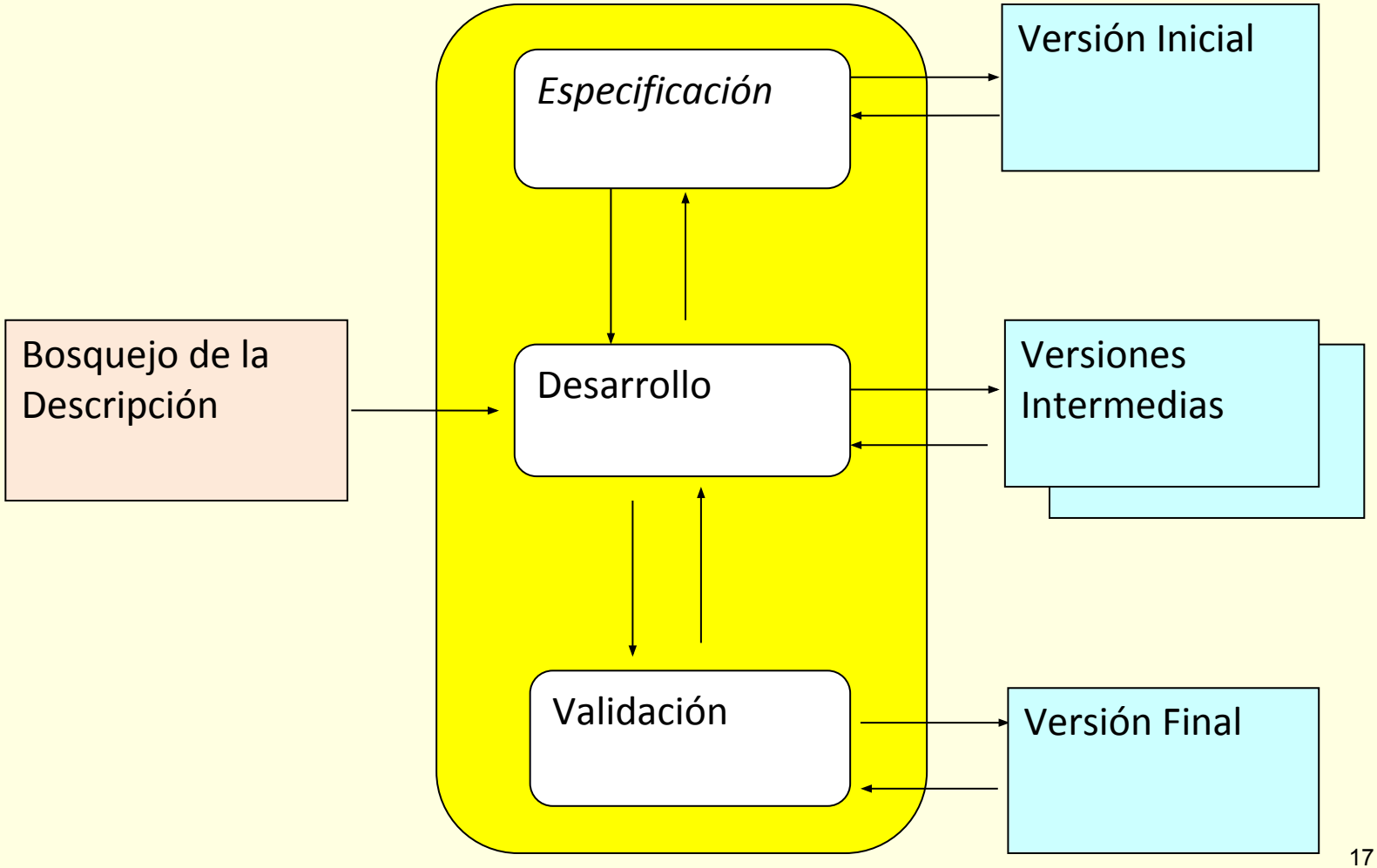


Proceso Genérico del Prototipeo

Modelo de Construcción de Prototipos

- Críticas:
  - Cliente cree que es el sistema.
  - Peligro de familiarización con malas elecciones iniciales (quick and dirty).
  - Difícil administrar: se necesita mucha experiencia
  - Costo
- Ventajas
  - Se detectan malos entendidos entre los desarrolladores y los usuarios
  - Se detectan servicios no detectados antes
  - Dificultades de uso o servicios confusos pueden ser identificados y refinados
  - Staff de desarrollo de software puede encontrar requerimientos incompletos o inconsistentes con el desarrollo del prototipo
  - El prototipo sirve como una base de la especificación para la producción de un sistema de calidad
- Consejo:¿Cuándo usar?
  - Usar cuando inicialmente no están claros los requerimientos.
  - Definir claramente de entrada las reglas de juego con el cliente.
  - No ceder a presión del cliente.

Modelo Prototipo Evolutivo



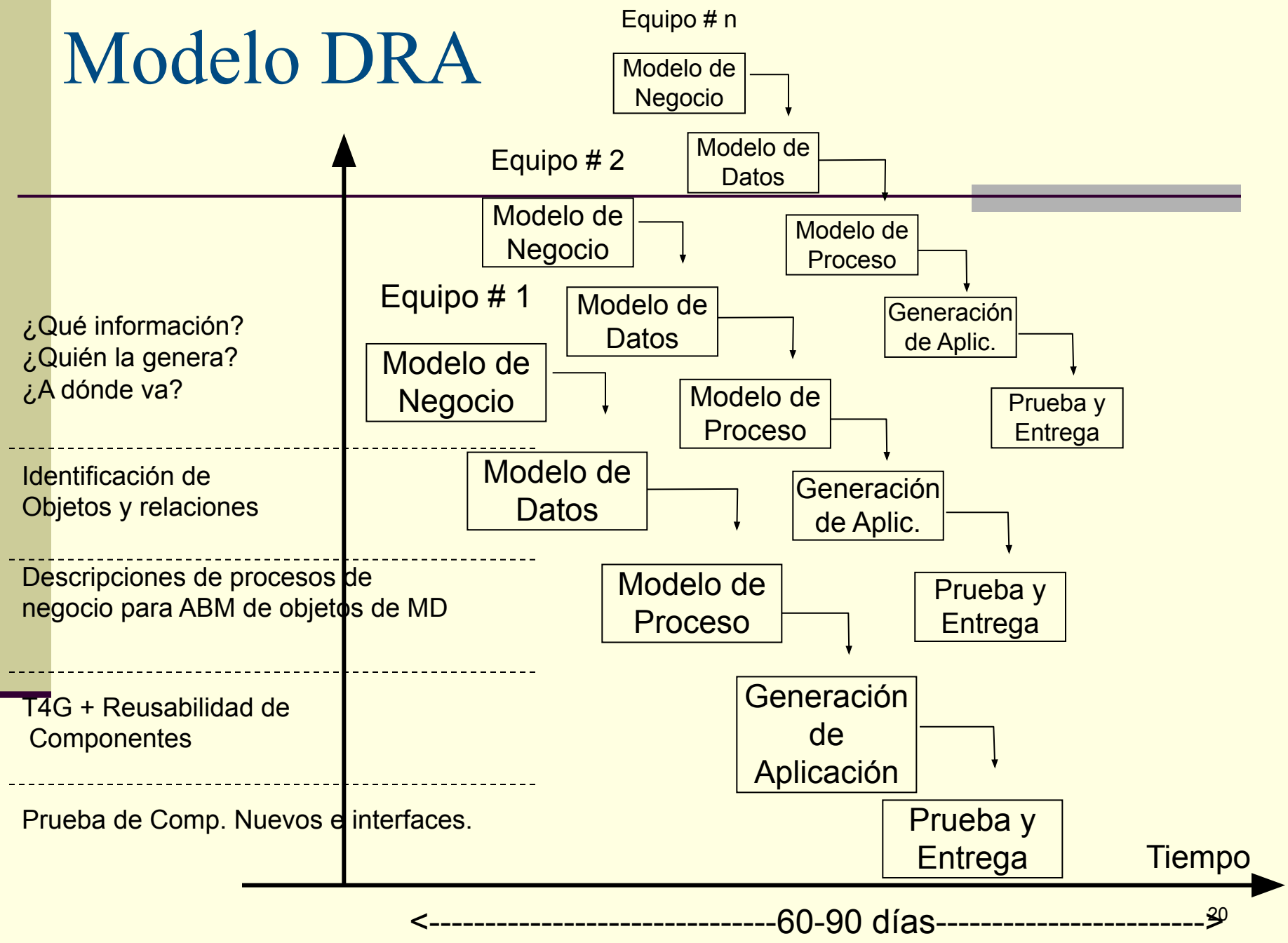
Modelo Prototipo Evolutivo

- Ventajas
  - Desarrollo rápido cuando no se conocen bien los requerimientos.
  - Cuando el usuario/desarrollador no sabe bien lo que quiere: acierta/falla.
  - Adecuado para sistemas pequeños y de vida corta.
- Desventajas
  - Requiere técnicas y herramientas especiales, para un desarrollo rápido.
  - Los cambios continuos tienden a corromper la estructura del sistema haciendo el mantenimiento futuro muy difícil.
  - Es imprescindible la pericia de un experto en prototipeo en el equipo de desarrollo.
  - La organización debe estar consciente que el tiempo de vida de los sistemas desarrollados así es corto.
- ¿Cuándo usar?
  - Es recomendable usar para sistemas pequeños o de vida corta. Cuando es difícil conocer bien los requerimientos.

Modelo de Desarrollo Rápido de Aplicaciones (DRA)

- Lineal secuencial con ciclo extremadamente corto.
- Candidatos: sistemas que se pueden modularizar => equipos de desarrollo paralelos.
- Basado en el uso de componentes y T4G.

Modelo DRA


Equipo # n
Modelo de Negocio
Equipo # 2
Modelo de Datos
Modelo de Negocio
Modelo de Proceso
Equipo # 1
Modelo de Datos
Generación de Aplic.
Modelo de Negocio
Modelo de Proceso
Prueba y Entrega
Modelo de Datos
Generación de Aplic.
Modelo de Proceso
Prueba y Entrega
Generación de Aplicación
Prueba y Entrega
¿Qué información?
¿Quién la genera?
¿A dónde va?
Identificación de Objetos y relaciones
Descripciones de procesos de negocio para ABM de objetos de MD
T4G + Reusabilidad de Componentes
Prueba de Comp. Nuevos e interfaces.
Tiempo
<----------60-90 días---------->

Modelo DRA

- Críticas:
  - Proyectos grandes => gran nro. de personas.
  - Alto compromiso en tiempo.
  - No apto para todo tipo de sistema (ej. no modularizable, bajo reuso de componentes).
  - Desaconsejable cuando existen riesgos tecnológicos altos o alta interoperatividad con programas ya existentes.

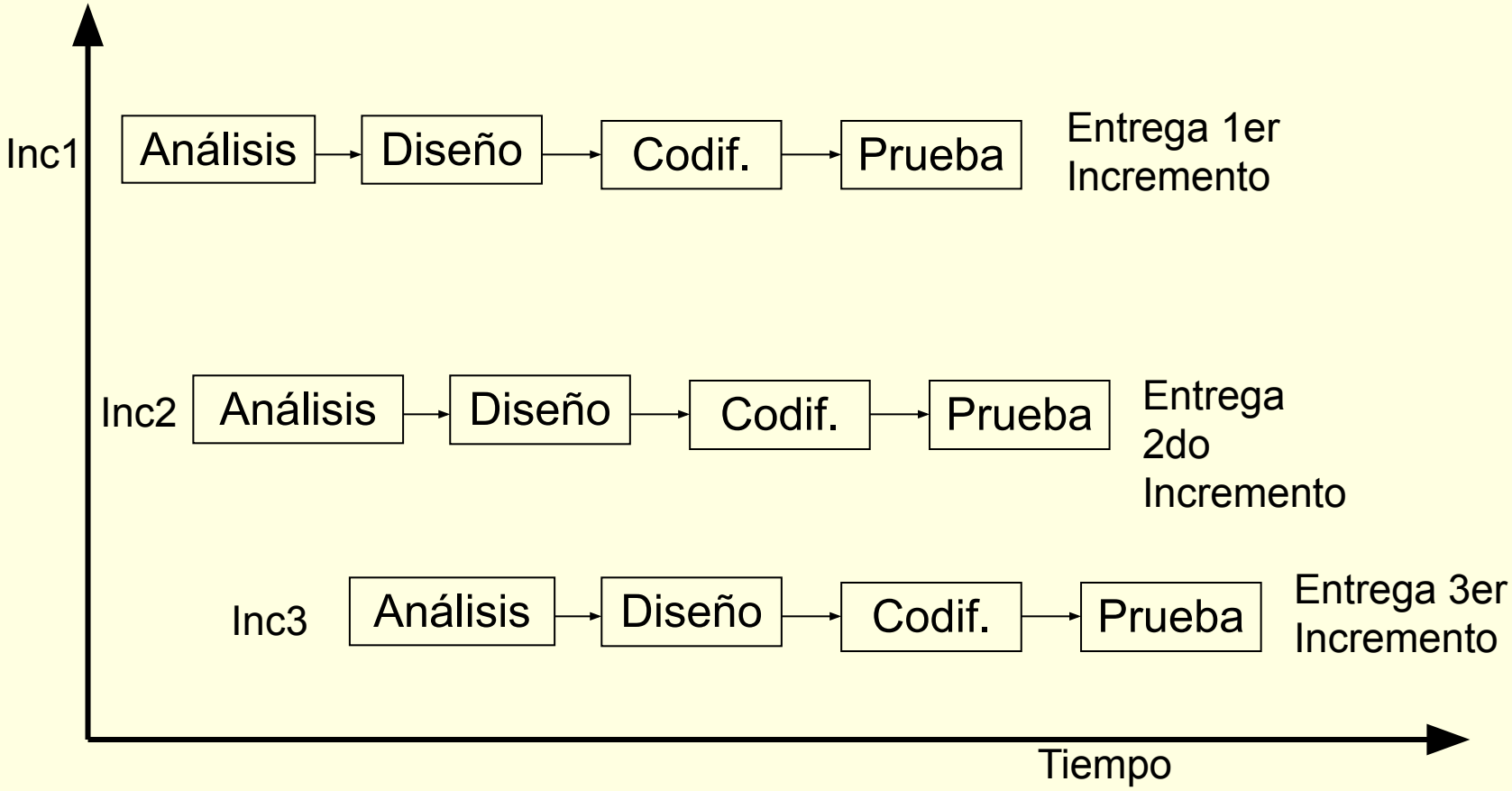
Modelos Evolutivos

- Se adaptan más fácilmente a los cambios introducidos a lo largo del desarrollo.
- Iterativos
- En cada iteración se obtienen versiones más completas del SW.
- Modelos Evolutivos:
  - Modelo Incremental (*)
  - Modelo en Espiral (*)
  - Modelo de Desarrollo Basado en Componentes (*)
  - Modelo WINWIN
  - Modelo de Desarrollo Concurrente

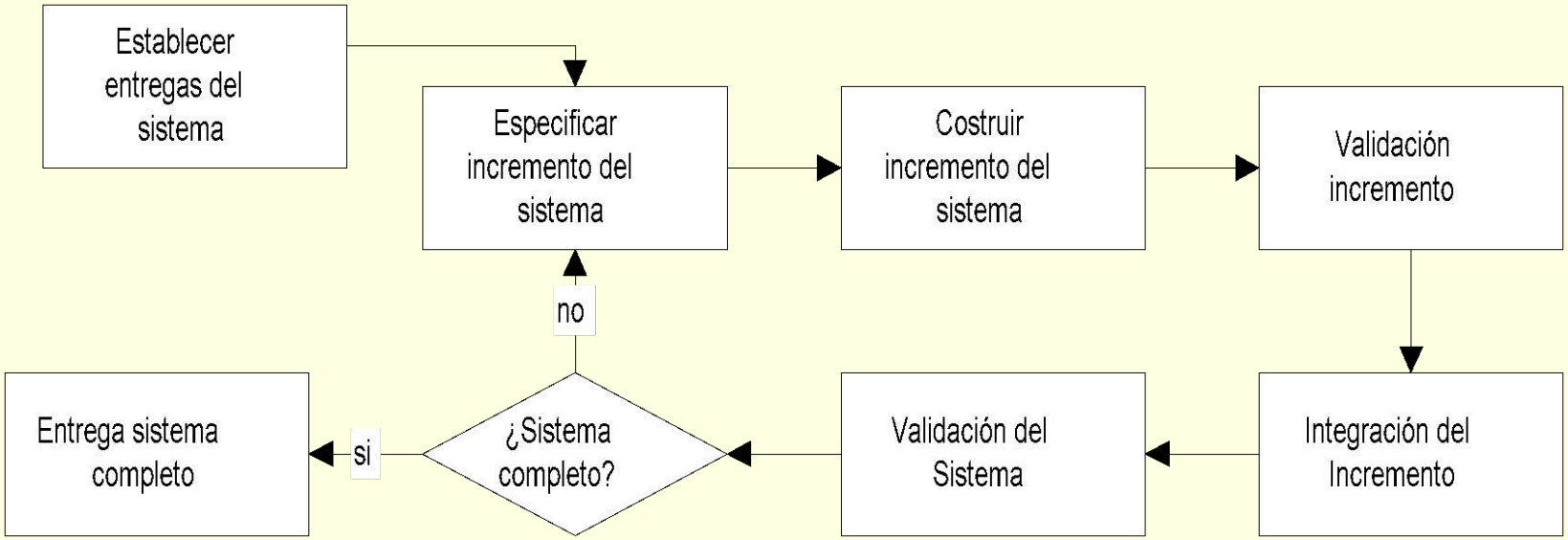
Modelo Incremental

- Iteración de Lineal Secuencial.
- Cada iteración devuelve un "**Incremento**" o **versión operativa**.
- Útil cuando no se está seguro de cumplir con plazos de tiempo o se tiene una fecha imposible de cambiar.

Modelo Incremental



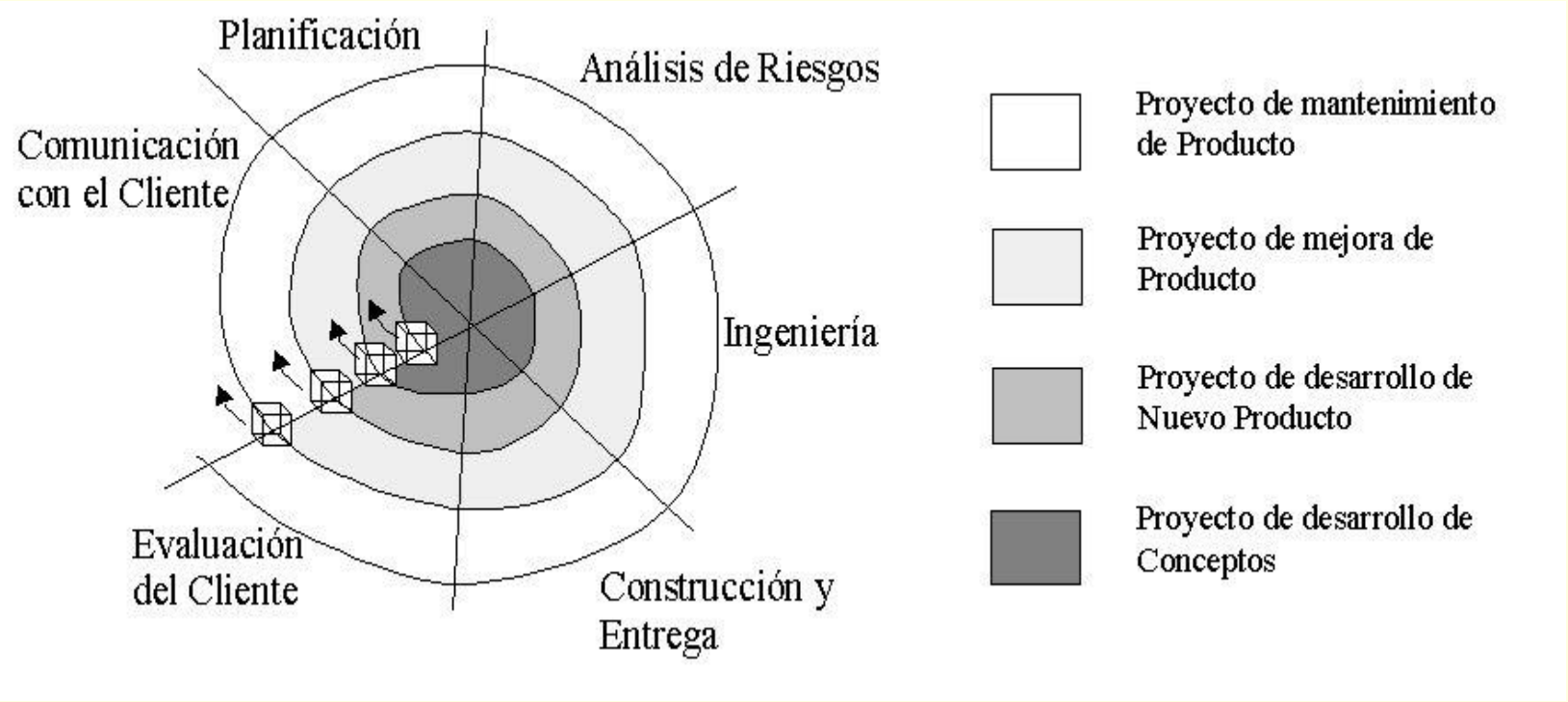
… Modelo Incremental



Modelo Incremental

- **Ventajas:**
  - Ofrece retroalimentación
  - Disminuye progresivamente el número de errores en las partes que faltan
  - Disminuye el riesgo del desarrollo, errores se corrigen progresivamente
  - Disminuye el tiempo de entrenamiento al usuario, que es progresivo
  - El usuario no tiene que esperar hasta el final para hacer uso del sistema
- **Problemas:**
  - Definición del contrato, porque se planifica en forma detallada por incremento
  - Los planes y documentación se entregan con cada incremento del sistema
  - Una vez que una parte es entregada sus interfaces son congeladas e incrementos posteriores deben adaptarse a estas
- **Nota**: Una evolución de este enfoque se conoce como *Programación Extrema (XP-Extreme Programming).*

Modelo en Espiral

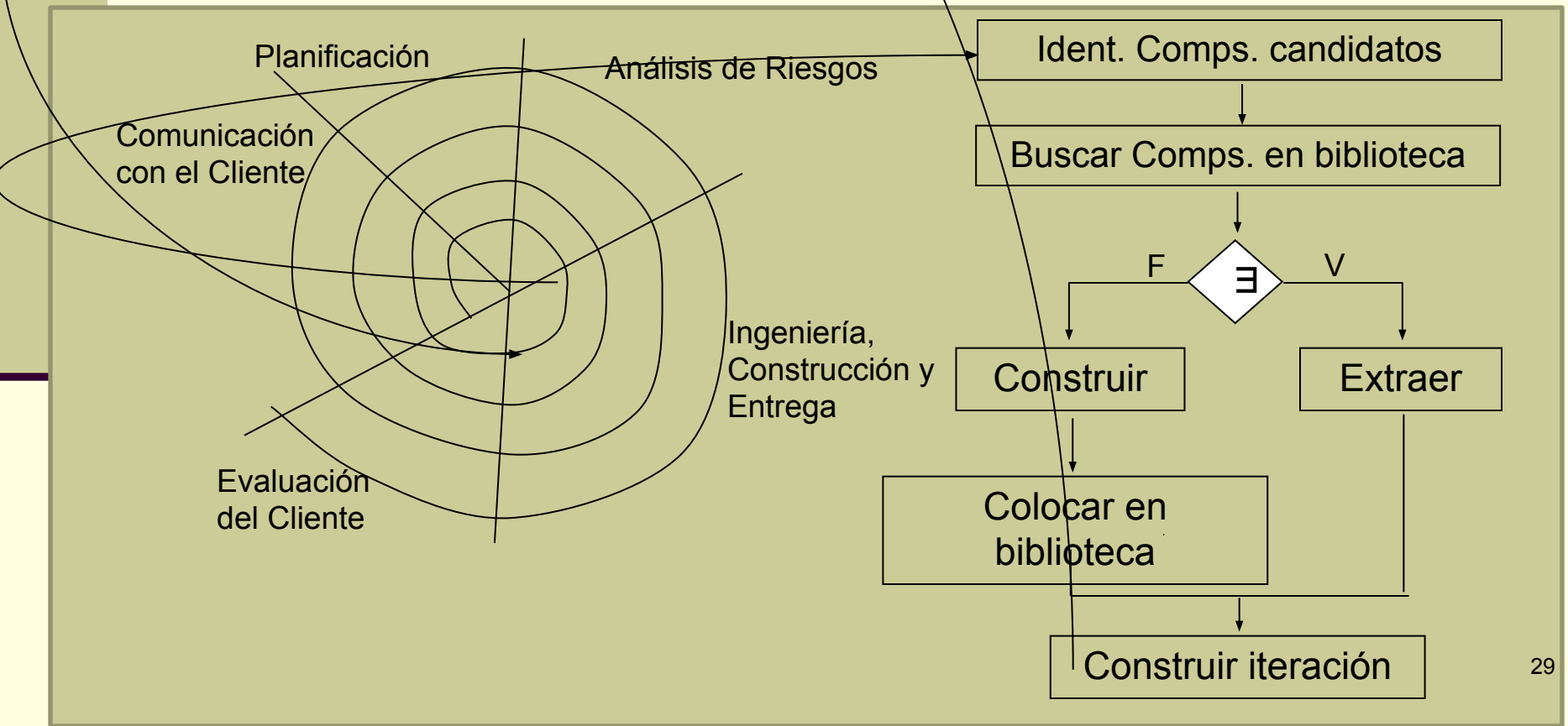


Modelo en Espiral

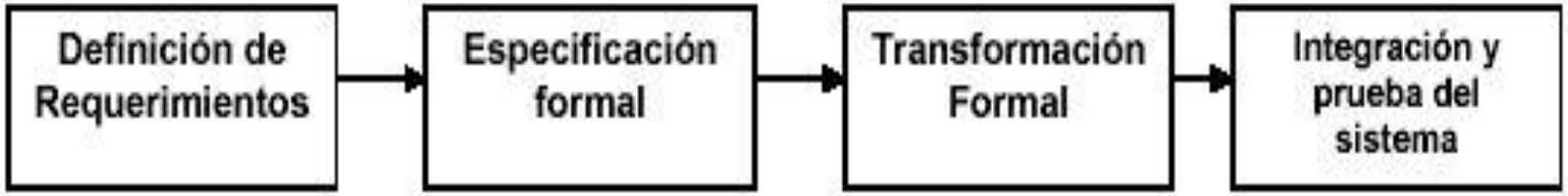
- Ventajas:
  - Útil para proyectos grandes.
  - Permite usar el prototipado en todas las etapas de la evolución para reducir el riesgo.
  - Mantiene el enfoque sistemático de los pasos sugeridos por el lineal secuencial, pero lo incorpora dentro de un marco iterativo más real.
- Críticas:
  - Difícil de convencer a los clientes de que es controlable.
  - Requiere mucha habilidad para el análisis de riesgos y de esta habilidad depende su éxito.
  - No ha sido utilizado tanto como el lineal secuencial o el de prototipos.
  - Se necesita mucha experiencia

Desarrollo Basado en Componentes

- Basado en modelo en Espiral (evolutivo e iterativo) + Tecnologías de Objetos.
- Enfatiza la Reusabilidad.



Modelo de Métodos Formales



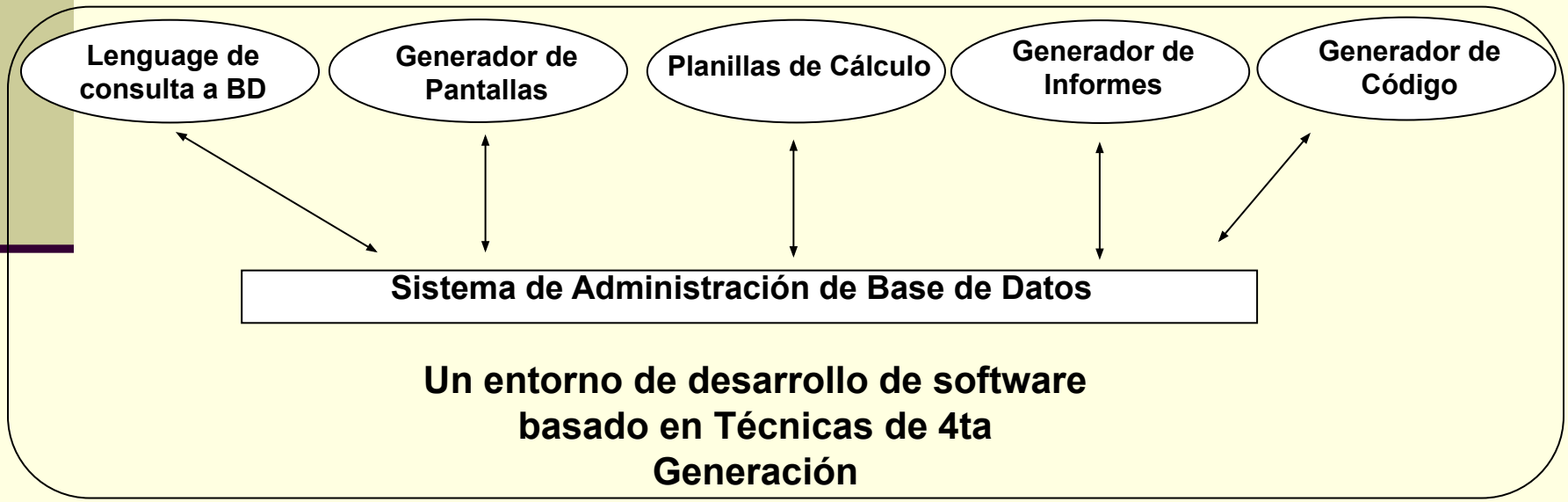
Usan notación rigurosa.

Buen nivel de manejo de Lógica y Algebra.

- Ventajas
  - Demostraciones formales de propiedades.
  - Especificaciones sin ambigüedades: Consistencia
  - Utiles para sistemas críticos.
- Críticas
  - Dificulta validación con cliente => combinación con otras técnicas semi-formales.
  - La ejecución de este tipo de modelos requiere mucho tiempo y esfuerzo.
  - Pocos desarrolladores dominan de algebra y matemáicas para especificación.

Técnicas de Cuarta Generación (T4G)

- Herramientas que facilitan la realización de especificaciones a alto nivel -> código fuente.
- Basadas en Lenguajes de 4ta Generación (L4G) y uso de herramientas CASE
- **Ventajas:** Reducción en tiempo de desarrollo.



Un entorno de desarrollo de software basado en Técnicas de 4ta Generación

Técnicas de Cuarta Generación (T4G)

- Críticas:
  - Código ineficiente.
  - No mas fáciles de usar que L3G.
  - Mantenimiento cuestionable.
- Consejo:

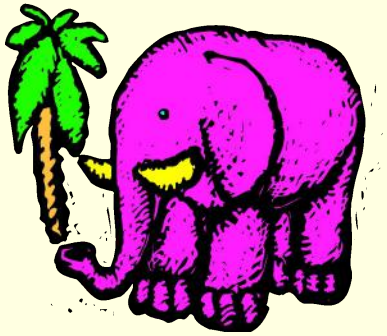
En sistemas grandes, aunque se usen T4G se debe hacer análisis, diseño y pruebas.

Métodos Agiles

Manifiesto ágil (2001)

Origen de los “métodos ágiles”

En marzo de 2001, 17 críticos de estos modelos, convocados por Kent Beck, que acababa de definir una nueva metodología denominada *Extreme Programming*, se reunieron en Salt Lake City para discutir sobre los modelos de desarrollo de software.
En la reunión se acuñó el término “**Métodos Ágiles**” para definir a aquellos que estaban surgiendo como alternativa a las metodologías formales, (CMM-SW, PMI, SPICE) a las que consideraban excesivamente “pesadas” y rígidas por su carácter normativo y fuerte dependencia de planificaciones detalladas, previas al desarrollo.
Los integrantes de la reunión resumieron en cuatro postulados lo que ha quedado denominado como “Manifiesto Ágil”, que capturaba el espíritu en el que se basan estos métodos.
Aunque como se verá más adelante, en la actualidad hay aproximaciones que combinan lo mejor de ambos enfoques (formal y ágil), hasta 2005, en cada lado sus defensores adoptaron posturas radicales, quizá más ocupadas en descalificar a la contraria que en estudiarla y conocerla para mejorar la propia.



Métodos Agiles

Manifiesto ágil (2001)

Estamos poniendo al descubierto mejores métodos para desarrollar software, haciéndolo y ayudando a otros a que lo hagan. Con este trabajo hemos llegado a valorar:

A los individuos y su interacción	por encima	de los procesos y las herramientas
El software que funciona	por encima	de la documentación exhaustiva
La colaboración con el cliente	por encima	la negociación contractual
La respuesta al cambio	por encima	seguimiento de un plan

Aunque hay valor en los elementos de la derecha, valoramos más los de la izquierda

Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, *Martin Fowler,* James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas

http://agilemanifesto.org/

Métodos Agiles

eXtreme Programming (XP)

Este es el método que más popularidad ha alcanzado entre las metodologías ágiles, y posiblemente sea también el más transgresor de la ortodoxia basada en procesos.
Su creador, Kent Beck fue el alma mater del Manifiesto Ágil.
Extreme Programming (XP) se basa sobre la suposición de que es posible desarrollar software de gran calidad a pesar, o incluso como consecuencia del cambio continuo.
Su principal asunción es que con un poco de planificación, un poco de codificación y unas pocas pruebas se puede decidir si se está siguiendo un camino acertado o equivocado, evitando así tener que echar marcha atrás demasiado tarde.

Valores que inspiran XP

- SIMPLICIDAD
- FEEDBACK
- CORAJE
- COMUNICACIÓN
- RESPETO

Métodos Agiles

eXtreme Programming (XP)

Definición: (De Wikipedia, la enciclopedia libre)

Extreme Programming (XP) es una metodología liviana para equipos pequeños encargados de desarrollar software en proyectos cuyos requerimientos son ambiguos o muy volátiles. XP propone un proceso de desarrollo liviano, eficiente, de bajo riesgo, flexible, predecible y científico.

Se puede considerar la programación extrema como la adopción de las mejores metodologías de desarrollo de acuerdo a lo que se pretende llevar a cabo con el proyecto, y aplicarlo de manera dinámica durante el ciclo de vida del software.

La **programación extrema** o *eXtreme Programming* (XP) es un enfoque de la ingeniería de software formulado por Kent Beck, autor del primer libro sobre la materia, *Extreme Programming Explained: Embrace Change* (1999). Es el más destacado de los procesos ágiles de desarrollo de software. Al igual que éstos, la programación extrema se diferencia de las metodologías tradicionales principalmente en que *pone más énfasis en la adaptabilidad que en la previsibilidad. Los defensores de XP consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Creen que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos.*

Métodos Agiles

eXtreme Programming (XP)

Principios

1. Simplicidad: *simplifica el diseño. Para que sea mantenible* necesario la *refactorización* del código.

simplicidad **+**autoría colectiva del código **+** la programación por parejas

➢ más grande el proyecto, todo el equipo conocerá más y mejor el sistema completo.

Métodos Agiles

eXtreme Programming (XP)

Principios

2. Comunicación:

•Código comunica mejor mientras más simple.

•Código autodocumentado más fiable que comentarios

•Pruebas unitarias muestran ejemplos concretos de como utilizar su funcionalidad.

•Programación por parejas.

•Cliente forma parte del equipo de desarrollo.

•El cliente decide qué características tienen prioridad y siempre debe estar disponible para solucionar dudas.

Métodos Agiles

eXtreme Programming (XP)

Principios

3. Retroalimentación (feedback):

- Cliente integrado en el proyecto: su opinión sobre el estado del proyecto se conoce en tiempo real.
- Ciclos que muestran resultados, ayuda a los programadores a centrarse en lo que es más importante.
- Pruebas unitarias informan sobre el estado de salud del código.

Métodos Agiles

eXtreme Programming (XP)

Principios

4. Coraje o Valentía:

•Programación en parejas puede ser difícil de aceptar, parece como si la productividad se fuese a reducir a la mitad (beneficia en calidad sin repercutir en productividad)

•Coraje para implementar las características que el cliente quiere ahora sin caer en la tentación de un enfoque más flexible que permita futuras modificaciones. No se debe emprender el desarrollo de grandes marcos de trabajo (frameworks) mientras el cliente espera.

• La forma de construir marcos de trabajo es mediante la r*efactorización* del código en sucesivas aproximaciones.

Métodos Agiles

Principios

5. Respeto:

•Añadido en la segunda edición de *Extreme Programming Explained*

•Programadores no pueden realizar cambios que hacen que las pruebas existentes fallen ó que demore el trabajo de sus compañeros.

• Los miembros respetan su trabajo, buscan alta calidad en el producto y diseño más óptimo para la solución a través de la *refactorización* del código.

Métodos Agiles

eXtreme Programming (XP)

Características

•**Desarrollo iterativo e incremental**: pequeñas mejoras, unas tras otras.

•**Pruebas unitarias continuas,** frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación. Véase, por ejemplo, las herramientas de prueba JUnit orientada a Java, DUnit orientada a Delphi y NUnit para la plataforma.NET. Estas dos últimas inspiradas en JUnit.

•**Programación en parejas**: dos personas en un mismo puesto. Mayor *calidad* del código escrito de esta manera -el código es revisado y discutido mientras se escribe- es más importante que la posible pérdida de productividad inmediata. Parejas se intercambian.

•Frecuente **integración del equipo de programación con el cliente** o usuario. Se recomienda que un representante del cliente trabaje junto al equipo de desarrollo.

•**Corrección de todos los errores** antes de añadir nueva funcionalidad. Hacer entregas frecuentes.

Métodos Agiles

eXtreme Programming (XP)

Características

- ***Refactorización* del código**, es decir, reescribir ciertas partes del código para aumentar su legibilidad y mantenibilidad pero sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo.
- **Propiedad del código compartida**: en vez de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve el que todo el personal pueda corregir y extender cualquier parte del proyecto.
- **Simplicidad** en el código: es la mejor manera de que las cosas funcionen. XP apuesta que es más sencillo hacer algo simple y tener un poco de trabajo extra para cambiarlo si se requiere, que realizar algo complicado y quizás nunca utilizarlo.

Métodos Agiles

eXtreme Programming (XP)

Características (todas)

- **Desarrollo iterativo e incremental:**
  - pequeñas mejoras, unas tras otras.
- **Pruebas unitarias continuas**, frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión.
- **Programación en parejas**
- Frecuente **integración** del equipo de programación **con el cliente** o usuario.
- **Corrección de todos los errores** antes de añadir nueva funcionalidad.
  - Hacer entregas frecuentes.
- **Refactorización del código**
- **Propiedad del código compartida**
- **Simplicidad en el código**:
  - es la mejor manera de que las cosas funcionen

Métodos Agiles

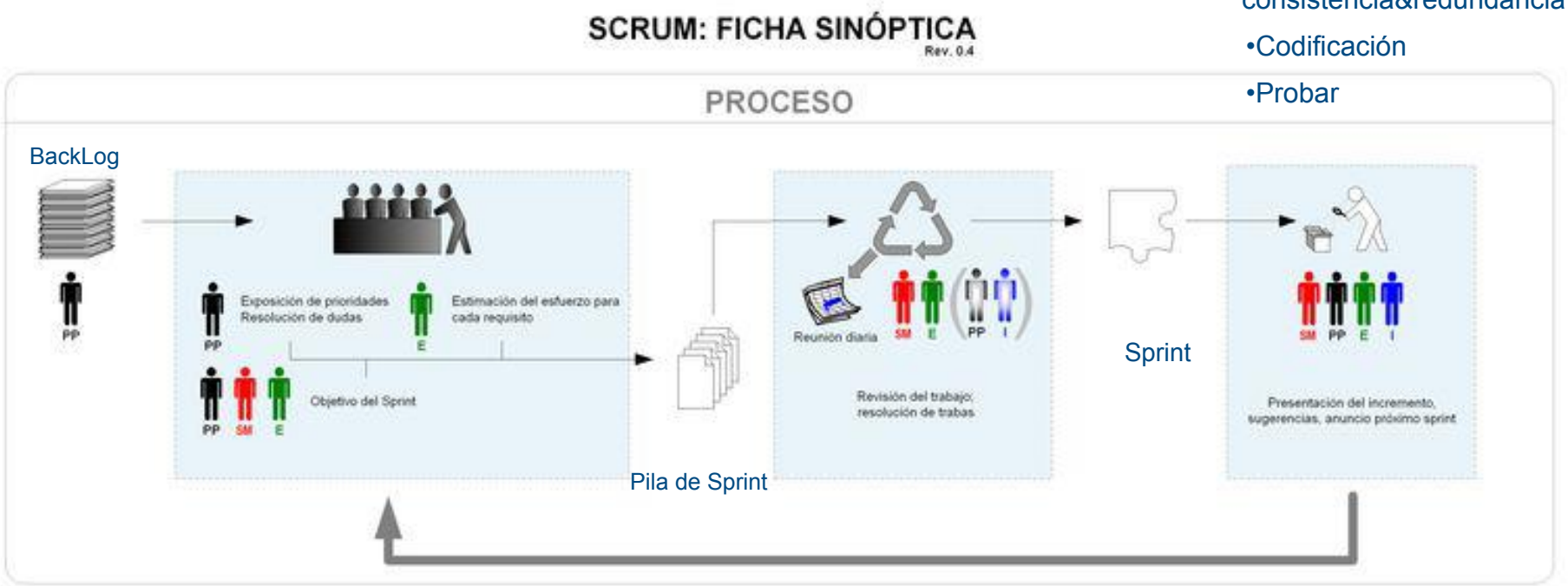
Conclusiones

La simplicidad y la comunicación son extraordinariamente complementarias.

- Con más **comunicación** resulta más fácil identificar qué se debe y qué no se debe hacer.
- Mientras más **simple** es el sistema, menos tendrá que comunicar sobre este, lo que lleva a una comunicación más completa, especialmente si se puede reducir el equipo de programadores.

Métodos Agiles: SCRUM

- Research
- Diseño
- Verificación de consistencia&redundancia
- Codificación
- Probar



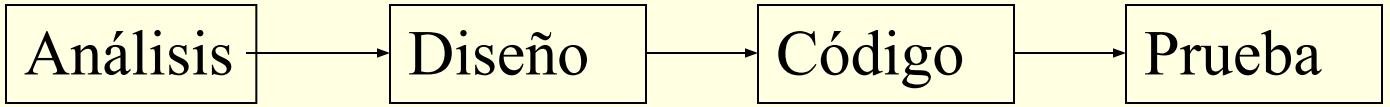
Minimizar el precio del error: Socializar

Proceso ágil de desarrollo iterativo e incremental. Origen : artículo "The New Product Development Game" (Takeuchi y Nonaka, 1986). Jeff Sutherland fue el 1ro en implementarlo en para desarrollo de software (1993, Ken Schwaber es su principal difusor)

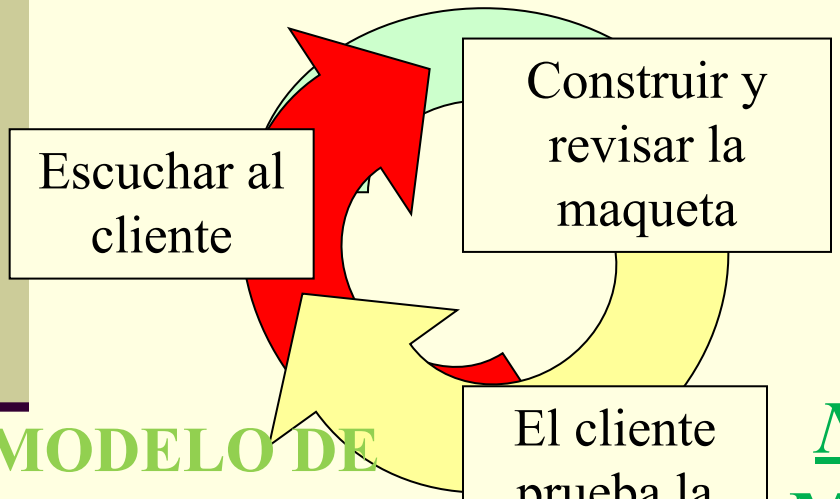
Backlog=Requerimientos aceptados que sirven para generar el sprint

Sprint= Requerimientos comprometidos para el desarrollo

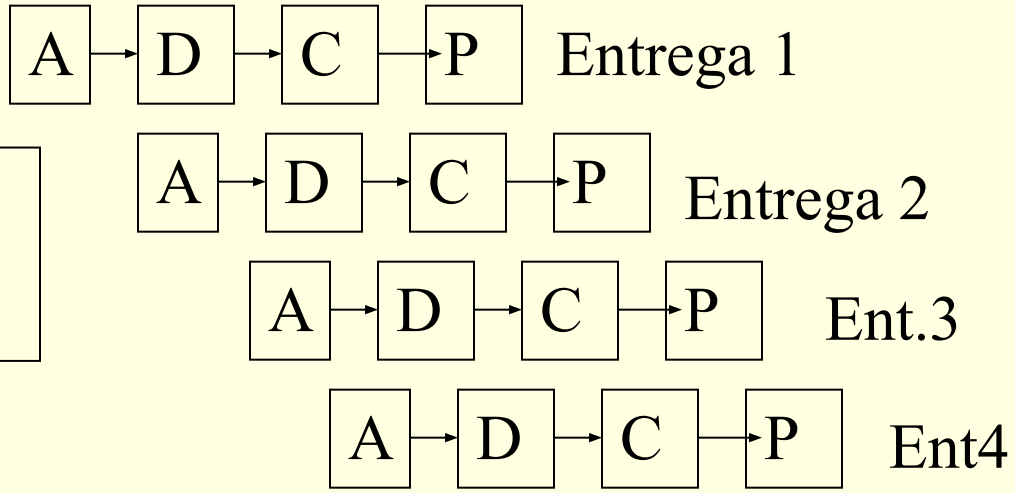
MODELO LINEAL



Conclusiones& Resumen



MODELO DE CONSTRUCCION DE PROTOTIPOS



NUEVO:
MODELO AGIL

MODELO INCREMENTAL

Conclusiones

¿Por qué utilizar uno de los modelos que ya existen ?

¿En qué se concreta el compromiso de calidad?

¿Planificación?

Para planificar el proceso de desarrollo se debe instanciar un modelo de procesos.

Este modelo puede ser propio o tomar uno de los que ya existen.

Sin importar el modelo de proceso que utilicemos debemos basarnos en un compromiso de calidad.