

# Desarrollo de software y programación

Carlos Fontela  
cfontela@fi.uba.ar



# Temario

Desarrollo de software

Disciplinas del desarrollo

Programas y sistemas

Problemas de los proyectos de desarrollo de software

Metodología



# Desarrollo de software

Principal ocupación de los egresados de carreras informáticas de FIUBA

Al menos los primeros años

Desarrollo  $\neq$  Programación

Desarrollo de software incluye a la Programación

Pero también a otras disciplinas



# Mini-historia del desarrollo

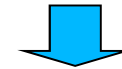
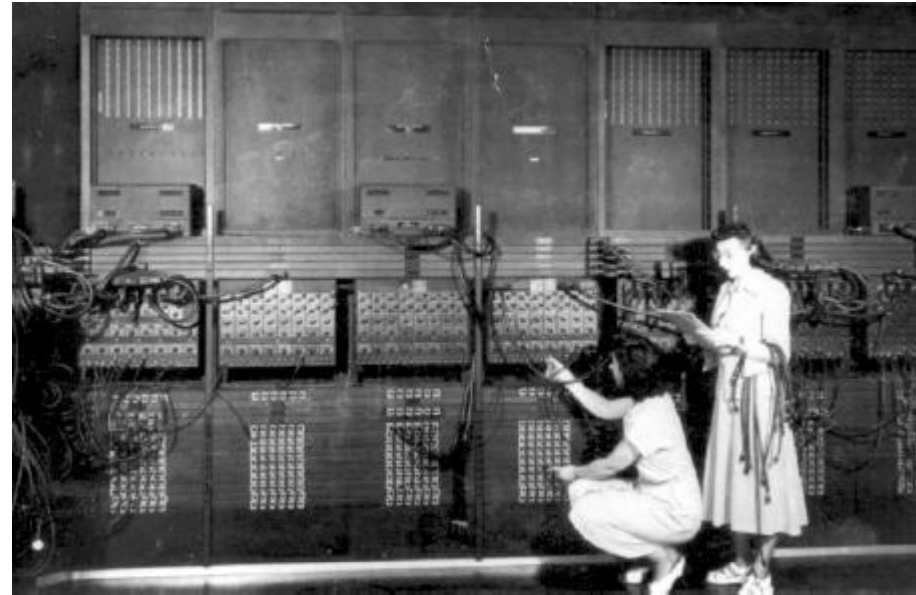
1945: ENIAC, programas cableados

1954: lenguajes de programación

1960s: aumenta la complejidad

1970s: sigue aumentando, búsqueda de un proceso, una “ingeniería”

> 1980: sigue aumentando, búsquedas en ingeniería y administración de proyectos



# Mini-historia de carreras argentinas

1961: llega Clementina a la FCEN

1960s: carrera de Computador Científico en FCEN

Departamento de Matemática  
Muy centrada en programación

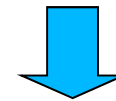
1970s: carrera de Analista Universitario de Sistemas en FIUBA

Centrada en problemas ingenieriles y cuestiones organizacionales

1990s: carrera de Ingeniería Informática en la FIUBA

Y cambio de nombres de las carreras antiguas: licenciaturas

1990s: primera Facultad de Informática de la Argentina: UNLP



# Disciplinas de desarrollo de software

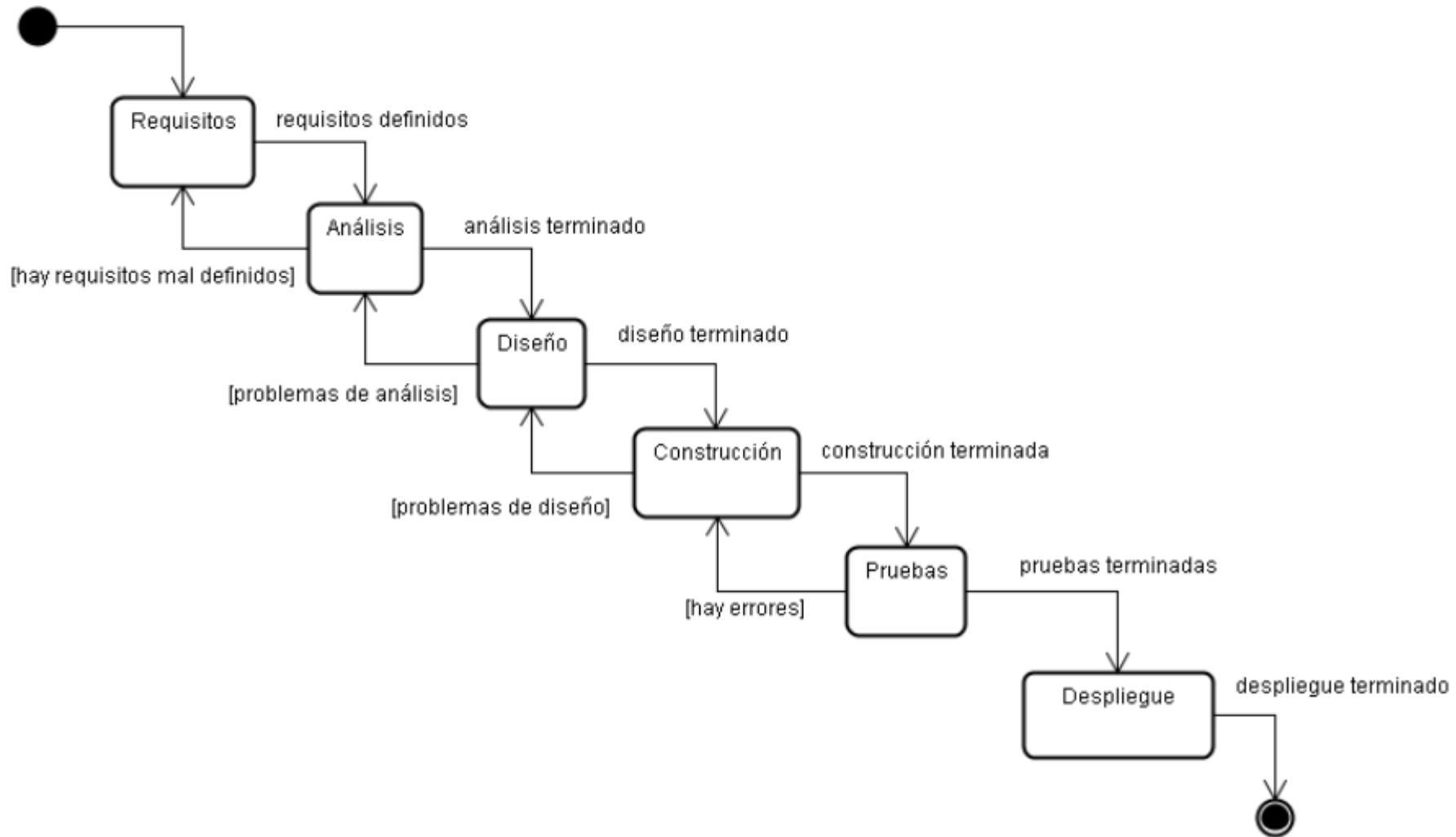
No todo es programación en el desarrollo de software

Muchas visiones

En la década del 70 se definieron, asociadas a un método, hoy llamado “cascada”



# Desarrollo en cascada



# Disciplinas del desarrollo (operativas)

Captura de requisitos: qué quiere el cliente

Análisis: qué vamos a construir

Diseño: cómo

Implementación o construcción

Pruebas: verificación y validación

Despliegue (en hardware)



# Disciplinas del desarrollo (soporte)

Administración del proyecto, incluyendo  
seguimiento y control

Gestión de cambios

Administración de la configuración

Gestión de los recursos humanos

Gestión del ambiente de trabajo

Gestión de la calidad



# Requisitos

Muy difícil

Nunca se puede anticipar totalmente la funcionalidad de un producto

Cliente: “No sé lo que quiero, pero si me mostrás algo, te digo por qué no me gusta”

Requisitos != Expectativas != Necesidades

No se puede congelar requisitos ni tan siquiera en un proyecto de mediano porte



# Análisis

Sistema a construir

Incluye cuestiones tecnológicas que sean requisitos

Requisitos  $\neq$  Expectativas  $\neq$  Necesidades

Tratar de concentrarse en las necesidades

Se obtienen especificaciones funcionales de la aplicación

Se deben acordar con el cliente



# Diseño

Actividad eminentemente ingenieril

Determinamos cómo resolver el problema

Muchos aspectos técnicos

Algo veremos en Algoritmos III

Hay materia específica



# Pruebas

## Validación

Que el sistema haga lo que el cliente espera

## Verificación

Que el sistema haga lo que dicen las especificaciones

Hay muchos tipos de pruebas

Complementarias

Tema de Algoritmos III y otras materias



# Disciplinas y cascada

No siempre están atadas

Ni tienen que estarlo

Es un tema que abordaremos luego



# Programas y sistemas

Sistema  $\neq$  Programa

tanto como Desarrollo  $\neq$  Programación

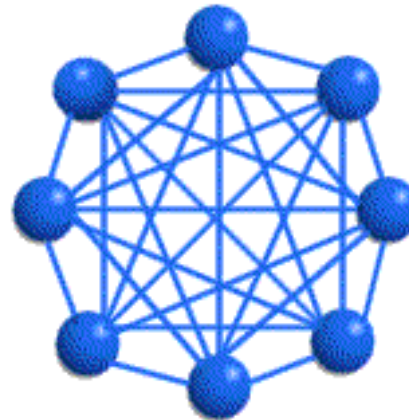
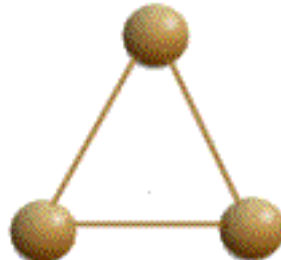
Sistema:

Conjunto de módulos interrelacionados

Puede no haber un “programa principal”

Pueden ser muchos programas comunicándose

Complejidad aumenta con el número de partes



# Software y hardware

Software =

Programas +

Datos +

Documentación +

Conocimiento y reglas del dominio del problema

El software controla al hardware

Y lo hace útil

Cada vez hay más aparatos controlados por software

Y cada vez más funciones del aparato



# Características del software (1)

Intangible

Maleable (“soft”)

No necesariamente implica facilidad de cambio

Sí posibilidad

Se desarrolla por proyectos

Diferencia con otros productos industriales

Parecidos con la industria de la construcción

Alto contenido intelectual

Generalmente disperso y difícil de reunir

Diseñado y construido por profesionales



# Características del software (2)

¿Producción de software?

Si hay “producción” de música y películas...

Es una actividad humana

El software no se “fabrica”

Se **construye** o se **desarrolla**

Mantenimiento constante

Desde su construcción

“Como con algunos parientes, es difícil deshacerse de algunos productos de software”  
(Freeman)



# Fracasos del desarrollo de software (1)

## Proyectos que no terminan a tiempo

Aeropuerto de Denver: sistema de administración de equipajes

agosto 1994 => diciembre => marzo => mayo  
pérdidas de U\$S 1 M por día de atraso

## Proyectos que cuestan más de lo estimado

PPARS (proyecto de administración de personal, Irlanda)

Estimado en € 8,8 M => € 140 M



# Fracasos del desarrollo de software (2)



## “Accidentes”

### Software del Ariane 5

Explota a poco de salir por pérdida total de información de guiado y altitud

Origen: uso de software del Ariane 4

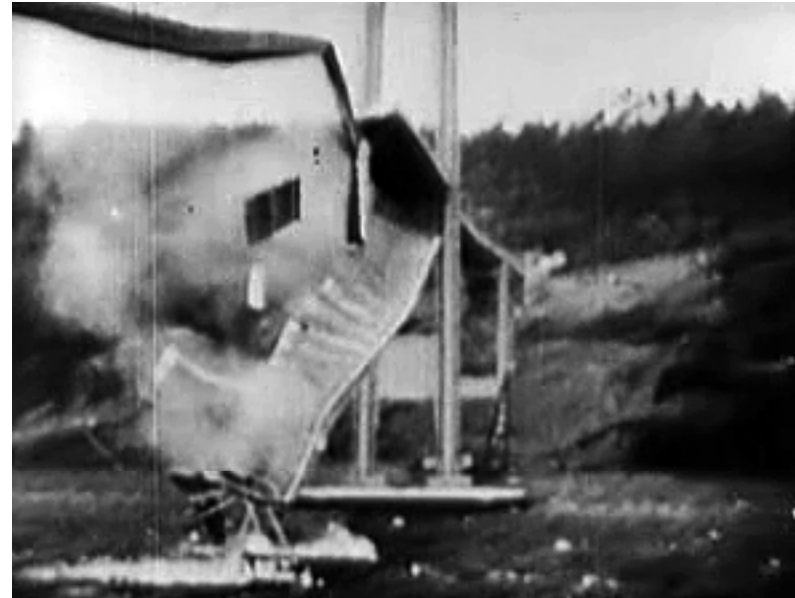
Accidente con pacientes oncológicos en Panamá

## Productos que no cumplen lo que el solicitante quiere

45% del software contratado nunca es usado

# Problemas en otras ingenierías de proyectos (1)

Tacoma – Narrows  
Colapsó en 1940



Puente de Aviñón  
Construido en 1171  
Destruído varias veces  
Último intento 1660



# Problemas en otras ingenierías de proyectos (2)

Accidentes en Three-Mile Island y Chernobyl

Yacyretá-Apipé

US\$ 11.000 M en 15 años

Puente Chaco-Corrientes

Fallas de diseño (1973)

Big-Dig, Boston, EEUU

2,8 MM => 14 MM

Catedral de Colonia, Alemania

1248-1880

Reparación permanente



# 2 casos propios

1987

Tecnologías desconocidas

Poca experiencia en desarrollo

Pruebas al final

1 mes => 3 meses

2007 (¡20 años más tarde!)

Fecha de entrega fija

Demoras en comienzo

Interacción con otro proveedor

Problemas de equipo

2 meses => 6 meses

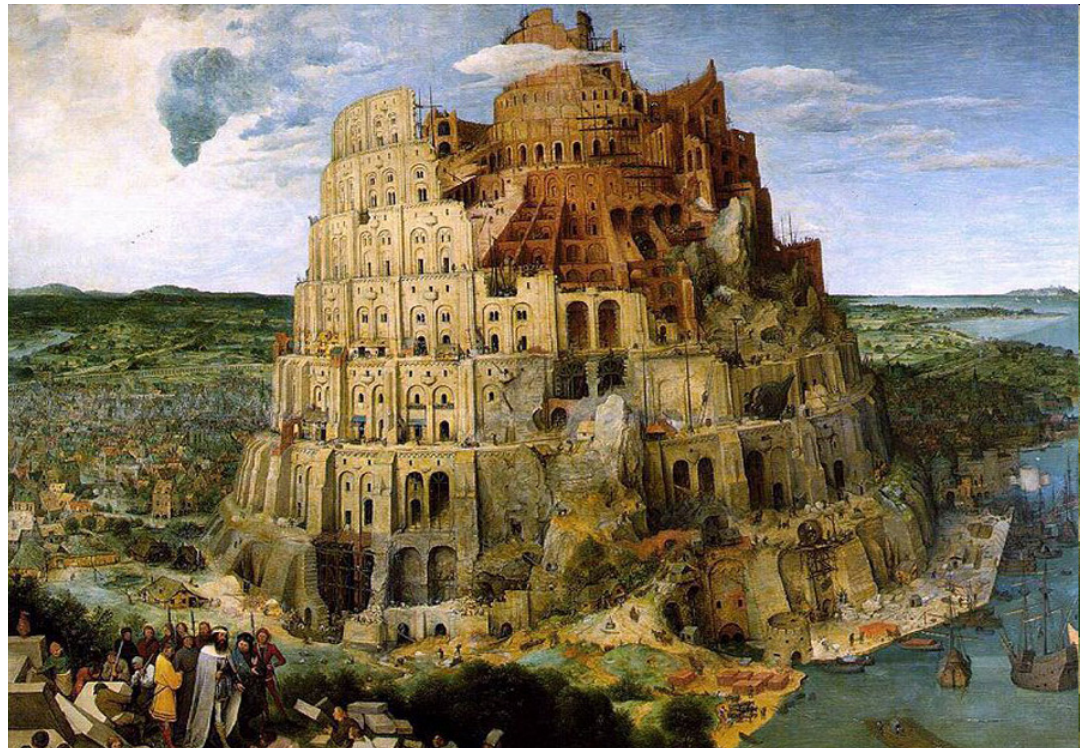


# Algunas reflexiones

Los problemas del desarrollo no son sólo tecnológicos

Ley de Brooks: agregar gente a un proyecto atrasado lo atrasa más

Cuidar la comunicación



# Desarrollo y Algoritmos III

Es una materia de Programación  
La última obligatoria de contenidos

Pero también se ve

Diseño

Pruebas

Buenas prácticas metodológicas

Buena calidad de código

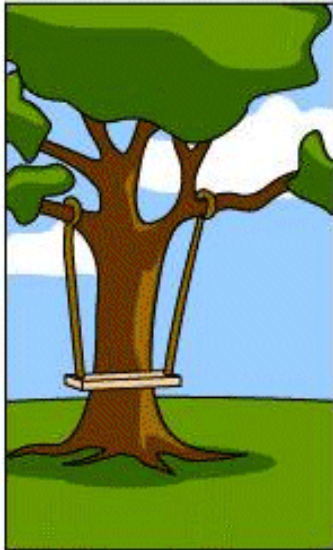
Usabilidad



# Metodología



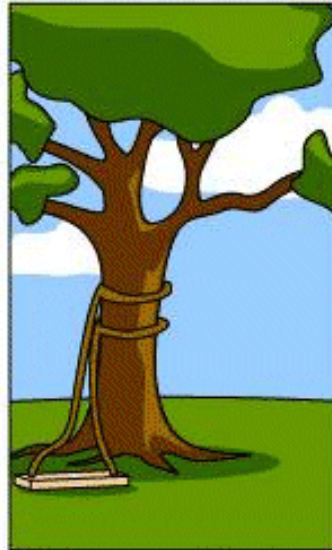
La solicitud del usuario



Lo que entendió el líder del proyecto



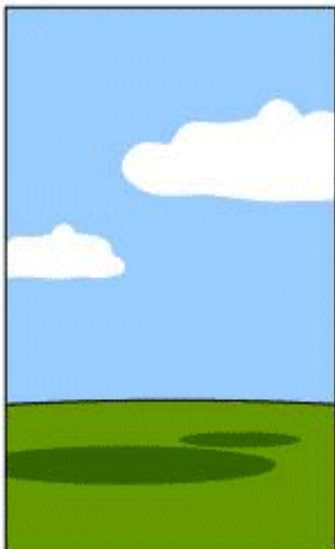
El diseño del analista de sistemas



El enfoque del programador



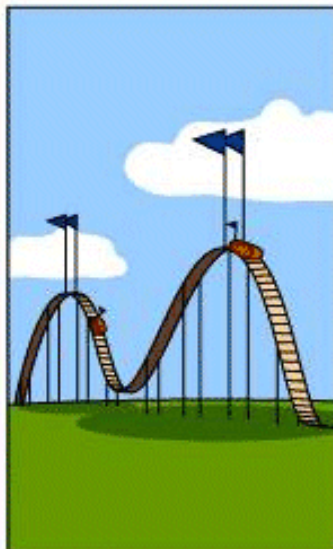
La recomendación del consultor externo



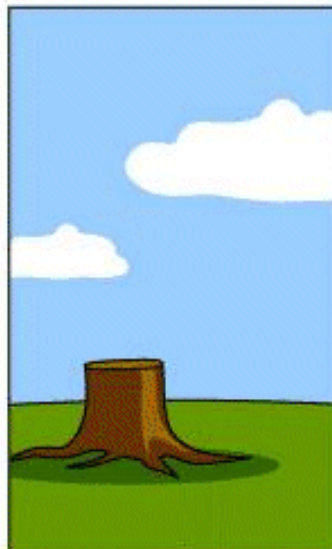
La documentación del proyecto



La implantación en producción



El presupuesto del proyecto



El soporte operativo



Lo que el usuario realmente necesitaba



# Inconvenientes de la cascada

Impide comenzar una etapa hasta que la anterior no está concluida => retraso

Cambios sobre una etapa ya terminada, a costa de burocracia y documentación

El usuario final recién ve el sistema una vez que está terminado:

- Poco consustanciado con el desarrollo

- No advierte errores de concepción a tiempo



# Métodos incrementales

Cascadas parciales (?)

Facilidad para atender cambios de requerimientos

Errores aparecen antes

Mayor continuidad entre actividades

Competencias más amplias:

Analista tiene algo de diseñador

Diseñador tiene algo de programador



# Categorías de métodos (1)

## Basados en etapas

Cascada => la distribución en el tiempo se basa en actividades

Análisis, diseño, programación, pruebas, ...

## Basados en funcionalidades

Incrementales => la distribución en el tiempo se basa en el desarrollo y entrega de grupos funcionales

Parte I, parte II, etc.



# Categorías de métodos (2)

## Procesos predictivos (¿basados en planes?)

Planificación más detallada y rígida

Se utilizan en grandes proyectos: suelen ser inaceptablemente pesadas para sistemas pequeños o medianos

Destacan el Proceso Unificado (UP), TSP, Cleanroom

## Métodos ágiles o adaptables

Más abiertos a los cambios

Permiten organizar desarrollos medianos sin caer en burocracias paralizantes

Alternativa a carecer de metodología

Destacan Extreme Programming (XP) y Scrum



# Ágiles vs. predictivos

Equipos pequeños y requisitos cambiantes => Ágiles

Equipos grandes y requisitos estables => Predictivos



# Métodos ágiles (I)

## Supuestos (o credo)

El proceso de desarrollo de software es inherentemente cambiante  
Abrazar el cambio, no gerenciarlo

## Objetivos

Bajar el riesgo  
Permitir cambios de especificaciones durante el desarrollo  
Favorecer la comunicación con el cliente  
Que la inversión crezca gradualmente

Hay un “Manifiesto ágil”

=>



# Manifiesto ágil

(<http://www.agilemanifesto.org/iso/es/>)

Estamos descubriendo formas mejores de desarrollar software tanto por nuestra propia experiencia como ayudando a terceros

A través de este trabajo hemos aprendido a valorar:

Individuos e interacciones sobre procesos y herramientas

Software funcionando sobre documentación extensiva

Colaboración con el cliente sobre negociación contractual

Respuesta ante el cambio sobre seguir un plan

Esto es, aunque valoramos los elementos de la derecha, valoramos más los de la izquierda.



# Métodos ágiles (II)

Ojo con:

Tiempos fijos.

Alcances fijos.

Presupuestos fijos.

Variables manejables

Calidad

Costo

Tiempo de desarrollo

Alcance

=> ajustar cualquier variable (puede ser el alcance) menos la calidad



# Los métodos ágiles

Extreme programming (XP), de Kent Beck y la comunidad  
Smalltalk

Lleva al extremo las buenas prácticas => es un conjunto de buenas prácticas

Lo analizamos acá, en un curso de Programación

Scrum, de Ken Schwaber y Mike Beedle

Provee roles y artefactos centrados en seguimiento y control del proyecto

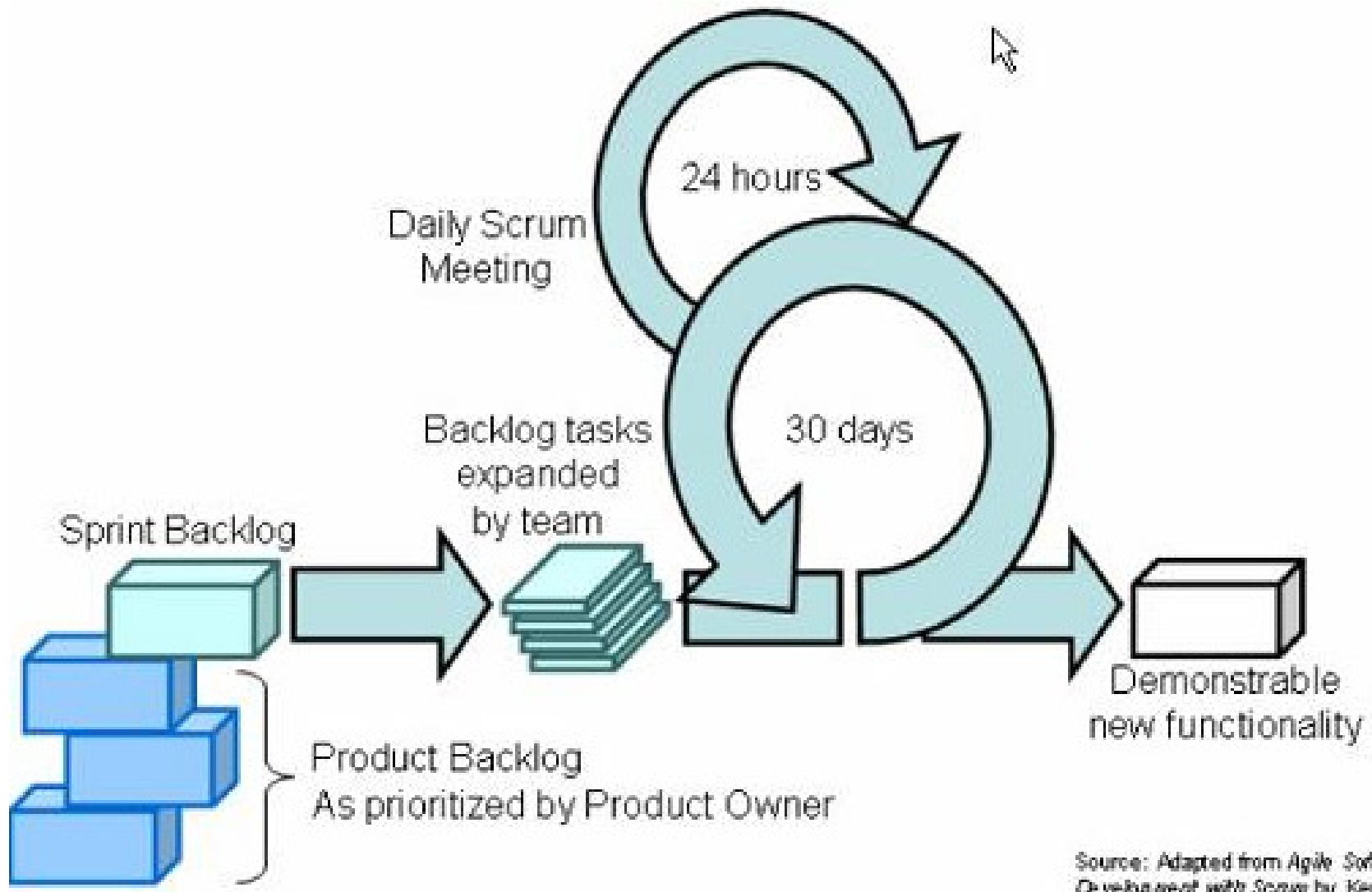
Lo van a analizar en materias de Administración de Proyectos

Pero un poco vamos a detenernos

Otros: ASD, Cristal, FDD, DSDM, MSF for Agile



# Scrum: ciclo



Source: Adapted from Agile Software Development with Scrum by Ken Schwaber and Mike Beedle.



# Scrum: roles

## Centrales

Product Owner

Scrum Master

Team Members

## Interesados o stakeholders

Clientes

Gerentes

Inversores



By Clark & Vizdos

© 2006 implementingscrum.com





# Claves

Desarrollo >> Programación

Sistemas >> Programas

Problemas desarrollo >> Problemas tecnológicos



# Lecturas opcionales (1)

Artículo de Wayt Gibbs en Scientific American,  
“Software’s Chronic Crisis” en:  
<http://www.cis.gsu.edu/~mmoore/CIS3300/handouts/SciAmSept1994.html>

Paper de Fred Brooks, “The Mythical Man-Month”:  
buscar en la Web

Artículos de Carlos Fontela en blog:  
<http://cysingsoft.wordpress.com/>



# Lecturas opcionales (2)

Básicos sobre métodos ágiles:

<http://agilemanifesto.org/iso/es/>

<http://agilemanifesto.org/iso/es/principles.html>

<http://www.mountangoatsoftware.com/topics/scrum>

<http://xprogramming.com/xpmag/whatisxp>

Más avanzado:

<http://www.proyectalis.com/wp-content/uploads/2008/02/scrum-y-xp-desde-las-trincheras.pdf>



# Qué sigue

Temas de diseño

RTTI, reflexión

Lenguajes y POO

