Ingeniería de Software II

Ingeniería de Software II

Segundo Cuatrimestre de 2011

Clase 1b: Modelos de Ciclo de Vida

Buenos Aires, 15 de Agosto de 2011

© Cátedra de Ingeniería de Software II – FCEN – UBA, 2011

¿Qué es un modelo del ciclo de vida de un sistema?

- Una representación estandarizada de:
  - Las etapas de un desarrollo de software
  - Su orden relativo
  - Sus criterios de transición
- Esto sirve para planificar, organizar y ejecutar un proyecto
- Un tema largamente discutido
- Una decisión crítica
- Existen cientos de modelos, la mayoría son variaciones de unos pocos
- Una clave: la visibilidad

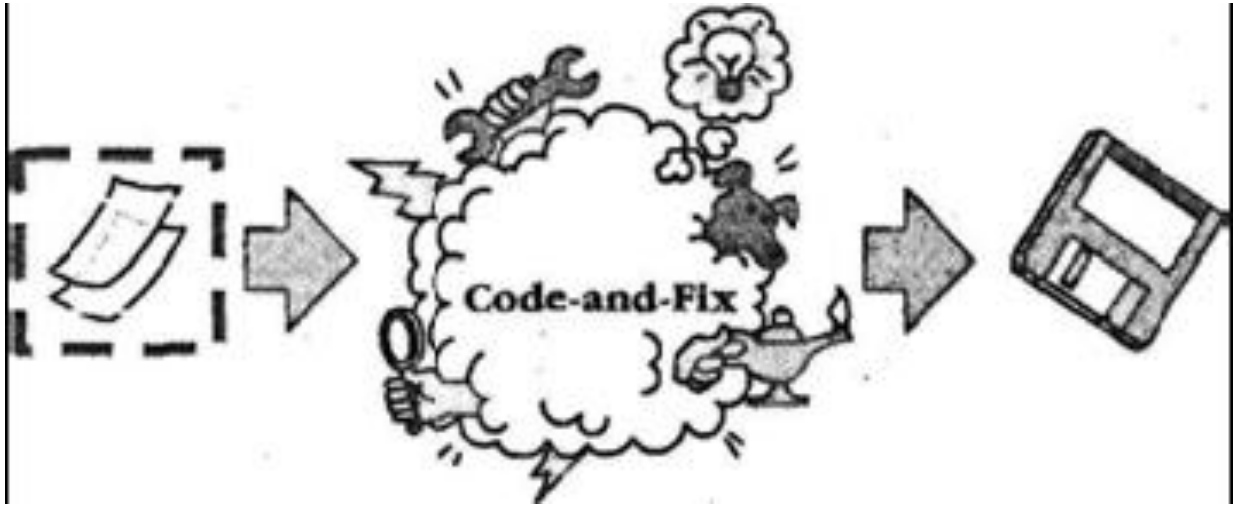
© Cátedra de Ingeniería de Software II – FCEN – UBA, 2011

¿Por qué esto es importante?

- Cambiando el modelo de ciclo de vida se hace un "tradeoff" entre:
  - Velocidad del desarrollo
  - Calidad del producto
  - Visibilidad del proyecto, posibilidad de implementar versiones intermedias
  - Carga de trabajo administrativo
  - Disponibilidad de documentación
  - Exposición al riesgo
  - Relación con el cliente
  - Etc…

© Cátedra de Ingeniería de Software II – FCEN – UBA, 2011

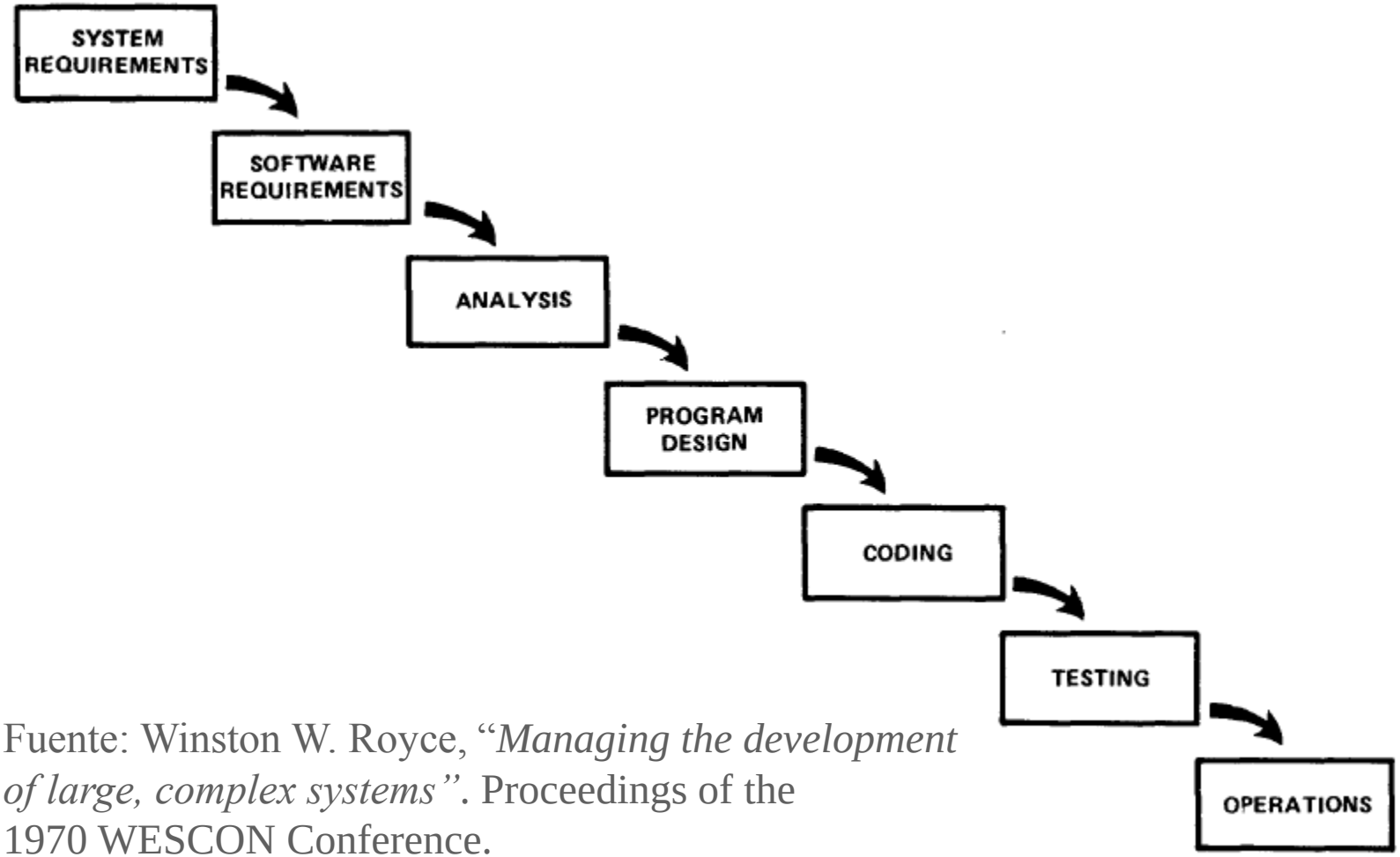
Empezando por el más simple...



Fuente: Steve Mc Connell, Rapid Development, Microsoft Press, 1996

© Cátedra de Ingeniería de Software II – FCEN – UBA, 2011

El Modelo en Cascada "clásico"



Fuente: Winston W. Royce, "*Managing the development of large, complex systems"*. Proceedings of the 1970 WESCON Conference.

Ojo, este excelente paper **NO** propone usar este modelo

© Cátedra de Ingeniería de Software II – FCEN – UBA, 2011

Problemas con el modelo en cascada

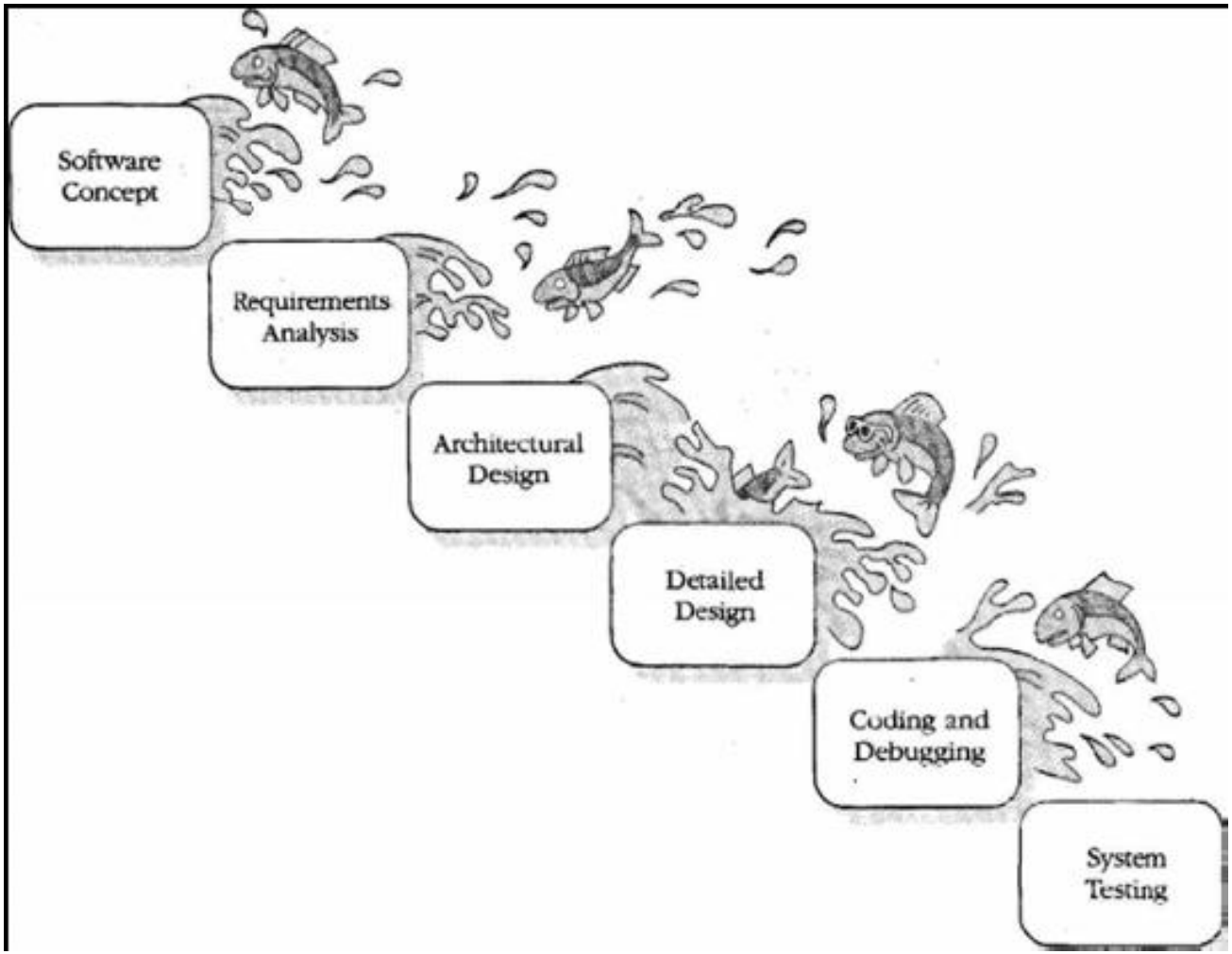
- Se retrasa la detección de problemas críticos
- Idealista pensar en identificar correctamente todos los requerimientos al principio
- No permite implementaciones parciales
- Usuario sólo involucrado al principio y al final

"We have an increasing awareness that system requirements cannot ever be stated fully in advance, not even in principle, because the user doesn't know them in advance – not even in principle. To assert otherwise is to ignore the fact that the development process itself changes the user's perceptions of what is possible, increases his or her insights into the applications environment, and indeed often changes the environment itself. We suggest an analogy with the Heisenberg Uncertainty Principle: any system development activity inevitably changes the environment out of which the need for the system arose. System development methodology must take into account that the user, and his or her need and environment, change during the process."

Life cycle concept considered harmful. Michael A. Jackson and Daniel D. Mc Cracken. ACM Software Engineering Notes. Abril de 1982

© Cátedra de Ingeniería de Software II – FCEN – UBA, 2011

El "Salmon Waterfall"

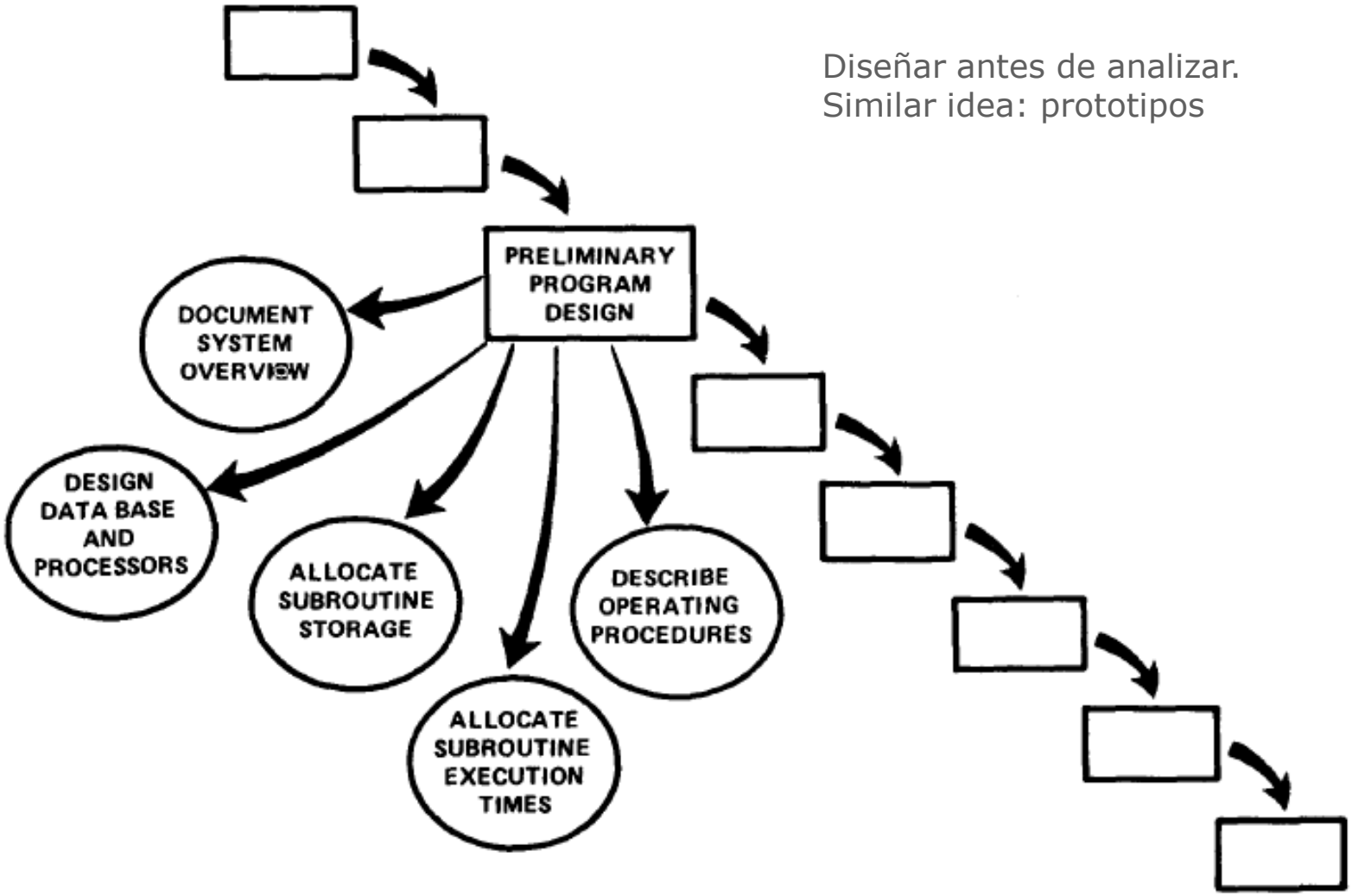


Por sus problemas, así termina siendo el Waterfall en la realidad

Fuente: Steve Mc Connell, Rapid Development, Microsoft Press, 1996

© Cátedra de Ingeniería de Software II – FCEN – UBA, 2011

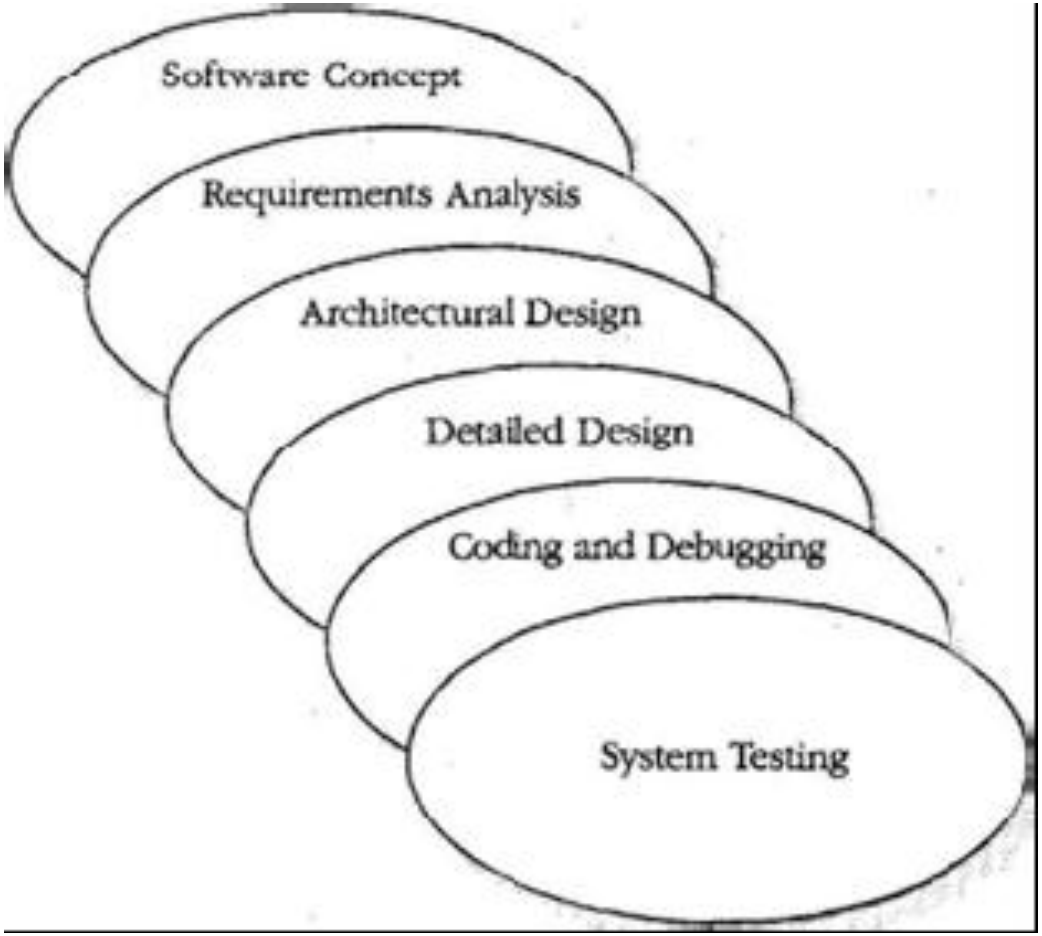
Mejoras del modelo en cascada (Royce)



Diseñar antes de analizar.
Similar idea: prototipos

© Cátedra de Ingeniería de Software II – FCEN – UBA, 2011

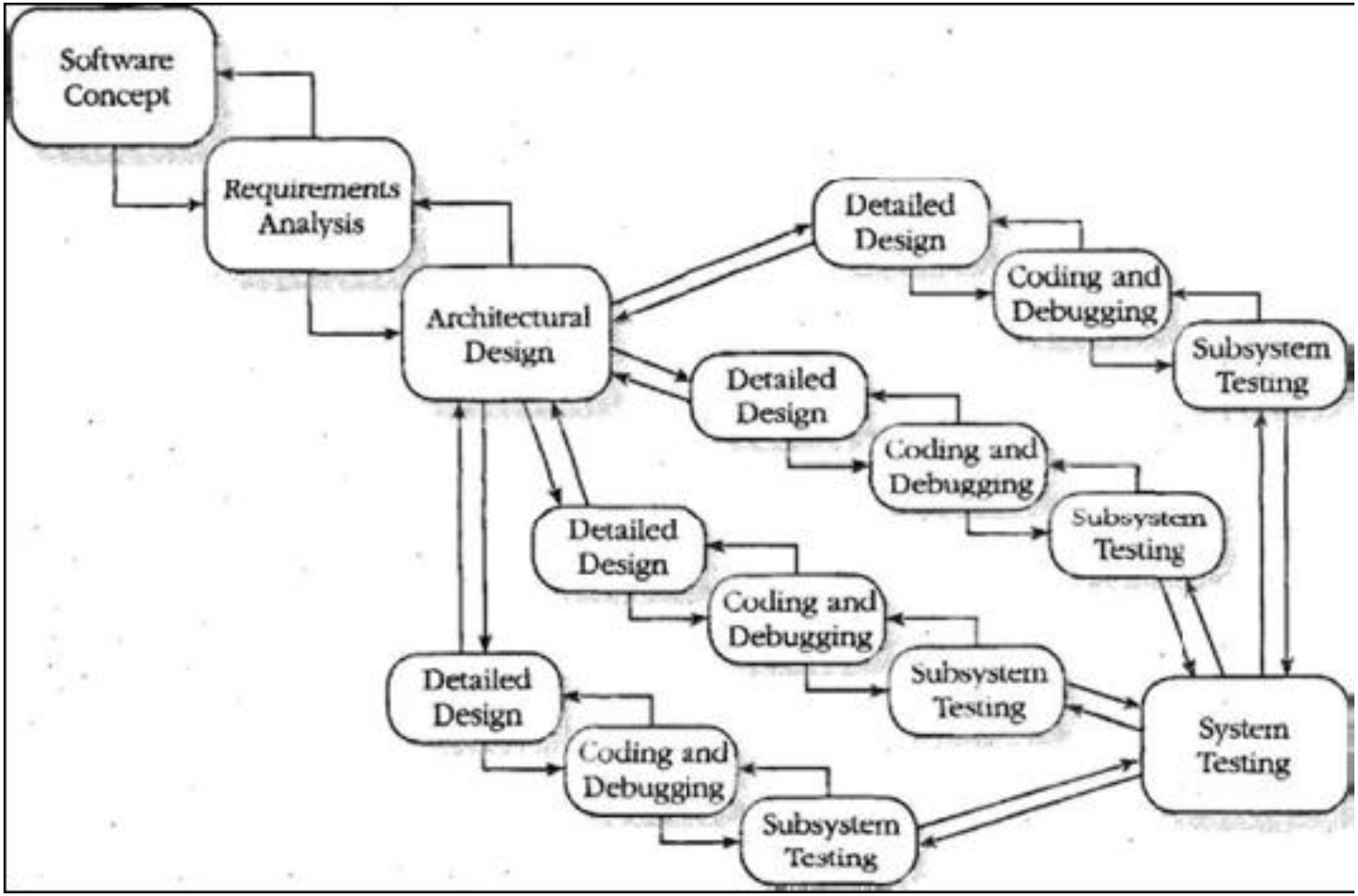
Otras mejoras: Sashimi



Fuente: Steve Mc Connell, Rapid Development, Microsoft Press, 1996

© Cátedra de Ingeniería de Software II – FCEN – UBA, 2011

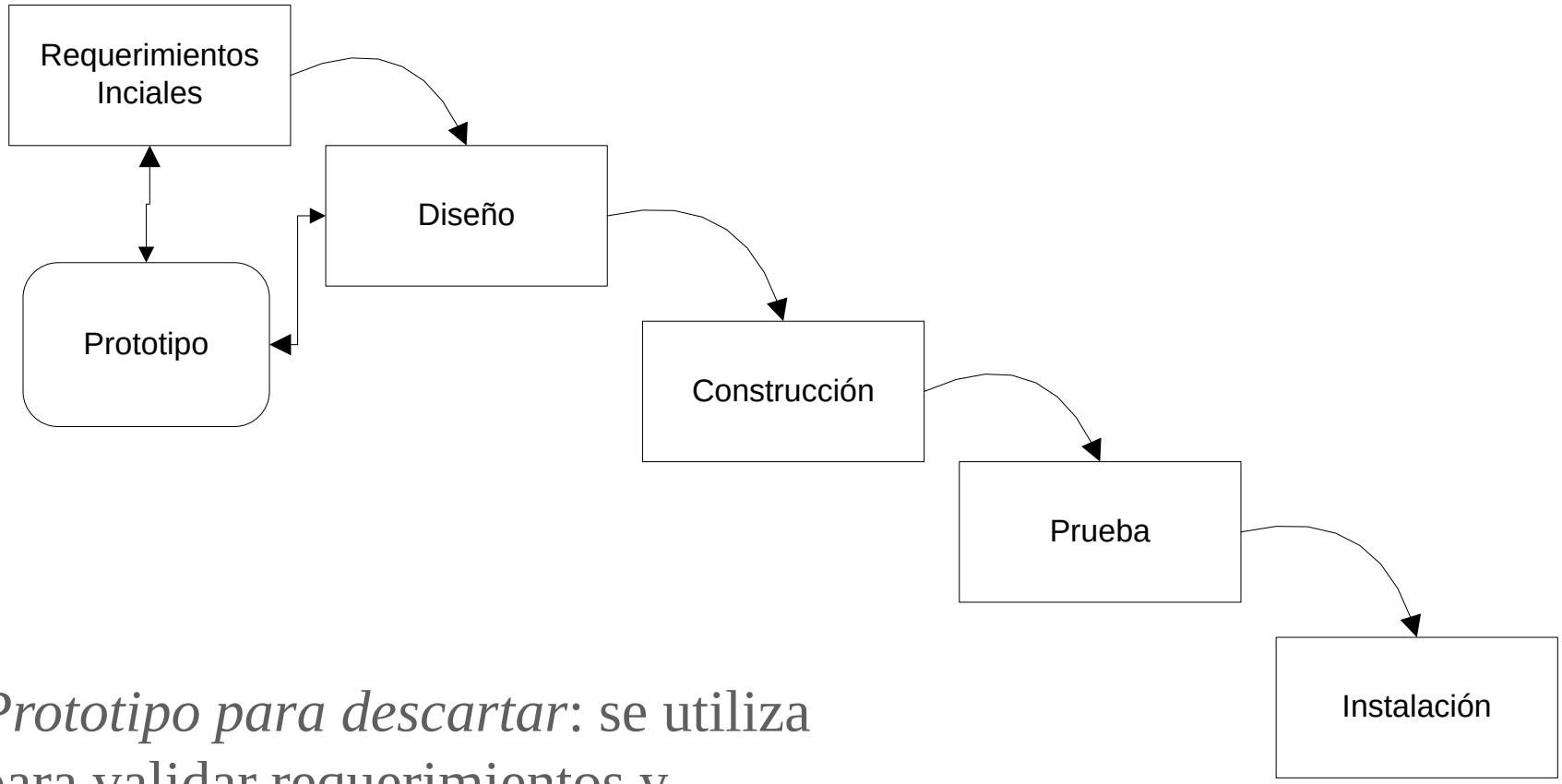
Waterfall con Subproyectos



Fuente: Steve Mc Connell, Rapid Development, Microsoft Press, 1996

© Cátedra de Ingeniería de Software II – FCEN – UBA, 2011

Waterfall con Prototipo (o "risk reduction")

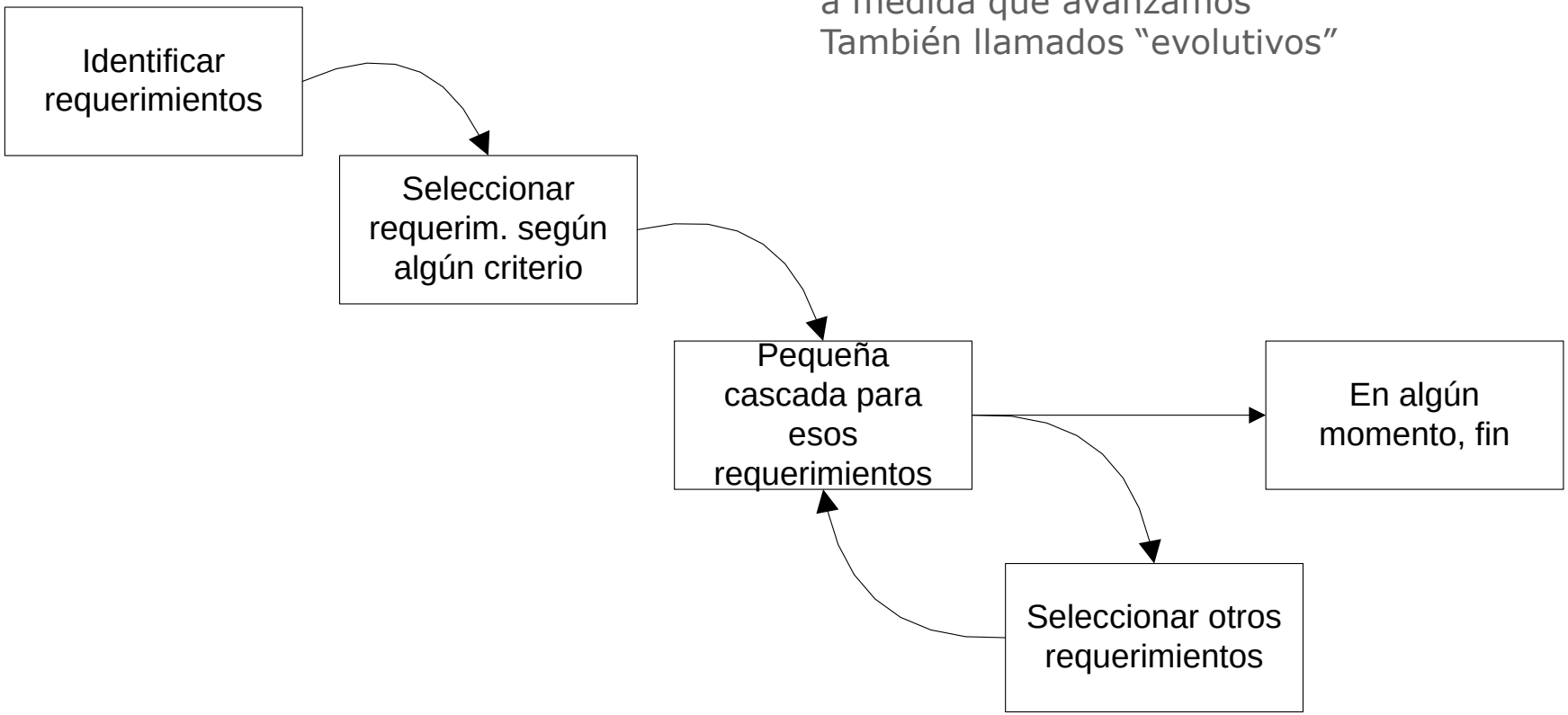


Prototipo para descartar: se utiliza para validar requerimientos y resolver aspectos críticos del diseño

© Cátedra de Ingeniería de Software II – FCEN – UBA, 2011

Modelos iterativos e incrementales

Iterativo: hacemos varias veces lo mismo
Incrementales: El producto se "incrementa" a medida que avanzamos
También llamados "evolutivos"



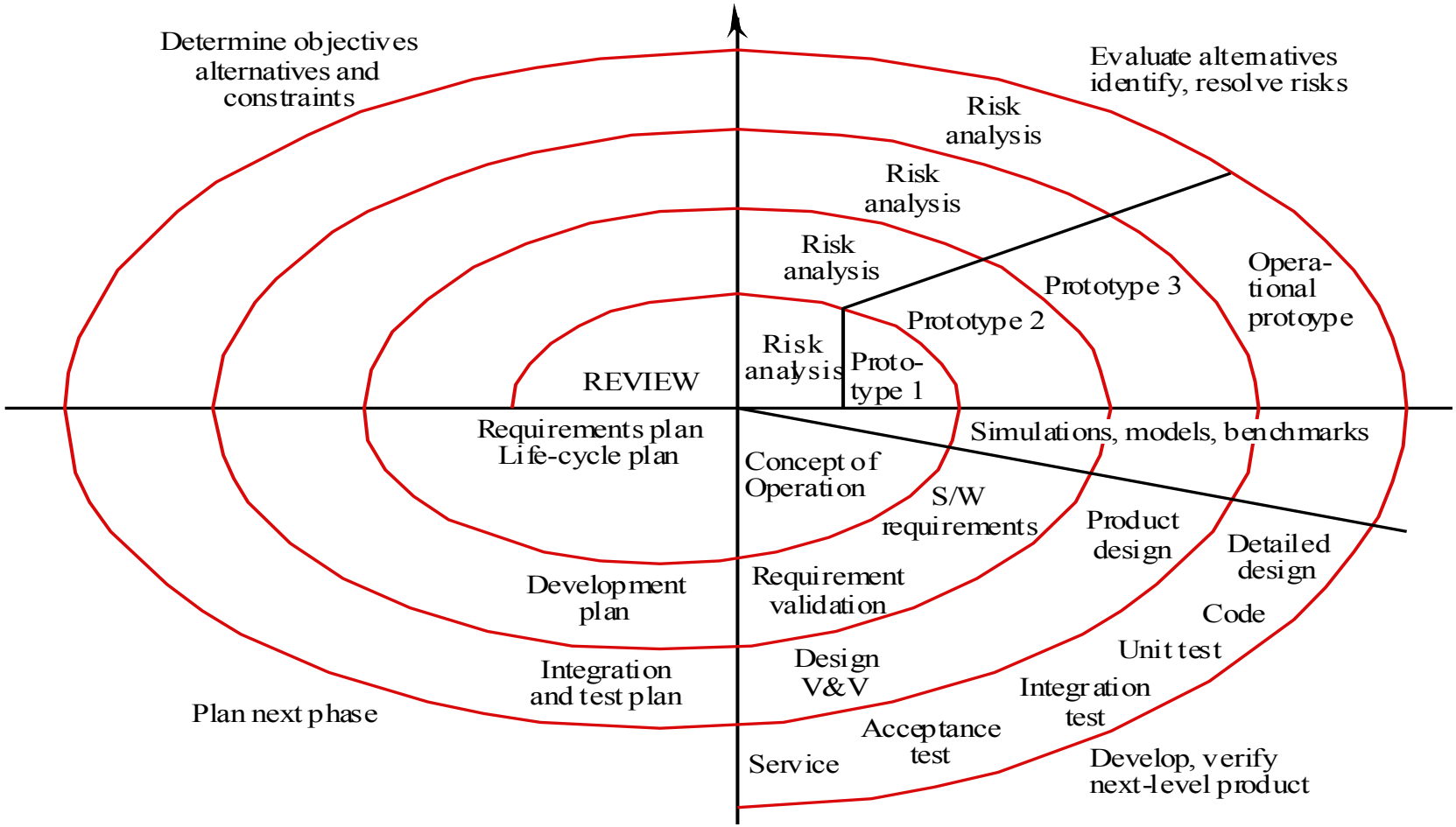
© Cátedra de Ingeniería de Software II – FCEN – UBA, 2011

Modelos iterativos e incrementales

- Principales ventajas
  - El usuario ve algo rápidamente
  - Se admite que lo que se está construyendo es el sistema, y por lo tanto se piensa en su calidad desde el principio
  - Se pueden atacar los principales riesgos
  - Los ciclos van mejorando con las experiencias de los anteriores
- Principales variaciones
  - Duración de las iteraciones
  - Existencia o no de **tipos** de iteraciones
  - Cantidad de esfuerzo dedicado a identificación inicial de requerimientos
  - Actitud "defensiva" ante los cambios

© Cátedra de Ingeniería de Software II – FCEN – UBA, 2011

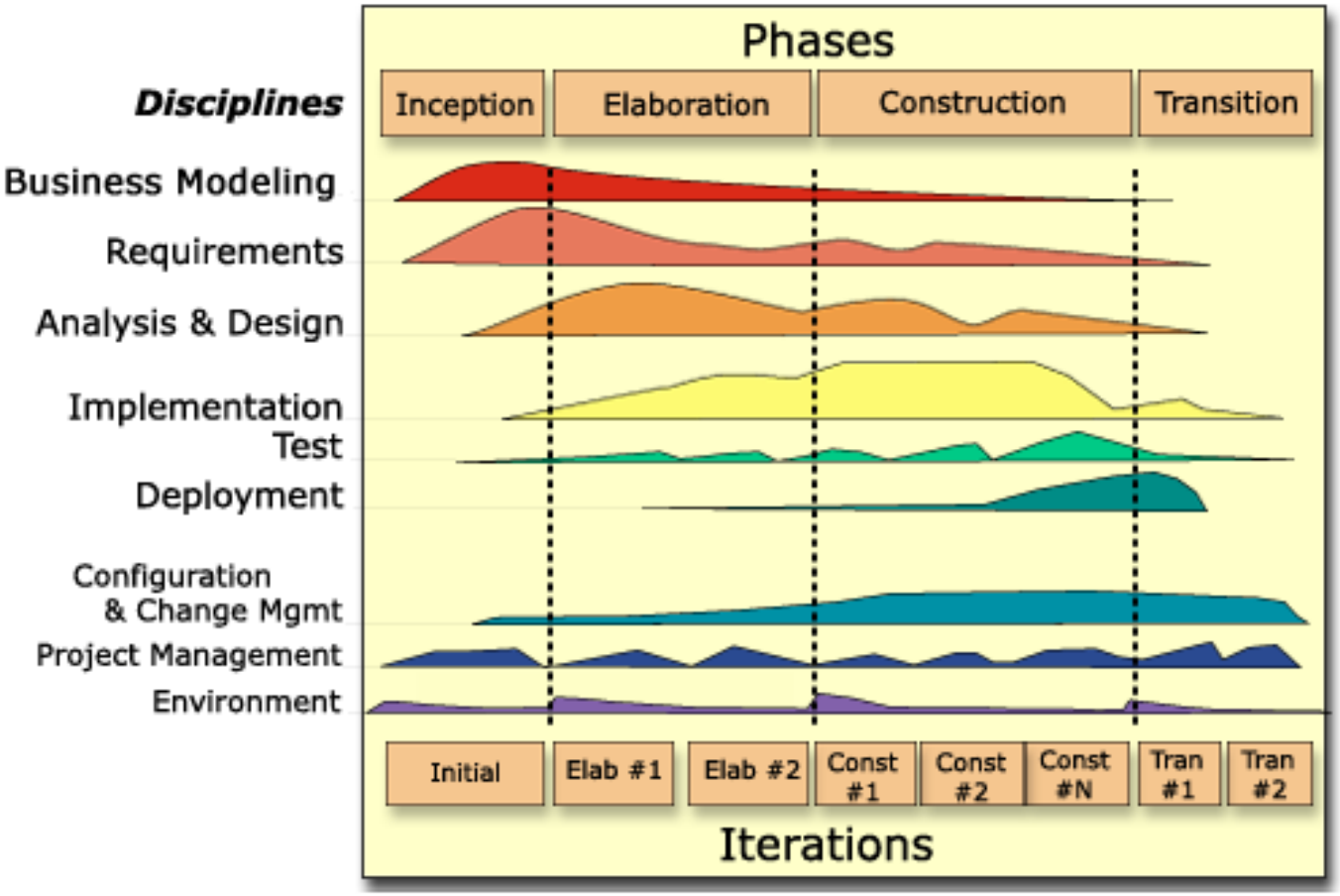
El Modelo en Espiral (Boehm)



El riesgo guía la elección de funcionalidad

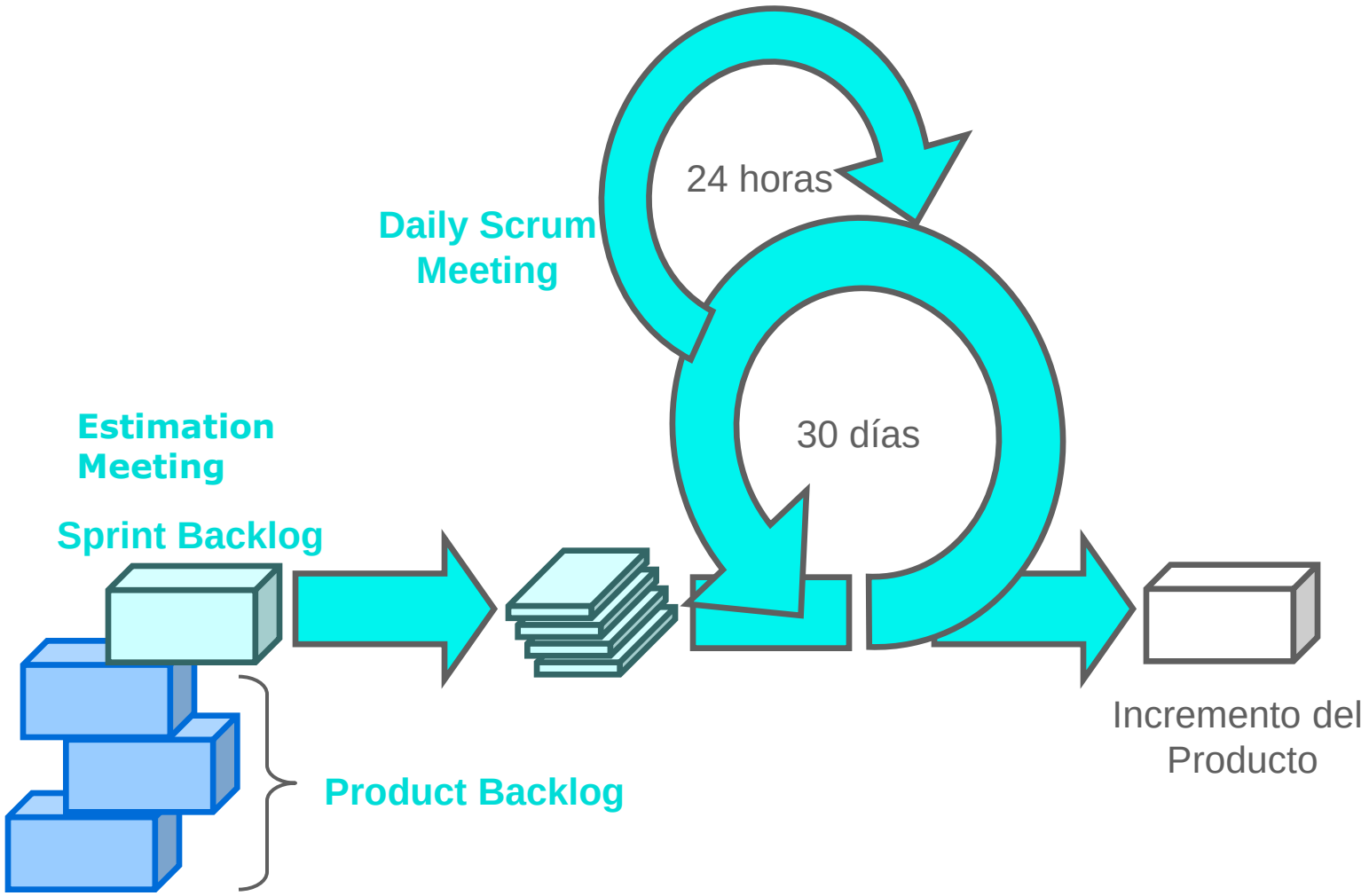
© Cátedra de Ingeniería de Software II – FCEN – UBA, 2011

Variaciones de modelos iterativos – UP / RUP



© Cátedra de Ingeniería de Software II – FCEN – UBA, 2011

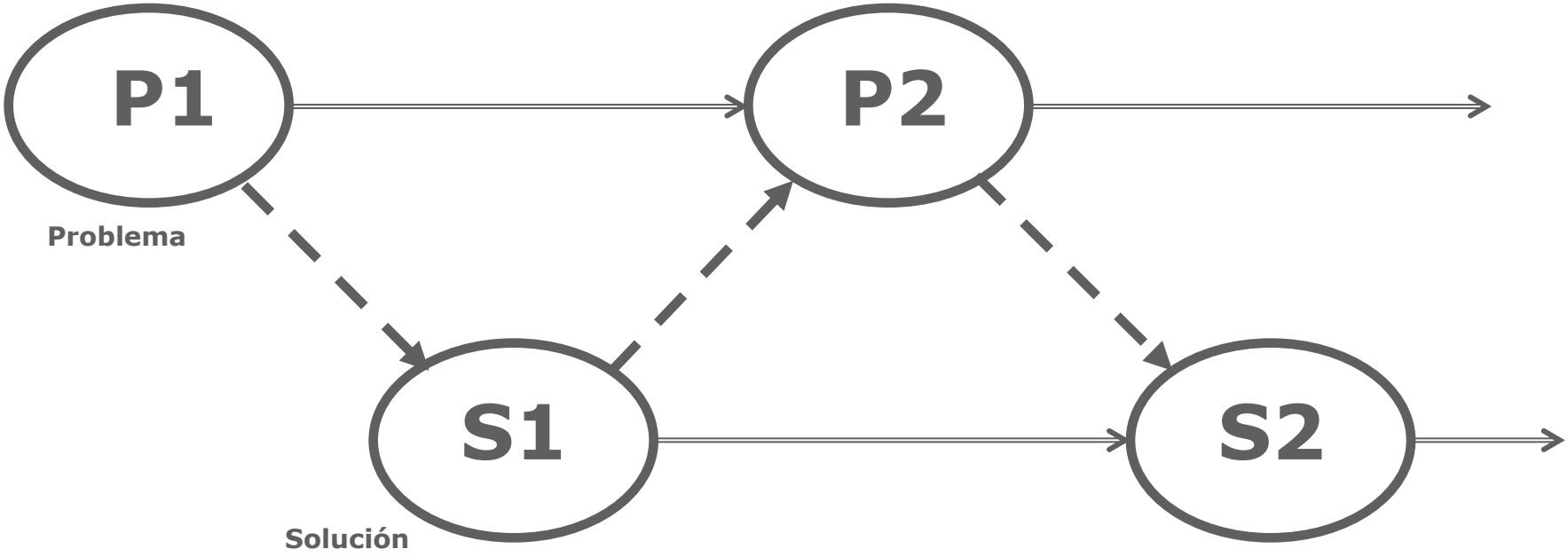
Variaciones – SCRUM



© Cátedra de Ingeniería de Software II – FCEN – UBA, 2011

El concepto clave: co-evolución

- Presentado por Fred Brooks en su conferencia al recibir el Turing Award. Propuesto por Maher – Cross.
- El espacio del problema evoluciona a medida que el espacio de la solución evoluciona al ser explorado… "as one wrestles with the problem"



© Cátedra de Ingeniería de Software II – FCEN – UBA, 2011

En la práctica

- Analizar diferentes factores:
  - Inestabilidad de requerimientos / novedad del producto / Innovación: → Mayor peso al enfoque evolutivo
  - Posibilidad concreta de partir el desarrollo → Iteraciones más cortas
  - Arquitectura más compleja o tecnología no conocida → Enfoque de atacar riesgos desde el inicio, cuidado con la "self emergence"
  - Mayor complejidad del negocio → No descuidar la especificación

© Cátedra de Ingeniería de Software II – FCEN – UBA, 2011