La esencia de la Ingeniería de Software

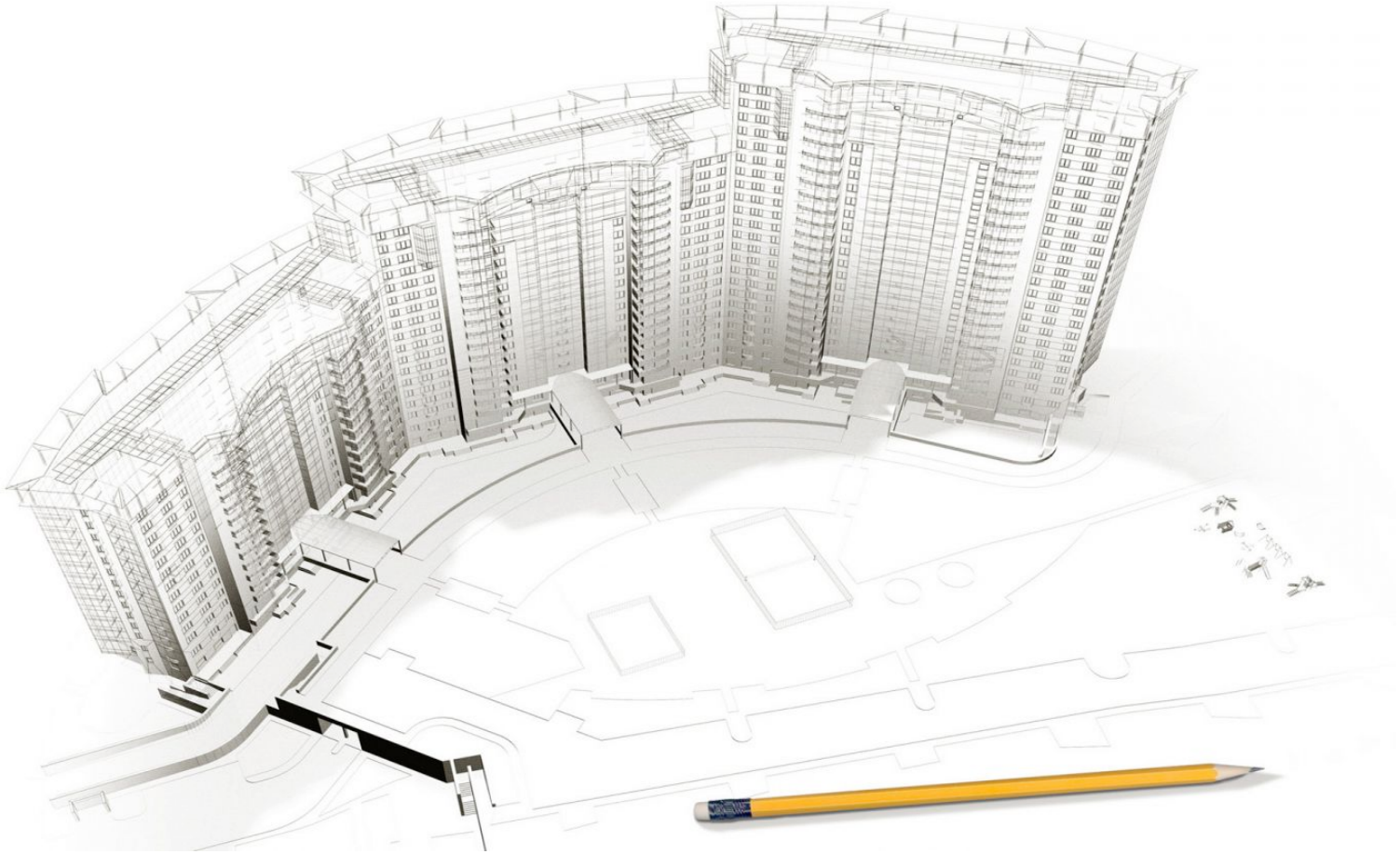
- Modelación: transformación del concepto en requerimientos, de requerimientos en diseño, del diseño en código fuente y de este al código ejecutable
- Diseño: encontrar la mejor solución que responda a requisitos sujeto a restricciones
- Optimización: encontrar las transformaciones de mejor calidad y más económicas

Escribir un Programa

Introducción a la Programación

Programación Avanzada

Desarrollar un Software Real



Diferencias

- participan muchas personas
- cuidadosa planeación de cada etapa
- cuidadoso diseño antes de edificar (planos, maquetas)
- cuidadosos cálculos para asegurar diseño
- revisiones y controles
- monitoreo y control del proyecto

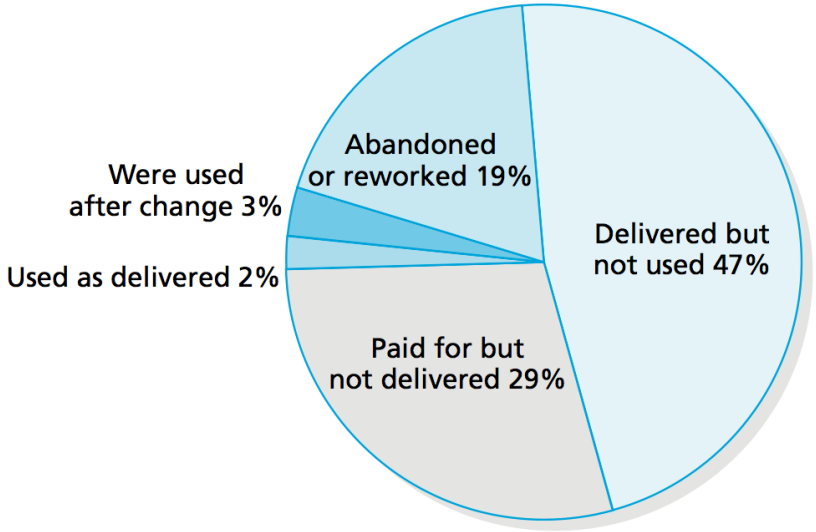
La Necesidad de Ingeniería

- Prácticamente todo (productos, servicios) incluye software
- Software puede ser extremadamente complejo
- Debe ser extendible, modificable, escalable
- Debe ser seguro
- Se requieren métodos, herramientas y técnicas porque enfoque ingenuo, intuitivo no funciona

Los Principales Desafíos

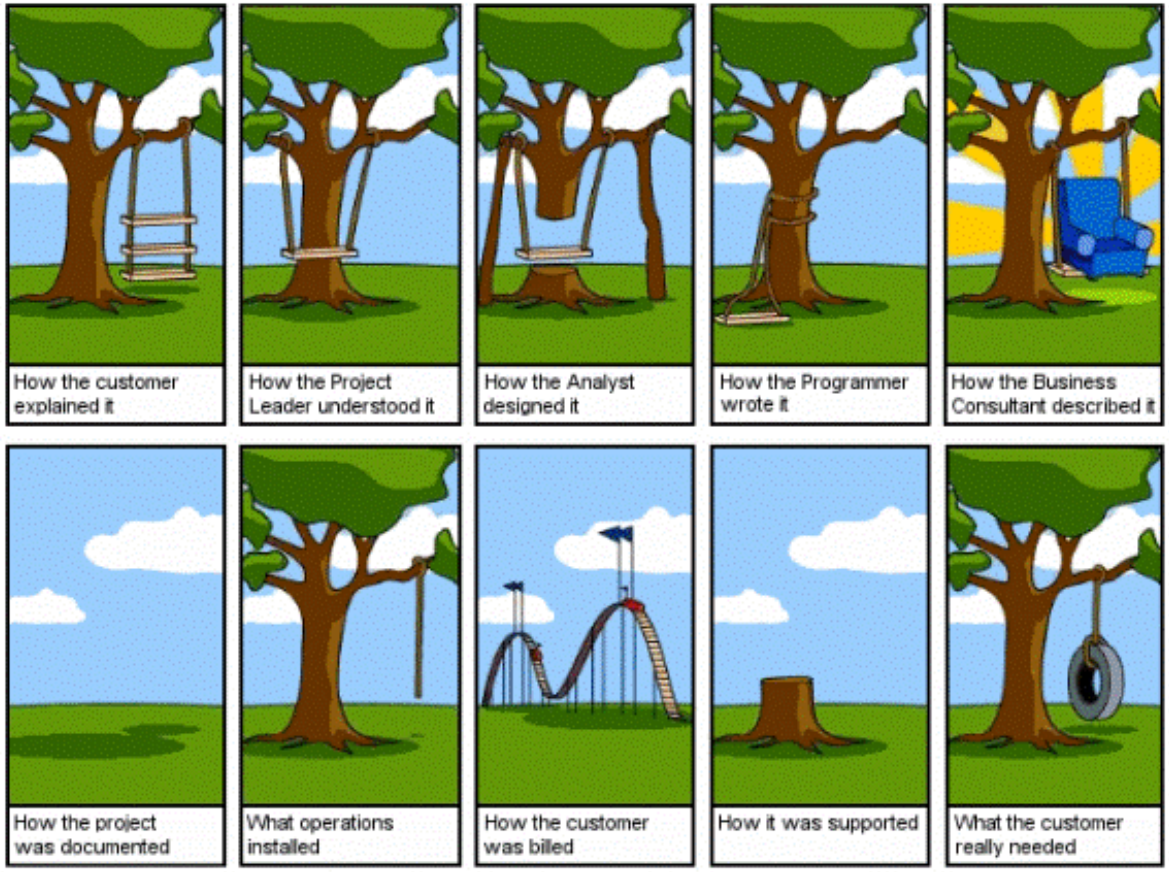
- responder a las necesidades de los usuarios
- costos reducidos
- exigencia de alto desempeño
- portabilidad
- bajo costo de mantención
- confiabilidad
- entrega a tiempo

No es fácil responder a necesidades de los usuarios



- Análisis de requisitos o Ingeniería de requisitos
- Se considera uno de los factores críticos en el éxito de un proyecto
- Evidencia histórica muestra algunos problemas

El problema de comunicación



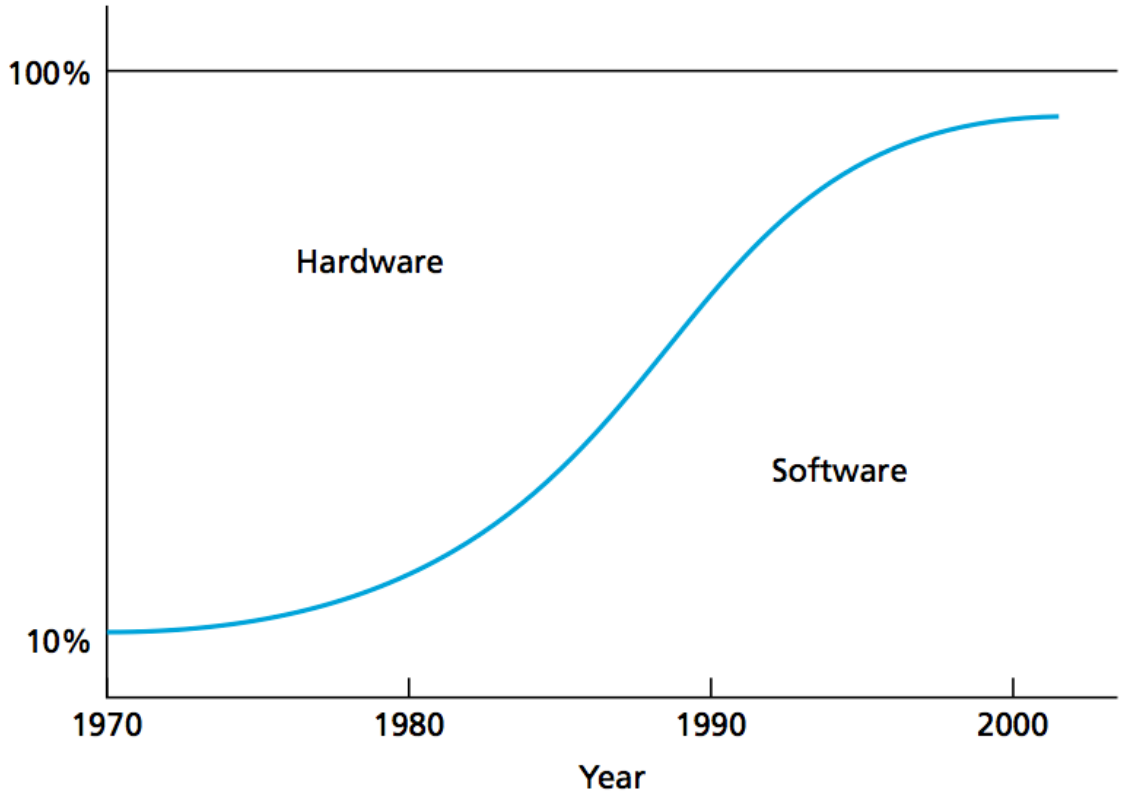
El problema de costos

- El software puede representar mucho dinero en un proyecto
- Se gastan más de US$1000 billones al año en el mundo
- Casi la totalidad se debe a salarios de los desarrolladores
- Productividad de programador es crucial (10 a 20 líneas de código al día)

Poniendo números …

- Un producto como Word tiene cerca de1 millón de líneas de código
- 1.000.000 / 20 = 50.000 días de programador
- 300 días/año = 167 años
- 80 desarrolladores durante 2 años

Los costos de Software dominan los proyectos



Confiabilidad

- testing permite detectar errores
- testing no permite asegurar que ya no quedan errores
- nos conformamos con “good enough”
- “zero defect” software (safety crítical)

Bugs pueden ser muy costosos

- Sonda enviada al planeta Venus incluía el siguiente bug en código Fortran
  - DO 3 I = 1.3 en lugar de DO 3 I = 1,3
  - Se interpreta como asignación DO3I = 1.3
  - Sonda fue perdida

Cumpliendo con los plazos de entrega

- Un dolor de cabeza permanente
- Dificultad de hacer estimaciones de esfuerzo
- Dificultad para corregir atrasos una vez que se producen
- Desarrolladores tienden a subestimar el trabajo
- Area comercial presiona por plazos mas cortos

Performance

- Ha disminuido el foco debido a disponibilidad de hardware rápido y barato
- Hay proyectos en que sigue siendo crucial
  - software interactivo
  - juegos
  - señales de control
  - restricciones de ventana de tiempo (banco)

Portabilidad

- Ya que se invierte tanto en el Software tiene sentido poder llevarlo a otro hardware
- Lenguajes standard de alto nivel permiten avanzar
- Casi nunca es tan sencillo

Otros bugs célebres

- Cohete Europeo Arianne 5 se estrelló en 1996 30 segundos después de lanzamiento (US$500 millones a la basura)

- En 1999 eBay se cayó por 22 horas - US$6 billones de pérdidas en la acción

Interacción Humano Computador

- Puede ser crítico en aceptación de un producto o servicio
- Algunos ejemplos
  - programación de un VCR
  - servicios de transferencia de llamadas
  - pago automatizado de estacionamiento
  - pagos por internet

Houston, we have a problem

- el software a menudo no hace lo que usuarios quieren
- el software es muy caro
- el software no es suficientemente rápido
- el software no puede ser portado
- el software es caro de mantener
- el software es poco confiable
- el software siempre se atrasa
- el software es difícil de usar

Ingeniería de Software al Rescate

- mayor énfasis en llevar a cabo el desarrollo en forma sistemática
- herramientas de apoyo
- mayor énfasis en averiguar lo que el usuario exactamente requiere
- demostrar en forma temprana el sistema a los clientes
- mayor énfasis en asegurar que no hay errores
- desarrollo incremental

¿ Que aprenderemos ?

1. Motivación
   - la necesidad de ingeniería de software
   - software como servicio
   - desafíos y oportunidades
2. Proceso
   - la necesidad de proceso
   - modelo de cascada
   - procesos iterativos: prototipos y RUP
   - procesos incrementales
   - métodos ágiles
   - Scrum y Kanban
3. Requisitos
   - funcionales y no funcionales
   - relatos de usuario
   - casos de uso
4. Diseño
   - modelo de dominio
   - atributos de un buen diseño
   - acoplamiento y cohesión
   - diagramas UML de clases, secuencia y estados
   - patrones de diseño
5. Arquitectura
   - conceptos fundamentales
   - atributos que impactan la arquitectura
   - patrones arquitectónicos
   - arquitecturas cliente servidor y multicapas
   - arquitectura orientada a servicios
   - microservicios
6. Gestión del Proyecto
   - actividades de gestión
   - estimaciones
   - planeación de producto, release y sprint
   - gestión de personas
7. Aseguramiento de Calidad (QA)
   - definiciones de calidad
   - prevención de defectos
   - detección y eliminación de defectos
   - testing

Bonus Track

- Arquitectura de una Aplicación Web (SaaS)
- Estándares de la Web (Http, Html, CSS, URI)
- Fundamentos del Lenguaje Ruby
- Utilización de un framework de desarrollo (Rails)
- Desarrollo de una aplicación en plataforma Ruby/Rails
- Pruebas unitarias en la plataforma Ruby/Rails