Introducción a la Ingeniería de Software

Matemática, lógica y diseño para el desarrollo de software

Maximiliano Cristiá

CIFASIS – UNR

cristia@cifasis-conicet.gov.ar

CONICET

CIFASIS

Equipo docente

Maximiliano Cristiá – cristia@cifasis-conicet.gov.ar

Andrés Krapf – akrapf@fceia.unr.edu.ar

Sebastián Scandolo – sebas@fceia.unr.edu.ar

CONICET CIFASIS

Recursos de la materia

Páginas web

Ingeniería de Software I: `www.fceia.unr.edu.ar/is1`

Ingeniería de Software II: `www.fceia.unr.edu.ar/is2`

Apuntes, prácticas, bibliografía

Lista de correo: `ingsoft@fceia.unr.edu.ar`

- Suscribirse con nombre y apellido
- La suscripción se hace en:

  `http://listas.fceia.unr.edu.ar/cgi-bin/mailman/listinfo/ingsoft`

Fotocopias en Alfa (?)

CET CIFASIS

Canal youtube

youtube.com/c/maximilianocristiais

Buscar en youtube:

maximiliano cristiá ingeniería software

NO son filmaciones de las clases

CONICET CIFASIS

Condiciones para cursar y rendir

- Tener todas las materias correlativas aprobadas al momento de rendir
- Estar inscriptos en la mesa de examen
- Si no cumplen estas condiciones no pueden rendir

Temario de la materia

1. Introducción a la Ingeniería de Software
2. El lenguaje de especificación formal Z
3. Introducción al modelo WRSPM para requerimientos y especificaciones
4. Statecharts
5. Communicating Sequential Processes (CSP)
6. Temporal Logic of Actions (TLA)
7. El asistente de pruebas Z/EVES

CONICET CIFASIS

Evaluaciones

- Dos exámenes parciales y un recuperatorio
  - 1er parcial evalúa Z (aprox. 29/4)
  - 2do parcial evalúa Statecharts y CSP (aprox. 27/5)
  - Recuperatorio recupera un parcial (última semana clase)
- Notas: 1-5 insuficiente, 6-7 aprobado, 8-10 promovido
- Promoción dura hasta que la materia se vuelve a dar
- Examen final
  - Promovidos en los dos parciales rinden TLA
  - Promovidos en un parcial rinden TLA más lo no promovido
  - Regulares (aprobados en los dos parciales) rinden práctica de toda la materia
  - Libres rinden práctica y teoría de toda la materia

CONICET CIFASIS

Evaluaciones

Z/EVES se evalúa en un trabajo práctico que se entrega a lo sumo el día que se presentan a rendir el final de **IS 2**

Una parte del TP consiste en escribir una especificación Z

Otra parte en usar $\{log\}$ que **NO** se enseña en clase

Por lo que pueden empezar a hacer el TP hacia fines de abril

NO tienen por qué esperar hasta terminar IS 2 para empezar

`www.fceia.unr.edu.ar/is2/tp.html`

CONICET CIFASIS

Aspectos epistemológicos de la Ingeniería de Software

CONICET
CIFASIS

¿Qué es la Ingeniería de Software?

NATO – 1968

Enfoque sistemático, disciplinado y cuantificable del desarrollo, operación y mantenimiento de software.

Parnas – 1978

La construcción de múltiples versiones de un software llevada a cabo por múltiples personas.

Ghezzi – 1991

Construcción de software de una envergadura o complejidad tales que debe ser construido por equipos de ingenieros.

CONICET CIFASIS

¿Qué es la Ingeniería de Software? (cont.)

Jackson – 1998

La ingeniería tradicional es altamente especializada y se basa en colecciones de diseños estándar o normalizados. ¿Hay especialidades en la Informática o cualquiera hace cualquier cosa? ¿Se basa la producción de software en diseños estándar? ¿Puede?

Parnas – 1997

Reemplazar las renuncias de responsabilidad por garantías

En resumen...

La Ingeniería de Software es o debería ser:

- Desarrollo de software de dimensión industrial
- Desarrollo sistemático, disciplinado y cuantificable
- Desarrollo de productos que tienen una vida muy larga
- Desarrollo en equipo
- Especialización
- Diseños estándar
- Producir software garantizado

CONICET
CIFASIS

¿Qué hace el ingeniero de software?

Máquinas de software

No construye el hardware, sino el comportamiento y las propiedades que lo harán útil para algo específico.

Escribe descripciones

- La actividad central del desarrollo de software es la descripción.
- Cualquier desarrollo de software requiere muchas descripciones.

Verifica las descripciones

CONICET CIFASIS

¿Qué hace el ingeniero de software?

Máquinas de software

No construye el hardware, sino el comportamiento y las propiedades que lo harán útil para algo específico.

Escribe descripciones

- La actividad central del desarrollo de software es la descripción.
- Cualquier desarrollo de software requiere muchas descripciones.

Verifica las descripciones

¿Qué hace el ingeniero de software?

Máquinas de software

No construye el hardware, sino el comportamiento y las propiedades que lo harán útil para algo específico.

Escribe descripciones

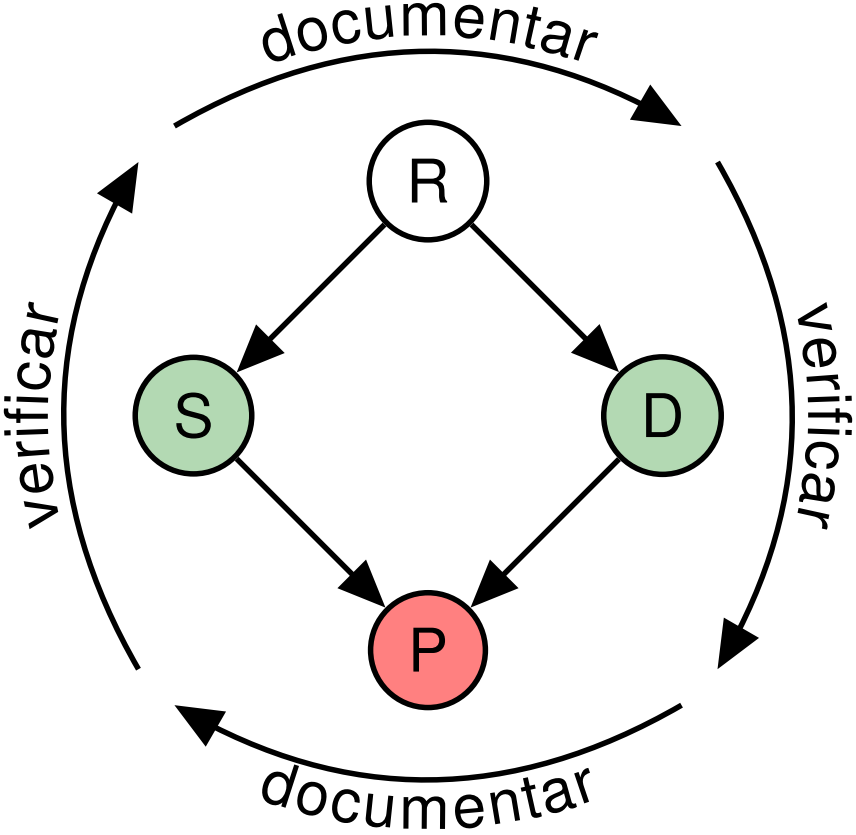
- La actividad central del desarrollo de software es la descripción.
- Cualquier desarrollo de software requiere muchas descripciones.

Verifica las descripciones

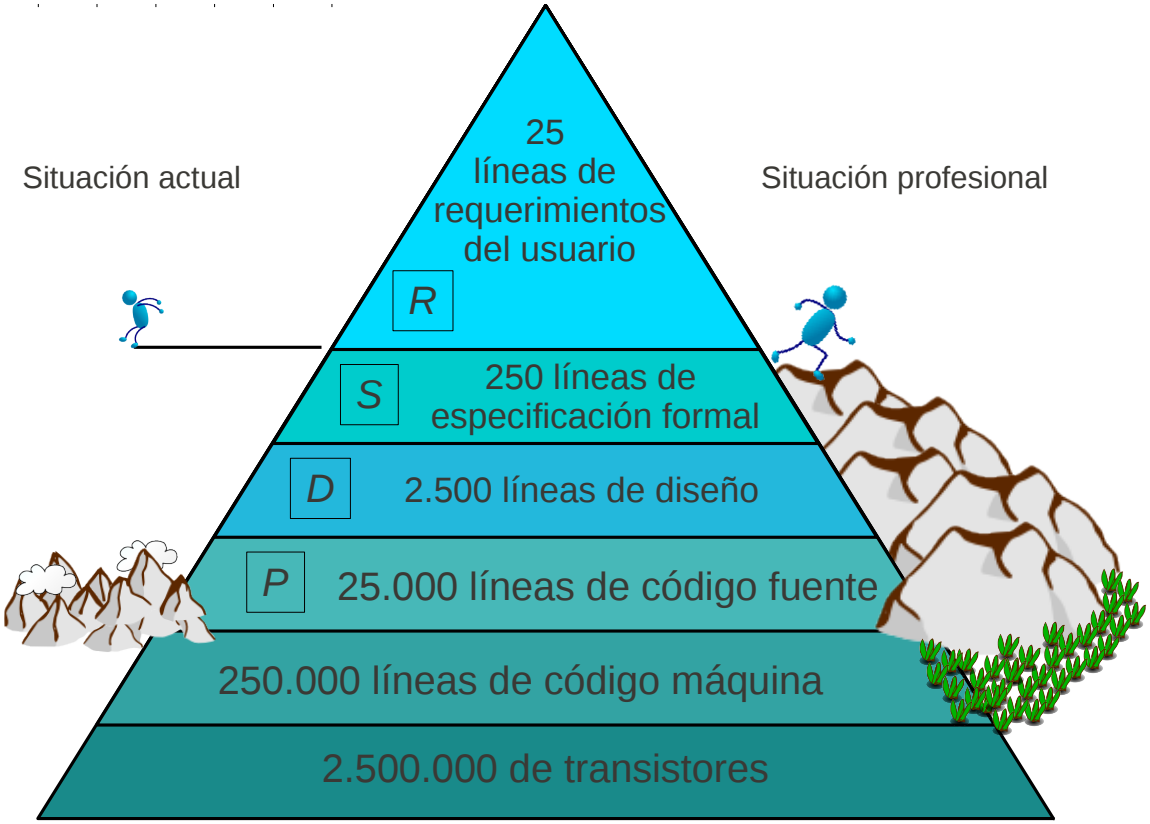
CONICET
CIFASIS

Las cuatro descripciones fundamentales

- Requerimientos del usuario (*R*)
  - Única descripción informal
- Diseño de la estructura del programa (*D*)
- Especificación funcional del programa (*S*)
- Programa (*P*)



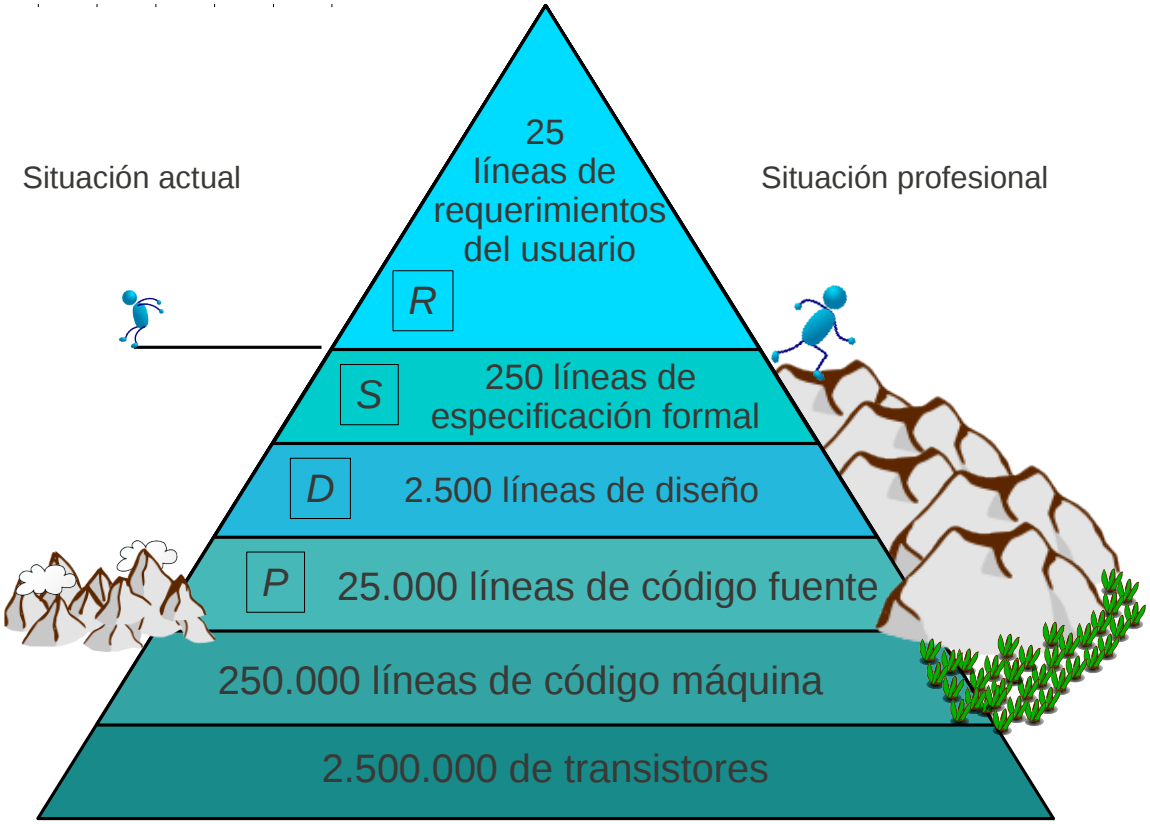
¿Por qué **cuatro** descripciones fundamentales?



Porque con menos la tarea del desarrollador es muy riesgosa

Porque con menos aumentan los costos de producción

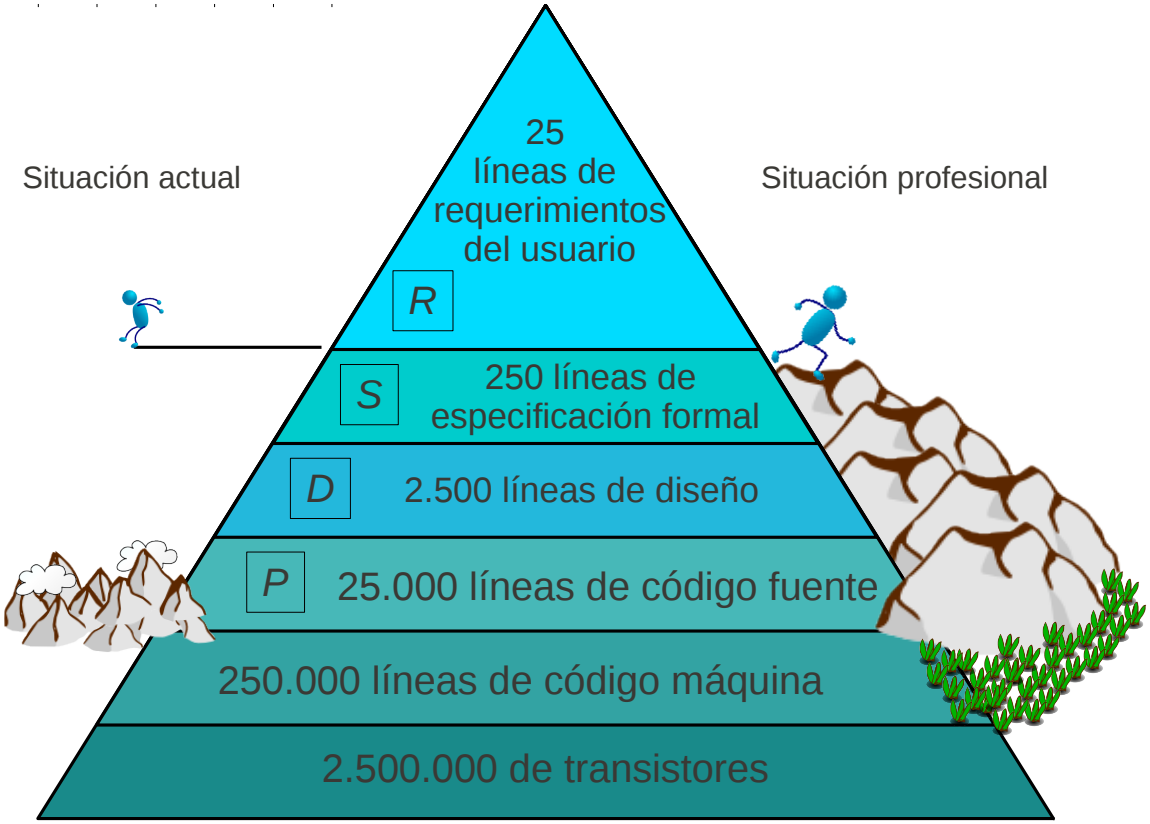
¿Por qué **cuatro** descripciones fundamentales?



Porque con menos la tarea del desarrollador es muy riesgosa

Porque con menos aumentan los costos de producción

¿Por qué **cuatro** descripciones fundamentales?



Porque con menos la tarea del desarrollador es muy riesgosa

Porque con menos aumentan los costos de producción

¿Qué debe saber un ingeniero de software?

- Dominar a fondo las técnicas de descripción
  - Esencialmente debe dominar los lenguajes formales
- Etender qué hace que una descripción particular sirva o no para un propósito determinado
- Moverse en distintos niveles de abstracción
- Describir modelos mediante lenguajes formales
- Verificar propiedades de los modelos

Escribir modelos **formales** y **abstractos** del programa

CONICET
CIFASIS

¿Qué debe saber un ingeniero de software?

- Dominar a fondo las técnicas de descripción
  - Esencialmente debe dominar los lenguajes formales
- Etender qué hace que una descripción particular sirva o no para un propósito determinado
- Moverse en distintos niveles de abstracción
- Describir modelos mediante lenguajes formales
- Verificar propiedades de los modelos

Escribir modelos **formales** y **abstractos** del programa

¿Qué debe saber un ingeniero de software? (cont.)

Documentar y validar los requerimientos del usuario

Escribir un modelo abstracto semiformal del diseño

Escribir una especificación funcional abstracta y formal

Verificar que el programa satisface el diseño y
la especificación funcional

¿Qué debe saber un ingeniero de software? (cont.)

Documentar y validar los requerimientos del usuario

Escribir un modelo abstracto semiformal del diseño

Escribir una especificación funcional abstracta y formal

Verificar que el programa satisface el diseño y la especificación funcional

¿Qué debe saber un ingeniero de software? (cont.)

Documentar y validar los requerimientos del usuario

Escribir un modelo abstracto semiformal del diseño

Escribir una especificación funcional abstracta y formal

Verificar que el programa satisface el diseño y la especificación funcional

¿Qué debe saber un ingeniero de software? (cont.)

Documentar y validar los requerimientos del usuario

Escribir un modelo abstracto semiformal del diseño

Escribir una especificación funcional abstracta y formal

Verificar que el programa satisface el diseño y
la especificación funcional

La Ingeniería de Software, ¿es ingeniería?

Sin garantía

Casi ningún programa se entrega con garantía. ¿Podemos decir que el software es el resultado de una ingeniería cuando todos los productos de otras ingenierías tienen garantía?

Sin diseños estándar

Excepto en pocas excepciones, los desarrolladores de software tienden a inventar todo en cada proyecto. ¿Podemos decir que el software es el resultado de una ingeniería cuando casi todo se hace de cero casi siempre?

La Ingeniería de Software, ¿es ingeniería?

Sin garantía

Casi ningún programa se entrega con garantía. ¿Podemos decir que el software es el resultado de una ingeniería cuando todos los productos de otras ingenierías tienen garantía?

Sin diseños estándar

Excepto en pocas excepciones, los desarrolladores de software tienden a inventar todo en cada proyecto. ¿Podemos decir que el software es el resultado de una ingeniería cuando casi todo se hace de cero casi siempre?

¿Es ingeniería? (I)

- Aeropuerto de Denver en 1993 (EE.UU.)
- 10 veces el tamaño de Heathrow (aeropuerto de Londres)
- Sistema subterráneo de traslado de equipaje: 4.000 telecarros independientes
- Software para el control del sistema de carros (21 meses)
- La inauguración se debió postergar 3 veces
- El presupuesto era de USD 193M
- BAE Automated Systems reconoció que no podía predecir el momento en que lograría estabilizarlo
- En 2005 United decidió abandonar el sistema y ahorrarse USD 1M por mes en mantenimiento

¿Es ingeniería? (II)

- Satélite Clementine: proyecto conjunto entre Departamento de Defensa (EE.UU.) y NASA
- Parte del programa de defensa denominado Guerra de las Galaxias
- Satélite para selección de blancos
- Por un error en el software de control, en lugar de situar a la Luna en la mira, el sistema encendió los motores durante 11 minutos y agotó el combustible.

¿Es ingeniería? (III)

Encuesta de IBM sobre sistemas distribuidos (aprox. 1994)

- 55 % costó más de lo calculado
- 68 % no cumplió los plazos
- 88 % tuvo que re-diseñarse casi desde cero

¿Es ingeniería? (IV)

Ariane-5

Hito en el proyecto espacial europeo; terminó en desastre debido a una falla en el software que controlaba el movimiento vertical.

Therac-25 (radioterapia)

Dispositivo médico de radioterapia para el tratamiento del cáncer. Se reemplazaron ciertos controles de hardware por software. El software falló y causó la muerte de varias personas por sobredosis.

FBI

Gastó 170 millones de dólares en el Virtual Case File para luego abandonarlo completamente.

¿Es ingeniería? (IV)

Ariane-5

Hito en el proyecto espacial europeo; terminó en desastre debido a una falla en el software que controlaba el movimiento vertical.

Therac-25 (radioterapia)

Dispositivo médico de radioterapia para el tratamiento del cáncer. Se reemplazaron ciertos controles de hardware por software. El software falló y causó la muerte de varias personas por sobredosis.

FBI

Gastó 170 millones de dólares en el Virtual Case File para luego abandonarlo completamente.

¿Es ingeniería? (IV)

Ariane-5

Hito en el proyecto espacial europeo; terminó en desastre debido a una falla en el software que controlaba el movimiento vertical.

Therac-25 (radioterapia)

Dispositivo médico de radioterapia para el tratamiento del cáncer. Se reemplazaron ciertos controles de hardware por software. El software falló y causó la muerte de varias personas por sobredosis.

FBI

Gastó 170 millones de dólares en el Virtual Case File para luego abandonarlo completamente.

¿Es ingeniería? (V)

Administración Federal de Aviación (EE.UU.)

Canceló un proyecto para actualizar los sistemas de control aéreo cuando ya se habían gastado 2.600 millones de dólares.

FoxMayer Drug Co. (EE.UU.)

Se fundió luego de que su ERP que costó USD 40M mostrara innumerables fallas.

Ministerio de Defensa del Reino Unido (2008)

Helicópteros Chinook. Problemas con el software de cabina dejaron 8 herlicópteros sin uso operacional. Luego de 7 años de trabajo, nunca fueron usados. El costo: USD 1.000M.

¿Es ingeniería? (V)

Administración Federal de Aviación (EE.UU.)

Canceló un proyecto para actualizar los sistemas de control aéreo cuando ya se habían gastado 2.600 millones de dólares.

FoxMayer Drug Co. (EE.UU.)

Se fundió luego de que su ERP que costó USD 40M mostrara innumerables fallas.

Ministerio de Defensa del Reino Unido (2008)

Helicópteros Chinook. Problemas con el software de cabina dejaron 8 herlicópteros sin uso operacional. Luego de 7 años de trabajo, nunca fueron usados. El costo: USD 1.000M.

¿Es ingeniería? (V)

Administración Federal de Aviación (EE.UU.)

Canceló un proyecto para actualizar los sistemas de control aéreo cuando ya se habían gastado 2.600 millones de dólares.

FoxMayer Drug Co. (EE.UU.)

Se fundió luego de que su ERP que costó USD 40M mostrara innumerables fallas.

Ministerio de Defensa del Reino Unido (2008)

Helicópteros Chinook. Problemas con el software de cabina dejaron 8 herlicópteros sin uso operacional. Luego de 7 años de trabajo, nunca fueron usados. El costo: USD 1.000M.

¿Es ingeniería? (VI)

BBC (2008-2013)

Herramienta de producción DMI (Iniciativa de Medios Digitales). El proyecto debía terminarse en 18 meses con un presupuesto de 82M libras. Luego de 5 años el proyecto fue abandonado con un costo de 100M de libras.

Novopay (2012-2013)

Sistema de pago a maestros y personal del sistema educativo de Nueva Zelanda. Se detectaron más de 18.000 liquidaciones erradas. Se detectaron más de 500 errores en el sistema. Costo: NZD 30M (aprox. USD 24M)

¿Es ingeniería? (VI)

BBC (2008-2013)

Herramienta de producción DMI (Iniciativa de Medios Digitales). El proyecto debía terminarse en 18 meses con un presupuesto de 82M libras. Luego de 5 años el proyecto fue abandonado con un costo de 100M de libras.

Novopay (2012-2013)

Sistema de pago a maestros y personal del sistema educativo de Nueva Zelanda. Se detectaron más de 18.000 liquidaciones erradas. Se detectaron más de 500 errores en el sistema. Costo: NZD 30M (aprox. USD 24M)

¿Es ingeniería? (VII)

Bank of America

En la década del 80 perdió USD 80M y todo su negocio de *trusts* a raíz de la imposibilidad de terminar el sistema para administrarlos, conocido como MasterNet. Hasta ese momento era considerado una de las empresas líderes en la adopción de nuevas tecnologías.

Fracasos en proyectos grandes

Se calcula que entre el 15 % y el 20 % de los proyectos de más de USD 10M se cancelan antes de ser si quiera terminados.

¿Es ingeniería? (VII)

Bank of America

En la década del 80 perdió USD 80M y todo su negocio de *trusts* a raíz de la imposibilidad de terminar el sistema para administrarlos, conocido como MasterNet. Hasta ese momento era considerado una de las empresas líderes en la adopción de nuevas tecnologías.

Fracasos en proyectos grandes

Se calcula que entre el 15 % y el 20 % de los proyectos de más de USD 10M se cancelan antes de ser si quiera terminados.

¿Es ingeniería? (VIII)

Software error doomed Japanese Hitomi spacecraft
Japan's flagship astronomical satellite Hitomi, which launched successfully on 17 February (2016) but tumbled out of control five weeks later, may have been doomed by a basic engineering error. Confused about how it was oriented in space and trying to stop itself from spinning, Hitomi's *control system* apparently commanded a thruster jet to fire in the wrong direction—accelerating, rather than slowing, the craft's rotation.

`http://www.nature.com/news/software-error-doomed-japanese-hitomi-spacecraft-1.19835`

CONICET CIFASIS

¿Es artesanía? (I)

CDIS - AdaCore (1992)

- sistema de información de tráfico aéreo de Gran Bretaña
- métodos formales en S, D y verificación
  - lenguajes de especificación: VDM, CSP

Resultados

- 197.000 líneas de código (LOC)
- esfuerzo total: 15.536 días-hombre
- defectos encontrados: 0.75 defectos por 1000 LOC

el esfuerzo total fue comparable o mejor respecto al de otros proyectos de la misma clase y el mismo tamaño

¿Es artesanía? (I)

CDIS - AdaCore (1992)

- sistema de información de tráfico aéreo de Gran Bretaña
- métodos formales en S, D y verificación
  - lenguajes de especificación: VDM, CSP

Resultados

- 197.000 líneas de código (LOC)
- esfuerzo total: 15.536 días-hombre
- defectos encontrados: 0.75 defectos por 1000 LOC

el uso de métodos formales no costó esfuerzo extra

CONICET CIFASIS

¿Es artesanía? (I)

CDIS - AdaCore (1992)

- sistema de información de tráfico aéreo de Gran Bretaña
- métodos formales en S, D y verificación
  - lenguajes de especificación: VDM, CSP

Resultados

- 197.000 líneas de código (LOC)
- esfuerzo total: 15.536 días-hombre
- defectos encontrados: 0.75 defectos por 1000 LOC

la cantidad de defectos encontrados fue menor en comparación con otros proyectos similares, y en general, no fueron fallas en los requerimientos ni en la especificación, lo cual hubiera sido muy costoso

¿Es artesanía? (II)

Tokeneer ID Station - AdaCore (2003)

- sistema biométrico aplicado a control de acceso
- proyecto conjunto NSA–AdaCore
- CC: difícil, demasiado costoso y económicamente inviable

Características

- Líneas de SPARK: 9.939 — Líneas de Z: 2.057
- Esfuerzo total: 260 días
- Productividad (LOC por día): 38
- Productividad (LOC por día, implementación): 203
- Defectos encontrados desde la entrega: 2
- CC Evaluation Assurance Level 5 (EAL5)

¿Es artesanía? (II)

Tokeneer ID Station - AdaCore (2003)

- sistema biométrico aplicado a control de acceso
- proyecto conjunto NSA–AdaCore
- CC: difícil, demasiado costoso y económicamente inviable

Características

- Líneas de SPARK: 9.939 — Líneas de Z: 2.057
- Esfuerzo total: 260 días
- Productividad (LOC por día): 38
- Productividad (LOC por día, implementación): 203
- Defectos encontrados desde la entrega: 2
- CC Evaluation Assurance Level 5 (EAL5)

¿Es artesanía? (III)

Operaciones matemáticas del AMD-K7

Demostración formal de la corrección funcional de las operaciones de multiplicación, división y raíz de punto flotante del micro AMD-K7. Especificación formal en ACL2 y prueba formal mediante el demostrador de ACL2.

MIL-STD 188-220

Especificación formal en Estelle del protocolo de comunicación 188-220 de las fuerzas armadas de EE.UU. Se generaron casos de prueba automáticamente a partir de la especificación que incrementaron en un 200 % la cobertura.

¿Es artesanía? (III)

Operaciones matemáticas del AMD-K7

Demostración formal de la corrección funcional de las operaciones de multiplicación, división y raíz de punto flotante del micro AMD-K7. Especificación formal en ACL2 y prueba formal mediante el demostrador de ACL2.

MIL-STD 188-220

Especificación formal en Estelle del protocolo de comunicación 188-220 de las fuerzas armadas de EE.UU. Se generaron casos de prueba automáticamente a partir de la especificación que incrementaron en un 200 % la cobertura.

¿Es artesanía? (IV)

Proyecto CICS de IBM:

- Proyecto entre IBM GB y el Laboratorio de Computación de la Universidad de Oxford.
- El 30 % del sistema se especificó en Z y luego de implementó normalmente.
- El 70 % restante se desarrolló normalmente.
- Por cada error en el 30 % especificado en Z se encontraron 10 en la otra parte.
- Ambas partes se terminaron proporcionalmente en el mismo tiempo.

¿Es artesanía? (V)

Línea 14 Meteor, sin conductor, del subterráneo de París:

- Software de control desarrollado por Alstom (Francia).
- Más de 110.000 líneas de código B.
- 86.000 líneas de código Ada.

No se detectaron errores en las etapas posteriores a la demostración formal

No se han detectado errores desde que la línea entró en funcionamiento en 1998

¿Es artesanía? (V)

Línea 14 Meteor, sin conductor, del subterráneo de París:

- Software de control desarrollado por Alstom (Francia).
- Más de 110.000 líneas de código B.
- 86.000 líneas de código Ada.

No se detectaron errores en las etapas posteriores a la demostración formal

No se han detectado errores desde que la línea entró en funcionamiento en 1998

¿Es artesanía? (V)

Línea 14 Meteor, sin conductor, del subterráneo de París:

- Software de control desarrollado por Alstom (Francia).
- Más de 110.000 líneas de código B.
- 86.000 líneas de código Ada.

No se detectaron errores en las etapas posteriores a la demostración formal

No se han detectado errores desde que la línea entró en funcionamiento en 1998

¿Es artesanía? (V – cont.)

Software crítico para la industria ferroviaria

En la actualidad Alstom y Siemens (80 % del mercado de subterráneos) desarrollan sus software críticos con B.

Aeropuerto Charles de Gaulle

ClearSy (Francia) ha utilizado B para desarrollar para Siemens el software de control del tren automático del Aeropuerto Charles de Gaulle (150.000 líneas de código).

¿Es artesanía? (V – cont.)

Software crítico para la industria ferroviaria

En la actualidad Alstom y Siemens (80 % del mercado de subterráneos) desarrollan sus software críticos con B.

Aeropuerto Charles de Gaulle

ClearSy (Francia) ha utilizado B para desarrollar para Siemens el software de control del tren automático del Aeropuerto Charles de Gaulle (150.000 líneas de código).

¿Es artesanía? (VI)

Microsoft's Protocol Documentation Program:

- 222 protocolos/documentos técnicos testeados.
- 36.875 requerimientos testeables convertidos en especificaciones de test.
  - 69 % testeados mediante métodos formales.
  - 31 % testeado usando testing tradicional.
- 66.962 días-hombre (más de 250 años).
- Spec Explorer (gratis con Visual Studio Gallery).

Mejora del 42 % en la eficiencia de testing

¿Es artesanía? (VI)

Microsoft's Protocol Documentation Program:

- 222 protocolos/documentos técnicos testeados.
- 36.875 requerimientos testeables convertidos en especificaciones de test.
  - 69 % testeados mediante métodos formales.
  - 31 % testeado usando testing tradicional.
- 66.962 días-hombre (más de 250 años).
- Spec Explorer (gratis con Visual Studio Gallery).

Mejora del 42 % en la eficiencia de testing

CONICET
CIFASIS

Es más artesanía que ingeniería

Hoy

Sin embargo, a pesar de la aparente paridad entre casos de éxito y fracaso, globalmente en la actualidad, la producción de software es esencialmente una actividad artesanal.

El futuro

- En la academia desde hace muchos años se conocen varias técnicas que podrían transformar la producción de software en una ingeniería.
- Por diferentes razones la industria no las acepta.

Es más artesanía que ingeniería

Hoy

Sin embargo, a pesar de la aparente paridad entre casos de éxito y fracaso, globalmente en la actualidad, la producción de software es esencialmente una actividad artesanal.

El futuro

- En la academia desde hace muchos años se conocen varias técnicas que podrían transformar la producción de software en una ingeniería.
- Por diferentes razones la industria no las acepta.

¿Por qué la producción de software es así?

Posibles causas:

1. No se atacan las dificultades esenciales de la producción de software.
2. Es una disciplina inmadura debido a su corta historia.
3. Es esencialmente diferente a las otras ingenierías.

Dificultades esenciales de la producción de software

¿Dónde están las dificultades?

- La parte difícil de construir software es especificarlo, diseñarlo y verificarlo, no la tarea de representarlo.
- Es decir, lo difícil es saber qué hay que hacer y cómo dividirlo en pequeñas partes, no implementar esas partes.

La predicción de Brooks – 1986

No existe un único desarrollo tecnológico, que por sí sólo pueda prometer, dentro de una década, un avance si quiera de un orden de magnitud en productividad, confiabilidad o simplicidad.

Dificultades esenciales de la producción de software

¿Dónde están las dificultades?

- La parte difícil de construir software es especificarlo, diseñarlo y verificarlo, no la tarea de representarlo.
- Es decir, lo difícil es saber qué hay que hacer y cómo dividirlo en pequeñas partes, no implementar esas partes.

La predicción de Brooks – 1986

No existe un único desarrollo tecnológico, que por sí sólo pueda prometer, dentro de una década, un avance si quiera de un orden de magnitud en productividad, confiabilidad o simplicidad.

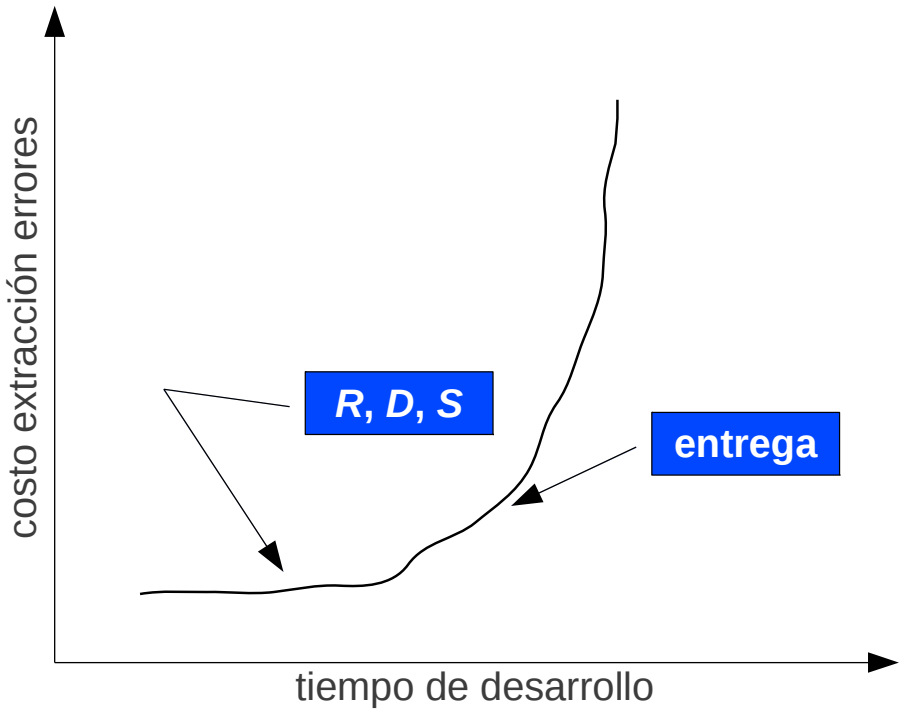
Se viene atacando lo secundario

¡Basta de cambiar de lenguaje de programación!

Los lenguajes de programación o los IDE no atacan la esencia.

Se cumple la predicción de Brooks: Java o .NET no produjeron un avance de un orden de magnitud en diez años.

Costo de extracción de errores en función del tiempo de desarrollo



Es inmadura... ¿debería serlo?

- ¿Electrónica? ¿Biotecnología? ¿Industria aeroespacial?

- Falta de especialización.
- En la mayoría de los casos carece de diseños normalizados.
- Se habla siempre de construir sistemas y no de construir dispositivos.

Es esencialmente diferente a las otras ingenierías

- La ciencia (formal) que subyace a la Ingeniería de Software es la lógica formal.
- La ciencia (fáctica) que subyace a las ingenierías tradicionales es la física.
- Esto es una diferencia cualitativa esencial.
- En la mayoría de las otras ingenierías el ingeniero se concentra en definir la solución.
- Los ingenieros de software deben concentrarse también en definir el problema.

El proceso de desarrollo del software

Proceso de desarrollo

Definición

El proceso que se sigue para construir, entregar y hacer evolucionar el software, desde la concepción de una idea hasta la entrega y el retiro del sistema.

- Sinónimos:
  - Ciclo de vida del desarrollo de software
  - Ciclo de vida del software
  - Proceso de software
- Propiedades: confiable, predecible y eficiente
- ISO 12207
- Proceso de desarrollo vs. modelo de proceso de desarrollo

Ordenar el proceso de desarrollo

Dividir y ordenar

- Dividir una tarea compleja en etapas manejables
- Determinar el orden de las etapas
- Criterio de transición para pasar a la siguiente etapa
  - Criterio para determinar la finalización de cada etapa
  - Criterio para comenzar y elegir la siguiente

¿Qué debemos hacer a continuación?

¿Por cuánto tiempo debemos hacerlo?

Algunos modelos de desarrollo

- Existen varios modelos de desarrollo
- Cada uno tiene sus ventajas y desventajas
- Cada equipo debe seleccionar el que más le sirva
- Combinarlos puede ser una buena opción
- Los más comunes son:
  - Modelo de cascada
  - Modelo de espiral
  - Desarrollo iterativo e incremental
  - Desarrollo ágil
  - Modelo de transformaciones formales

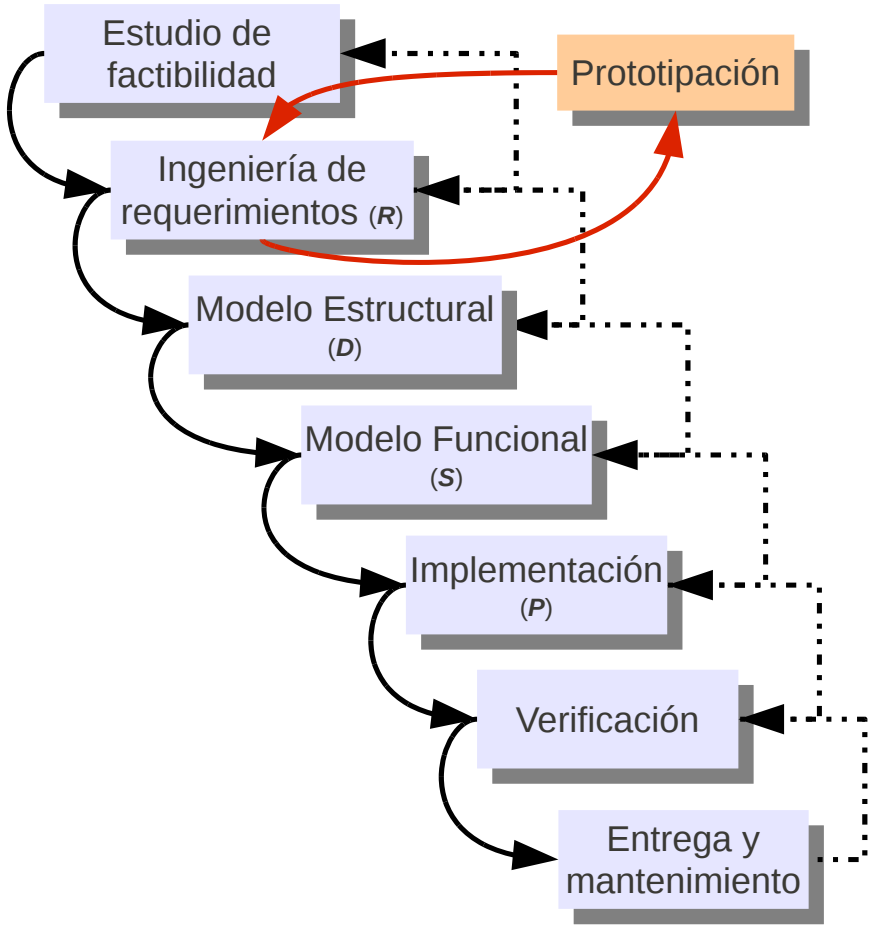
Modelo de cascada

Primera versión

- Flujo secuencial entre las etapas
- Cada etapa tiene una entrada y una salida
- Para comenzar con una etapa deben haber finalizado las anteriores

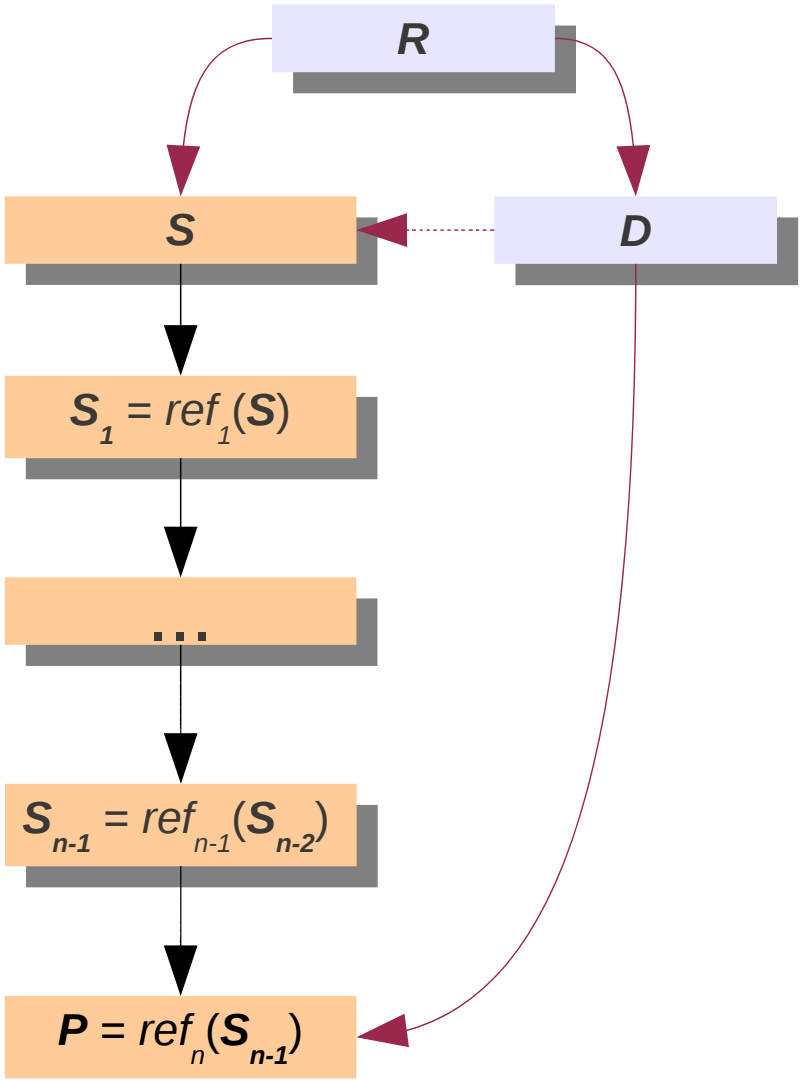
Versión actual

- Es posible volver a las etapas anteriores
- No es necesario haber terminado con las anteriores
- La verificación no se hace solo al final



Modelo de transformaciones formales

- S se transforma progresivamente en un modelo menos abstracto
  - – abstracto = + concreto = + refinado
- Cada ref_i es una transformación formal que devuelve un modelo más concreto
- $P \Rightarrow S_{n-1} \Rightarrow \cdots \Rightarrow S_1 \Rightarrow S$
- La transformación final devuelve P
- De esta forma P es correcto por construcción



CIFASIS

Desarrollo ágil

- Se basa en desarrollo iterativo e incremental
- Descompone las tareas en pequeños incrementos con mínima planificación
- Las iteraciones duran entre 2 y 4 semanas
- En cada iteración un equipo realiza *R*, *D*, *P* y verificación para implementar un incremento
- El sistema resultante se muestra al cliente
- Comunicación cara a cara más que documentos técnicos
- El equipo incluye a un representante del cliente
- Software que funciona es la medida de progreso
- Técnicas que se usan: xUnit, *pair programming*, *test driven development*, patrones de diseño, etc.

Introducción a los métodos formales

CONICET
CIFASIS

Métodos formales

- Lenguajes, técnicas y herramientas basadas en matemática y/o lógica para describir y verificar sistemas de software
- Comprenden:
  - Lenguajes de especificación formal
  - Verificación de modelos (*model checking*)
  - Prueba de teoremas
  - Testing basado en modelos
  - Cálculo de refinamiento
- Varios estándares internacionales exigen el uso de métodos formales: RTCA DO-178B, IEC SCAISRS, ESA SES, etc.

Lenguajes de especificación formal

- Una sintaxis formal y estandarizada
- Una semántica formal descripta en términos operativos, denotacionales o lógicos
- Un aparato deductivo, también formal, que permite manipular los elementos del lenguaje según su sintaxis para demostrar teoremas.

Especificación funcional (S)

Los lenguajes de especificación formal se usan casi siempre para escribir la especificación funcional de un programa.

Lenguajes de especificación formal

- Una sintaxis formal y estandarizada
- Una semántica formal descripta en términos operativos, denotacionales o lógicos
- Un aparato deductivo, también formal, que permite manipular los elementos del lenguaje según su sintaxis para demostrar teoremas.

Especificación funcional (S)

Los lenguajes de especificación formal se usan casi siempre para escribir la especificación funcional de un programa.

Ejemplo 1: Z – lógica y teoría de conjuntos

$[NCTA] \qquad SALDO == \mathbb{N}$

Banco

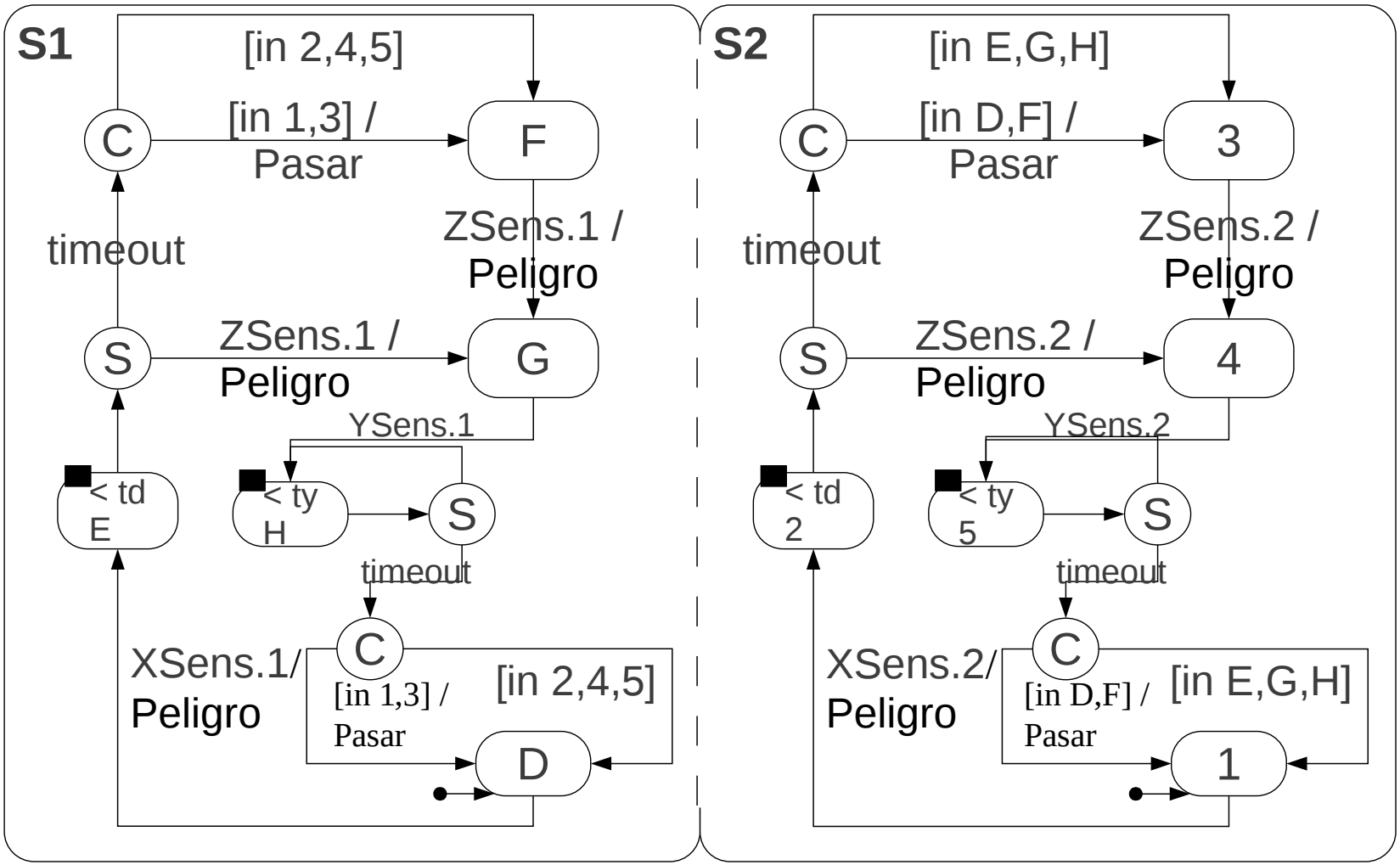
$$cajas : NCTA \pfun SALDO$$

DepositarOk

$$\Delta Banco$$
$$num? : NCTA$$
$$monto? : \mathbb{Z}$$

$$num? \in \dom cajas$$
$$monto? > 0$$
$$cajas' = cajas \oplus \{num? \mapsto cajas\ num? + monto?\}$$

Ejemplo 2: Statecharts – máquinas de estado



Ejemplo 2: CSP – álgebra de procesos

$$BUFFER = long?n + 1 \rightarrow B(n+1, \langle\rangle)$$

$$B(m+1, \langle\rangle) = left?n : \mathbb{N} \rightarrow B(m, \langle n \rangle)$$

$$B(m+1, s \frown \langle y \rangle) =$$

$$left?n : \mathbb{N} \rightarrow B(m, \langle n \rangle \frown s \frown \langle y \rangle)$$
$$\mid right!y \rightarrow B(m+2, s)$$

$$B(0, s \frown \langle y \rangle) = right!y \rightarrow B(1, s)$$

$$SYSTEM = PRODUCER \parallel BUFFER \parallel CONSUMER$$

Especificación funcional (S)

- ¿Qué tiene que hacer el programa?
- S es abstracta, no se puede ejecutar
- S es un modelo del programa
- De una u otra forma S es una fórmula de lógica
- S es independiente del lenguaje de programación
- Los programadores tienen que leer S para escribir P
- Más o menos: $P \Rightarrow S$
  - P también, de una u otra forma, es una fórmula de lógica

S es el criterio de corrección para P

CONICET CIFASIS

Especificación funcional (S)

- ¿Qué tiene que hacer el programa?
- S es abstracta, no se puede ejecutar
- S es un modelo del programa
- De una u otra forma S es una fórmula de lógica
- S es independiente del lenguaje de programación
- Los programadores tienen que leer S para escribir P
- Más o menos: $P \Rightarrow S$
  - P también, de una u otra forma, es una fórmula de lógica

S es el criterio de corrección para P

CONICET CIFASIS

Aprenda a escribir *S*

- Trate de NO pensar como un programador
  - *S* no es un programa: no hay asignaciones, no hay bucles, no hay referencias, no hay métodos
  - Hay igualdades, eventos, estados, variables, conjuntos, funciones matemáticas
- Trate de NO pensar computacionalmente
  - *S* no se hace para una computadora
  - *S* es como una ecuación de física:

$$F = m * a$$

No se asigna $m * a$ a F; m no es un `int` ni un `float`.

- Describa sólo los fenómenos esenciales de la interfaz entre el entorno y el sistema

Aprenda a escribir *S*

- Trate de NO pensar como un programador
  - *S* no es un programa: no hay asignaciones, no hay bucles, no hay referencias, no hay métodos
  - Hay igualdades, eventos, estados, variables, conjuntos, funciones matemáticas
- Trate de NO pensar computacionalmente
  - *S* no se hace para una computadora
  - *S* es como una ecuación de física:

$$F = m * a$$

No se asigna $m * a$ a F; m no es un `int` ni un `float`.

- Describa sólo los fenómenos esenciales de la interfaz entre el entorno y el sistema

Aprenda a escribir *S*

- Trate de NO pensar como un programador
  - *S* no es un programa: no hay asignaciones, no hay bucles, no hay referencias, no hay métodos
  - Hay igualdades, eventos, estados, variables, conjuntos, funciones matemáticas
- Trate de NO pensar computacionalmente
  - *S* no se hace para una computadora
  - *S* es como una ecuación de física:

$$F = m * a$$

No se asigna $m * a$ a F; m no es un `int` ni un `float`.

- Describa sólo los fenómenos esenciales de la interfaz entre el entorno y el sistema

High-quality software is not expensive. High-quality software is faster and cheaper to build and maintain than low-quality software, from initial development all the way through total cost of ownership.

Capers Jones

El software de alta calidad no es costoso. El software de alta calidad es más rápido y más barato de construir y mantener que el software de mala calidad, incluso teniendo en cuenta el desarrollo inicial y el costo total de propiedad.

Bibliografía

J.-R. Abrial.

The B-book: Assigning Programs to Meanings.

Cambridge University Press, New York, NY, USA, 1996.

Brian Berenbach, Daniel Paulish, Juergen Kazmeier, and Arnold Rudorfer.

Software & Systems Requirements Engineering: In Practice.

McGraw-Hill, Inc., New York, NY, USA, 2009.

Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stad.

Pattern-Oriented Software Architecture — A System of Patterns.

John Wiley Press, 1996.

Paul Clements, David Garlan, Len Bass, Judith Stafford, Robert Nord, James Ivers, and Reed Little.

Documenting Software Architectures: Views and Beyond.

Pearson Education, 2002.

Bibliografía (cont.)

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides.
Patrones de diseño.
Addison Wesley, 2003.

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli.
Fundamentals of software engineering (2. ed.).
Prentice Hall, 2003.

David Harel.
Statecharts: A visual formalism for complex systems.
Sci. Comput. Program., 8:231–274, June 1987.

Michael Huth and Mark Ryan.
Logic in Computer Science: Modelling and Reasoning about Systems.
Cambridge University Press, New York, NY, USA, 2004.

Bibliografía (cont.)

Michael Jackson.

Software requirements & specifications: a lexicon of practice, principles and prejudices.

ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1995.

D. L. Parnas.

On the criteria to be used in decomposing systems into modules.

Commun. ACM, 15:1053–1058, December 1972.

B. Potter, D. Till, and J. Sinclair.

An introduction to formal specification and Z.

Prentice Hall PTR Upper Saddle River, NJ, USA, 1996.

A. W. Roscoe.

The Theory and Practice of Concurrency.

Prentice Hall PTR, Upper Saddle River, NJ, USA, 1997.