



# Procesos de software

IIC2143 Ingeniería de Software

“Desde ese primer día cuando el primer equipo de desarrollo colaboró para entregar el primer sistema de software económicamente viable, ... ha habido y continúa habiendo una batalla filosófica y pragmática sobre cómo organizar mejor un equipo y su trabajo.

En un extremo están los que defienden procesos muy formales, artefactos rígidamente definidos y una estricta ordenación de actividades; ... en el otro, los que prefieren procesos poco formales, muy focalizados en la codificación, y que consideran todo lo demás irrelevante o inconsecuente.

El péndulo ha oscilado entre estos dos extremos por años.”

Grady Booch, 2006


## **Construir y mantener software es difícil ( y **no** se está volviendo más fácil )**

A pesar de la creciente importancia del software,

... un proyecto típico es difícil de gestionar:

- se exceden los plazos
- se exceden los presupuestos
- no se entrega la funcionalidad acordada o que realmente se necesita

**Esto se traduce en oportunidades de negocio perdidas  
o empresas que fracasan**



## El problema está en la forma en que desarrollamos software ( este es un supuesto )

El **proceso** que seguimos en cada proyecto

Los **métodos** que empleamos en cada proyecto

La forma en que **gestionamos** cada proyecto

## Un proceso de desarrollo de software cumple cuatro roles

( esta es una de muchas definiciones posibles )

- 1) Proporciona una guía para el orden de las **actividades** del equipo
- 2) Especifica qué **artefactos** deben ser desarrollados y cuándo
- 3) Dirige las **tareas** de los desarrolladores individuales y del equipo como un todo
- 4) Ofrece **criterios** para supervisar y medir los productos y las actividades del proyecto

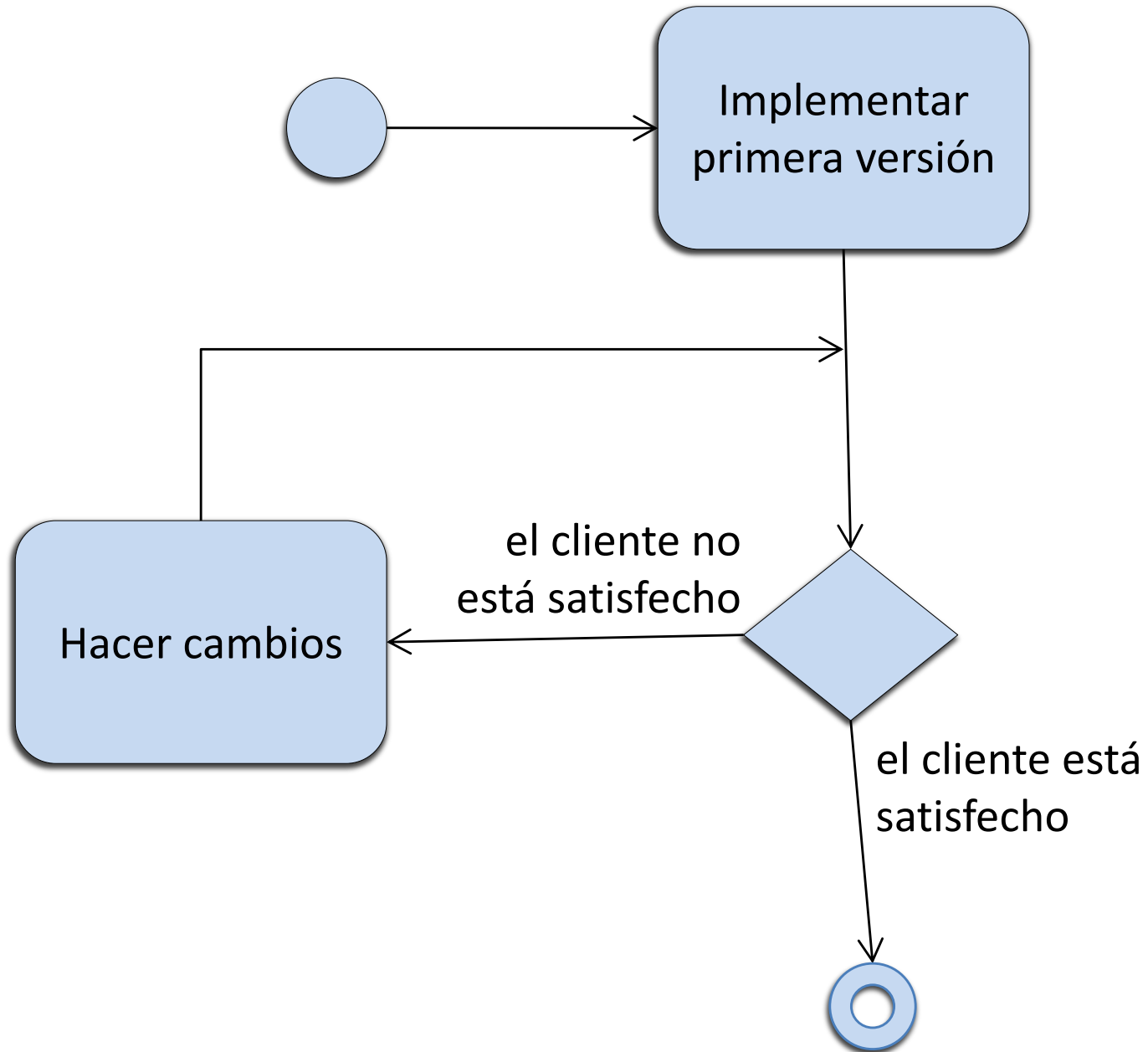


## **Sin un proceso definido, el desarrollo de software no es sustentable** ( esto está comprobado empíricamente )

El equipo desarrolla de manera ad hoc

El éxito depende de los esfuerzos heroicos de unos pocos individuos dedicados

La próxima diapositiva muestra visualmente, mediante un *diagrama de actividades* del UML, este tipo de desarrollos





# Una instancia del caso anterior es el desarrollo “big bang”

El cliente: “Esto es más o menos lo que necesito, para comenzar.”



# Las organizaciones exitosas emplean procesos bien definidos

( esto también está comprobado empíricamente )

Pueden desarrollar sistemas complejos de manera **repetible**:

- aplican el mismo proceso en varios proyectos

... y **predecible**:

- pueden predecir los resultados del proyecto (duración, costo, funcionalidad, calidad, etc.)

El negocio mejora con cada nuevo proyecto, aumentando la eficiencia y la productividad de la organización —el negocio es sustentable

## **Los procesos bien definidos facilitan y promueven las prácticas eficaces de la ingeniería de software**

Al codificar estas prácticas en un proceso, se puede aprovechar la experiencia colectiva de muchísimos proyectos exitosos

## Pero las organizaciones exitosas no son la mayoría

La mayoría de las empresas **no están preparadas** para crear (desarrollar, construir) el software que

- define cómo interactúan con los clientes
- las ayuda a entregar sus productos y servicios

## **Los procesos de desarrollo de software son una parte vital de la cadena de valor del negocio**

La capacidad de desarrollar software debe ser entendida como una competencia central del negocio

...si queremos aprovechar los beneficios de la innovación:

- los proyectos de software debe ser entendidos como propuestas de valor que fortalecen la competitividad de la compañía

## ¿Qué estamos haciendo?

Varias organizaciones profesionales dedican muchos recursos a mejorar la gestión de proyectos de software:

- el PMBOK del PMI
- las mejores prácticas en gestión de proyectos de software de la IEEE
- el CMMI del SEI
- el RUP de IBM Rational

## Otras organizaciones promueven varios métodos de desarrollo

... cada uno con méritos para equipos particulares

... según tamaño, estilo, experiencia colectiva:

- *Agile Alliance*
- *Scrum Alliance*
- la comunidad *eXtreme Programming (XP)*
- *OpenUP*

## ¿Qué hacen otras industrias?

Los profesionales del software consultamos pautas generales (no específicas para la industria del software) sobre gestión de proyectos:

- *Six Sigma*, de Motorola —conjunto de prácticas que apuntan a la reducción de defectos
- *Lean Manufacturing*, de Toyota —concepto asociado a la eliminación de desperdicios en los procesos de manufactura



# Dos tipos principales de modelos de procesos

## Modelos prescriptivos:

- nos dicen cómo algo debería ser hecho
- procuran hacerse cargo de todos los problemas relevantes del desarrollo de una pieza de software
- están basados en modelos ideales de desarrollo de software, p.ej., un conjunto de mejores prácticas que deberían ser aplicadas en los proyectos

## Modelos descriptivos:

- describen cómo algo es hecho en realidad
- los creamos a partir de observar los procesos que efectivamente son llevados a cabo

Pueden existir en paralelo en una organización de software

No difieren necesariamente en su contenido, sino en su propósito

## Dos tipos de modelos prescriptivos

### Modelos del ciclo de vida:

- modelan el ciclo de vida completo de un producto de software
- hacen abstracción de muchos detalles
- se enfocan en el “qué” y no tanto en el “cómo”

### Modelos de ingeniería:

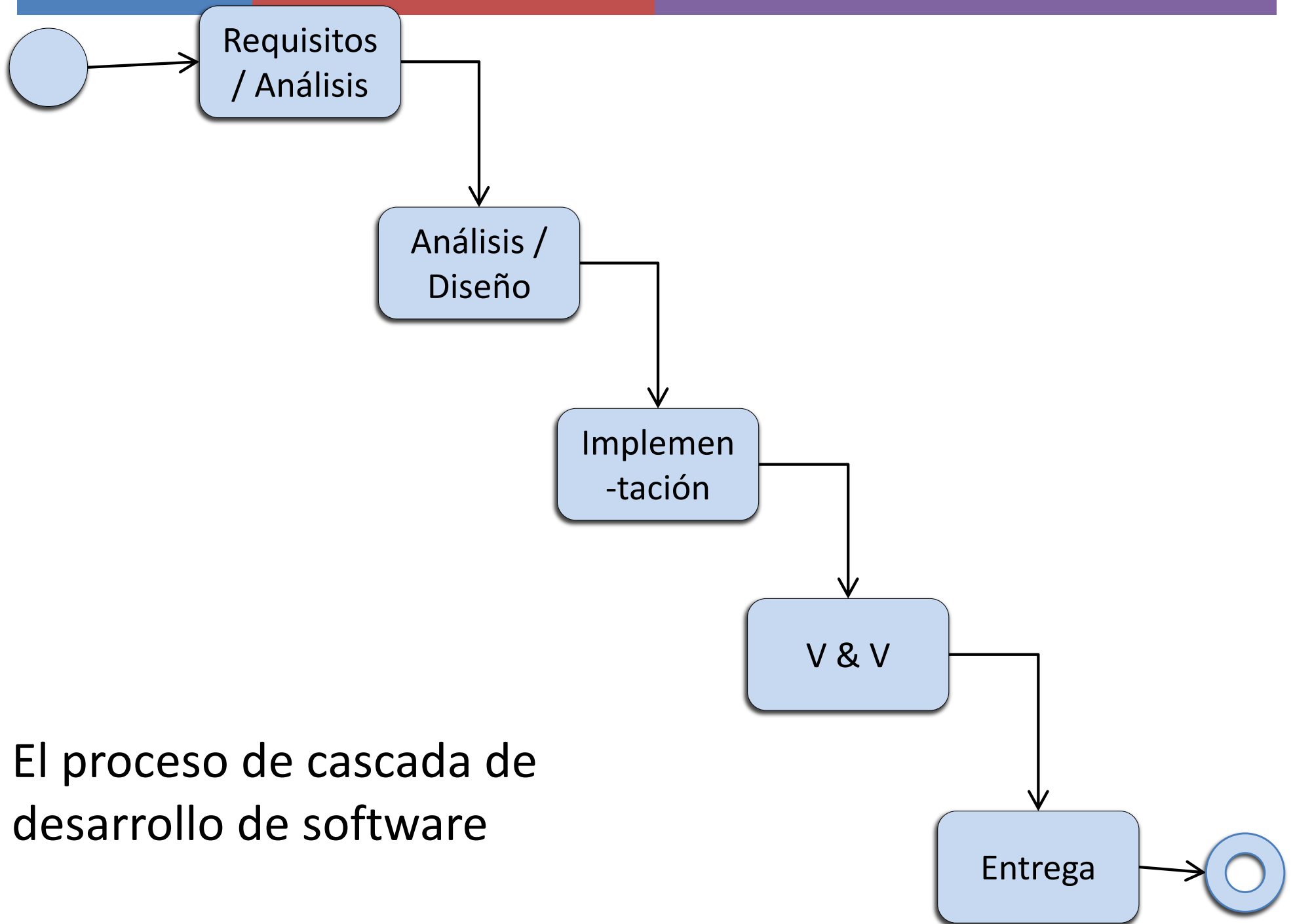
- describen en mucho más detalle una parte del proceso de ciclo de vida
- no sólo describen “qué” hacer, sino también explican “cómo” hacerlo

Uno de los modelos de proceso del ciclo de vida prescriptivos más conocidos es el modelo (o proceso) de cascada (o *waterfall*)

## El Proceso de Cascada: Cómo resolver (muchos) problemas de ingeniería

Sugiere pasos o **fases** similares a los siguientes:

- entender completamente el problema, sus requisitos y restricciones, escribirlo y conseguir que todos los interesados estén de acuerdo
- diseñar una solución que cumpla con todos los requisitos y restricciones, y asegurarse que todos los interesados estén de acuerdo
- implementar la solución
- verificar que la implementación cumple con los requisitos establecidos
- entregar: ¡El problema está resuelto!



El proceso de cascada de desarrollo de software

## **El proceso de cascada está basado en varios supuestos**

( no necesariamente ciertos en todos los proyectos )

Los productos pueden ser desarrollados completamente en “una pasada”  
—no es necesario revisar, deshacer, rehacer

Todos los requisitos son igualmente importantes, y pueden ser identificados y quedar congelados al inicio del proyecto

Los diseños pueden ser verificados sin construir y probar algo

El proyecto puede ser planificado en detalle con un alto grado de precisión desde el inicio

Todo lo importante puede ser conocido al inicio del proyecto

Si seguimos los pasos prescriptivos del proceso, todos los riesgos del proyecto serán abordados eficazmente

## El proceso de cascada es razonable en la ingeniería tradicional

Es una forma racional de proceder:

- es como se construye los rascacielos, los barcos y los satélites de comunicaciones

... que está basada en las leyes de la física (Newton, Maxwell),

... y las propiedades de los materiales,

... que son disciplinas establecidas

También está basada en la madurez de los códigos y prácticas de construcción y manufactura

## **... y, en algunos casos, también en desarrollo de software**

Funciona bien en proyectos de software en los que se puede hacer y seguir un plan:

- proyectos cortos
  - ... en los que podemos anticipar lo que va a pasar
  - ... y entendemos bien los aspectos difíciles
- proyectos parecidos a los que hemos hecho antes,
  - ... en que empleamos la misma gente, herramientas y diseños

## **El proceso de cascada no funciona tan bien en la ingeniería de software actual**

C. Larman [“Agile and Iterative Development: A Manager’s Guide”, Addison-Wesley 2003] muestra que su uso ha sido una mala práctica:

- su tasa de éxito es baja (sólo un 10%)
- en promedio, el 45% de las propiedades identificadas en la etapa de requisitos no son usadas
- las estimaciones iniciales de costos y plazos varían hasta en un 400% con respecto a los valores reales finales
- presenta menor productividad y mayores tasas de errores que los procesos iterativos



## Varias razones explican los malos resultados de esos proyectos

El proceso de cascada no toma en cuenta la creatividad que se necesita para llevar adelante proyectos que parten con **niveles de incertidumbre importantes** con respecto a aspectos esenciales:

- el **problema** —qué quiere o necesita realmente el usuario
- la **solución** —qué arquitectura y combinación de tecnologías es la más apropiada
- la **planificación** (para ir del problema a la solución) —restricciones de presupuesto y tiempo, composición del equipo, comunicación entre los interesados, las fases más apropiadas, etc.

## Desarrollo de software: Un dominio de cambios e inestabilidad

El **cambio** es la constante en los proyectos de software

En el proceso de cascada hay al menos **dos suposiciones que no son ciertas**:

- los requisitos son predecibles, estables y pueden ser definidos correctamente al comienzo, con bajas tasas de cambio —podemos “congelar” los requisitos
- podemos hacer un diseño correcto, completo, eficiente y factible de la solución antes de implementarla

## **Supusimos que los requisitos son predecibles y estables**

Cambian los usuarios, el problema, la tecnología subyacente y el mercado

No podemos capturar requisitos con suficiente detalle y precisión

Un proyecto típico experimenta entre un 25% y un 50% de cambios en los requisitos:

- B. Boehm, P. Papaccio, Understanding and controlling software costs, "IEEE TSE" 14(10) Oct. 1988
- C. Jones, "Applied Software Measurement", McGraw-Hill 1997

## **Supusimos que podíamos hacer un diseño correcto y completo antes de la implementación**

Tal vez ... si aplicásemos técnicas tales como seguimiento de requisitos, derivaciones formales, demostraciones automáticas, simulaciones de diseño

Pero estas técnicas no están fácilmente disponibles, y muchas requieren empezar con una definición formal del problema

Las teorías subyacentes son débiles y no las entendemos bien

## **La ingeniería de software no es como otras ingenierías**

Para construir un puente, una carretera o un edificio, podemos basarnos en cientos de ejemplos muy similares:

- nos conviene construirlo exactamente como el último, para eliminar los riesgos del proyecto

En el caso del software, si alguien ha desarrollado un sistema como el que necesitamos, nos conviene comprar la licencia:

- no es buena idea gastar dinero en desarrollar software que podemos comprar

## **El software es el dominio del desarrollo de productos nuevos**

Los proyectos de desarrollo de software que se llevan a cabo son aquellos que no han sido hechos antes

... o no están disponibles comercialmente

Por lo tanto, el desarrollo de software es difícil y riesgoso

... y, por lo tanto, el proceso es muy importante

## El desarrollo de software es una actividad riesgosa

El proceso de cascada no funciona bien para proyectos con un nivel importante de novedad, incógnitas y riesgos:

- no podemos anticipar las dificultades que vamos a encontrar, menos aún cómo las enfrentaremos
- faltan las “leyes fundamentales del software”
- sólo queda agregar holguras a los plazos

Hay que incorporar el **análisis de riesgos**

## **Un problema más general con todos los modelos de procesos**

Las personas tendemos a ser reacias a aplicar prácticas (o procesos) que percibimos como aburridas, innecesariamente complicadas, o cargadas de burocracia inútil:

- especialmente, si demandan trabajo que no parece beneficiarnos directamente



El problema no es sólo construir un modelo de proceso,

... sino también **hacer que la gente lo siga como se pretende** —un problema más difícil (y a menudo ignorado):

- el modelo nos dice que tenemos que hacer las cosas de manera diferente,
- ... pero conseguir que la gente cambie su conducta es una de las tareas más difíciles en ingeniería de software

La puesta en práctica exitosa de un modelo de proceso es difícil porque implica cambiar la conducta de la gente

## 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]

**Scrum** [K. Schwaber & J. Sutherland, 1996]

**XP** [K. Beck, 1999]