# La última lección

- Resumen del curso
- Buenas prácticas
- Malas prácticas
- Conclusión

# Objetivos

- Mostrar las técnicas básicas para planificar, gestionar y desarrollar productos de software complejos (Proyectos Informáticos, Sistemas de Información) de gran tamaño.

- Dominar el proceso y las herramientas de análisis, diseño, implementación y pruebas de software orientado a objetos (PUD, UML).

- Aplicación práctica a un problema real.

- Un mensaje básico: en cada ámbito, una buena organización es necesaria si queremos producir software de calidad (eficaz, eficiente, robusto, etc.) rápidamente.

# Temario

- Planificación y Gestión de proyectos
- Análisis y diseño de Sistemas de Información
- Implementación y pruebas

# El Proyecto

- El proyecto lo realiza un equipo de 6 personas
- Tiene un peso del 50% de la nota final.

- El proyecto consiste en planificar, analizar, diseñar, construir, probar y entregar un producto software.
- EL objetivo del proyecto es enseñar qué puede ir mal en el desarrollo de un proyecto informático (de la manera más real).

# La motivación

- El desarrollo de software frecuentemente va mal. Mal significa:
  - TARDE (nunca se cumplen los plazos)
  - CARO (por encima del presupuesto)
  - INEFICACES (no consiguen lo que se pretendía)
  - causan STRESS (al informático ¡y al cliente!)
- Se ha estimado que el 15% de los proyectos informáticos se anulan antes de terminarse. Este número crece al 25% de los proyectos que requieren más de 25 años/persona.

# La motivación

- ¿Por qué desarrollar software es tan difícil?
- El software es:
  - siempre nuevo (si no, queda obsoleto)
  - cada vez más complejo (¿cómo se gestiona la complejidad?)
  - difícil de controlar y verificar (poco robusto)
  - desarrollado básicamente de forma artesanal (la mayor parte del coste del desarrollo de un SI es en personal)
  - desarrollado básicamente a medida (casuística)
  - Entre el 60 - 85% de los costes en software se invierten en *modificaciones*

## Las soluciones

- Lenguajes de alto nivel
- Programación estructurada
- Diseño modular
- Métodos formales
- Tipos Abstractos de Datos
- Programación Orientada a objetos
- Programación lógica, funcional, etc.
- Lenguajes de 4 generación
- Entornos visuales de desarrollo, ...

# Buenas prácticas

- No hay varita mágica (una solución única)
- Nuestro mejor aliado será analizar y comprender la diferencia entre buenos y malos proyectos, descubrir cuáles son las buenas prácticas y adoptarlas.
- En otras palabras, debemos aplicar el "sentido común":
  - Estar atentos al proceso de desarrollo
  - Adoptar estándares para la documentación, codificación, etc.
  - Aprender de los errores (¡preferiblemente de otra gente!)

# Malas prácticas

- Aprendemos más equivocándonos
- Es mucho mejor aprender de las "malas prácticas"
- Las malas prácticas son aquellas que nos llevan al desastre…
- Preguntaos si alguna de estas cosas pasan en vuestro grupo...

# Malas prácticas

- Errores sobre la gente
- Errores sobre el proceso
- Errores sobre el producto
- Errores sobre la tecnología

# Errores sobre la gente

- Personal poco preparado: si el personal no es bueno, el proyecto tampoco. Mejor esperar y buscar al personal adecuado. Autoformación y formación en la empresa.

- Falta de motivación: la falta de motivación destruye los proyectos. ¿Cómo aumentar la motivación del personal?

- Falta de compromiso: ¡todo el mundo debe conocer su función y sus responsabilidades!

# Errores sobre la gente

- Mala distribución de la carga entre el personal: el trabajo debe distribuirse racionalmente. Nadie debe ser imprescindible.
  - El empleado acaparador "ni come, ni deja comer"
  - Nadie es infalible
- Creer en las heroicidades: los esfuerzos sobrehumanos (normalmente de última hora) producen supercatástrofes.
- Añadir más gente a proyectos retrasados: un clásico. Mejor rehacer la planificación y posponer la entrega. Si no, los miembros del equipo además perderán su "valioso" tiempo para ayudar a los nuevos.

# Errores sobre la gente

- Falta de sintonía entre clientes y desarrolladores
  - El cliente piensa que el desarrollador no trabaja suficiente
  - Los desarrolladores piensan que el cliente no sabe lo que quiere, no se aclara o exige demasiado
  - El PUD exige una MUY BUENA comunicación entre usuarios y desarrolladores
- Expectativas poco realistas: la mejor manera de deprimir al equipo y entregar un producto de baja calidad. Valorar la capacidad del equipo es la tarea más difícil.
- Pensar que todo se solucionará por si sólo (otro del equipo ya se encargará de arreglarlo ...)

# Errores sobre el proceso

- Plan deficiente (o irreal): otro clásico. No tener un plan suficientemente detallado, que nos indique dónde estamos y cuantos recursos hemos consumido.

- Abandonar el plan: ¡Nunca! En todo caso, mejorarlo

- No coger al toro por los cuernos: al principio del proyecto nadie sabe qué debe hacerse. No podemos perder tiempo pensando en términos difusos. Manos a la obra cuanto antes.

- El código es lo único que importa (y que ejecute, claro): otro clásico. Creer que el tiempo de análisis de requisitos, diseño, control del desarrollo, entrevistas con los usuarios, etc. es tiempo perdido (o no tomárselo en serio)

# Errores sobre el producto

- Requisitos nuevos: podemos perder mucha energía, tiempo y dinero añadiendo nuevas funcionalidades no previstas inicialmente y que el usuario puede que nunca vaya a entender

- Pensar que es un producto acabado cuando lo terminamos (y que no tiene errores)

- Huida hacia delante: otro clásico. Si vamos mal de tiempo, por qué no añadir unas nuevas funcionalidades...

# Errores sobre la tecnología

- El síndrome de la "panacea". ¿Que la programación no va bien? Pues usemos un lenguaje de alto nivel. Ah, ya lo hacemos, pues uno OO. Con eso es imposible ir mal. Ah, que ya lo usamos... Pues probar con una herramienta CASE, eso solucionará todos nuestros problemas... ¡eso dice en la caja!

- Ninguna herramienta o método, por si sola, resuelve nada. Este error es aún peor a mitad de proyecto.

- Sobreestimar el ahorro que producirá una nueva tecnología.

- Falta de control automático sobre el código fuente.

# Conclusión

- El desarrollo de SI es un proceso "artesanal" sumamente complejo. Todos los errores descritos pueden ser transcendentes... y no existe la "varita mágica".

- Las "buenas prácticas" sólo reducen el riesgo de encontrarnos con problemas realmente serios

- ¿Cómo va el proyecto? Pues la vida real no tiene por que ser mejor.