Introducción a la Ingeniería de Software

Matemática, lógica y diseño para el desarrollo de software

Maximiliano Cristiá

UNR – CIFASIS
cristia@cifasis-conicet.gov.ar

CONICET
CIFASIS

Equipo docente

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

Natalia Colussi

Andrés Krapf

Sebastián Scandolo

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`

Sitio comunidades UNR

Google: campus virtual unr ingenieria software
Clave: ingenieria2022

Canal youtube

 https://youtube.com/c/maximilianocristiais

Buscar en youtube:

maximiliano cristiá ingeniería software

NO son filmaciones de las clases

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. Modelo WRSPM para requerimientos y especificaciones
4. Statecharts
5. Propiedades de seguridad, vitalidad y equidad
6. Communicating Sequential Processes (CSP)
7. Temporal Logic of Actions (TLA)
8. $\{log\}$ (setlog)
9. 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

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 $\{log\}$ (setlog)

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

NO tienen por qué esperar hasta terminar IS 2 para empezar

`https://www.fceia.unr.edu.ar/ingsoft/tp.html`

CONICET CIFASIS

Aspectos epistemológicos de la Ingeniería de Software

¿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.

¿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

¿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.

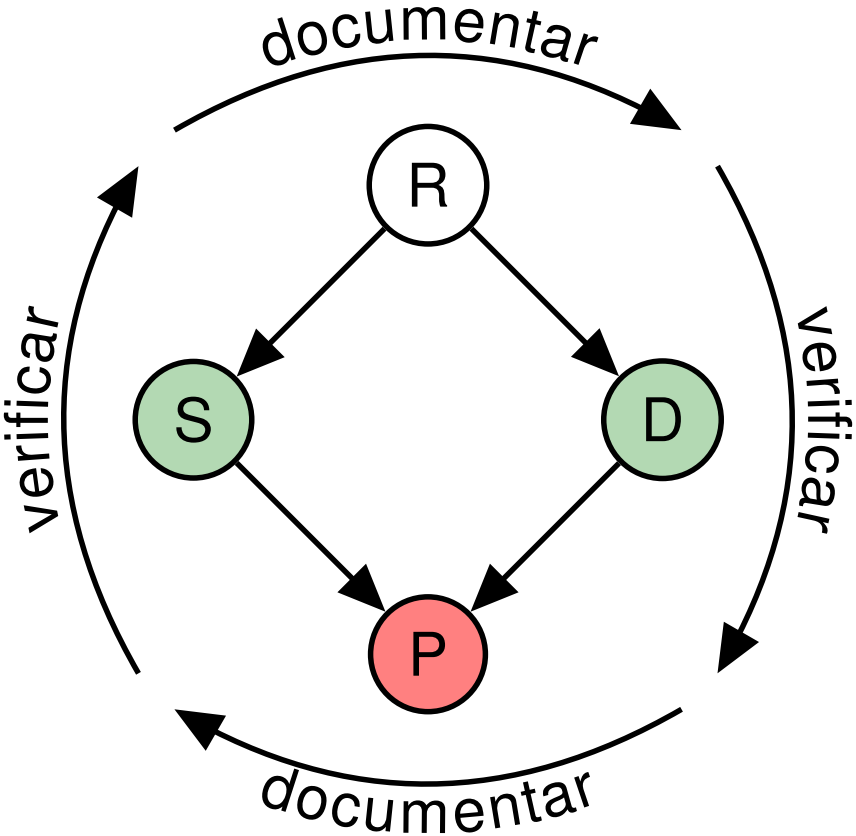
Escribe descripciones

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

Verifica las descripciones

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)



Requerimientos (R)

Requerimientos funcionales: servicios o funcionalidades que el sistema de proveer

Requerimientos no funcionales: restricciones bajo las cuales el sistema debe ser desarrollado y operar

Documento informal estructurado escrito en lenguaje natural

Especificación funcional (S)

Programa que calcula el mínimo de un arreglo

```
int min(int a[],int n) {
  min := a[1];
  for i = 2 to n {
    if a[i] < min
    then min := a[i];
  }
  return min
}
```

Especificación funcional (S)

Programa que calcula el mínimo de un arreglo

```
int min(int a[],int n) {
  min := a[1];
  for i = 2 to n {
    if a[i] < min
    then min := a[i];
  }
  return min
}
```

```
int nim(int a[],int n) {
  nim := a[n];
  i := n - 1;
  while 1 =< i {
    if nim > a[i]
    then nim := a[i];
    i := i - 1;
  }
  return nim
}
```

Especificación funcional (S)

Programa que calcula el mínimo de un arreglo

```
int min(int a[],int n) {
  min := a[1];
  for i = 2 to n {
    if a[i] < min
    then min := a[i];
  }
  return min
}
```

```
int nim(int a[],int n) {
  nim := a[n];
  i := n - 1;
  while 1 =< i {
    if nim > a[i]
    then nim := a[i];
    i := i - 1;
  }
  return nim
}
```

¿Son iguales los dos programas?

Especificación funcional (S)

Programa que calcula el mínimo de un arreglo

```
int min(int a[],int n) {
  min := a[1];
  for i = 2 to n {
    if a[i] < min
    then min := a[i];
  }
  return min
}
```

```
int nim(int a[],int n) {
  nim := a[n];
  i := n - 1;
  while 1 =< i {
    if nim > a[i]
    then nim := a[i];
    i := i - 1;
  }
  return nim
}
```

¿Son iguales los dos programas?

Lo que los hace “iguales” es la especificación funcional

CONICET CIFASIS

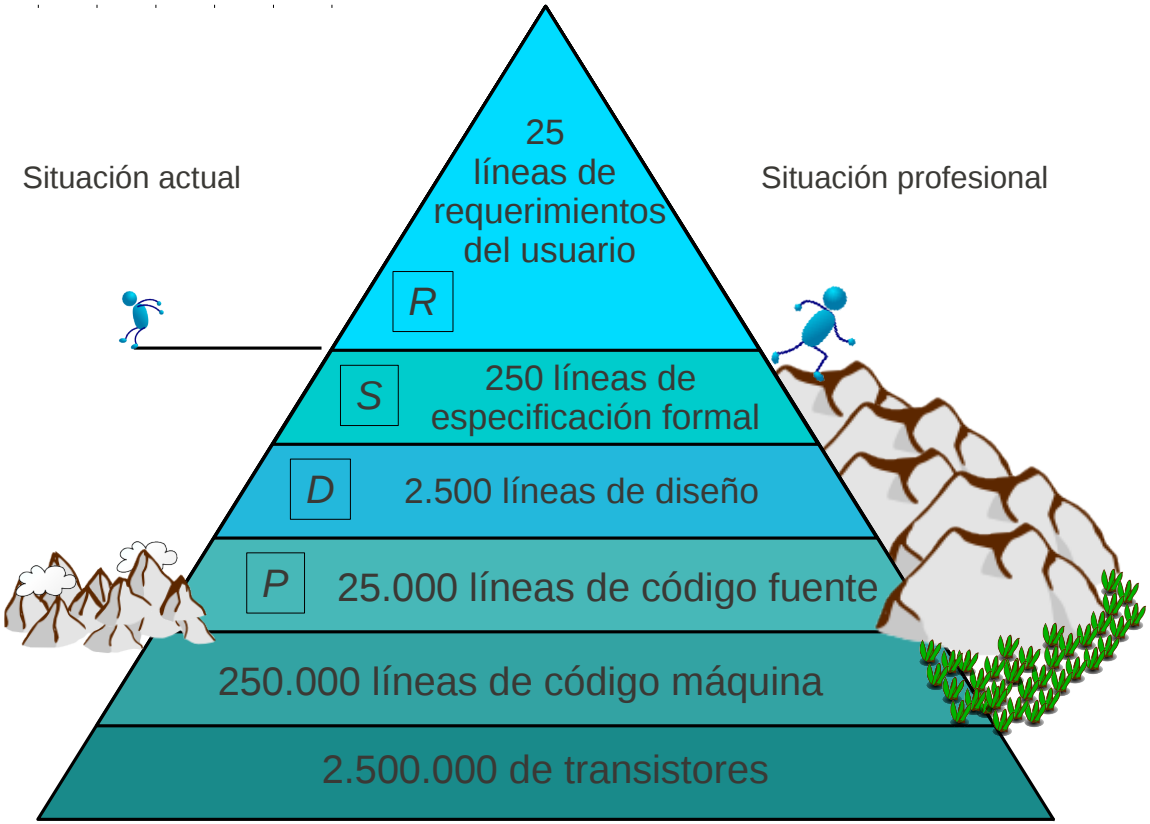
Diseño de software (D)

Descomponer el sistema en elementos de software

Asignar una función a cada elemento

Establecer las relaciones entre los elementos

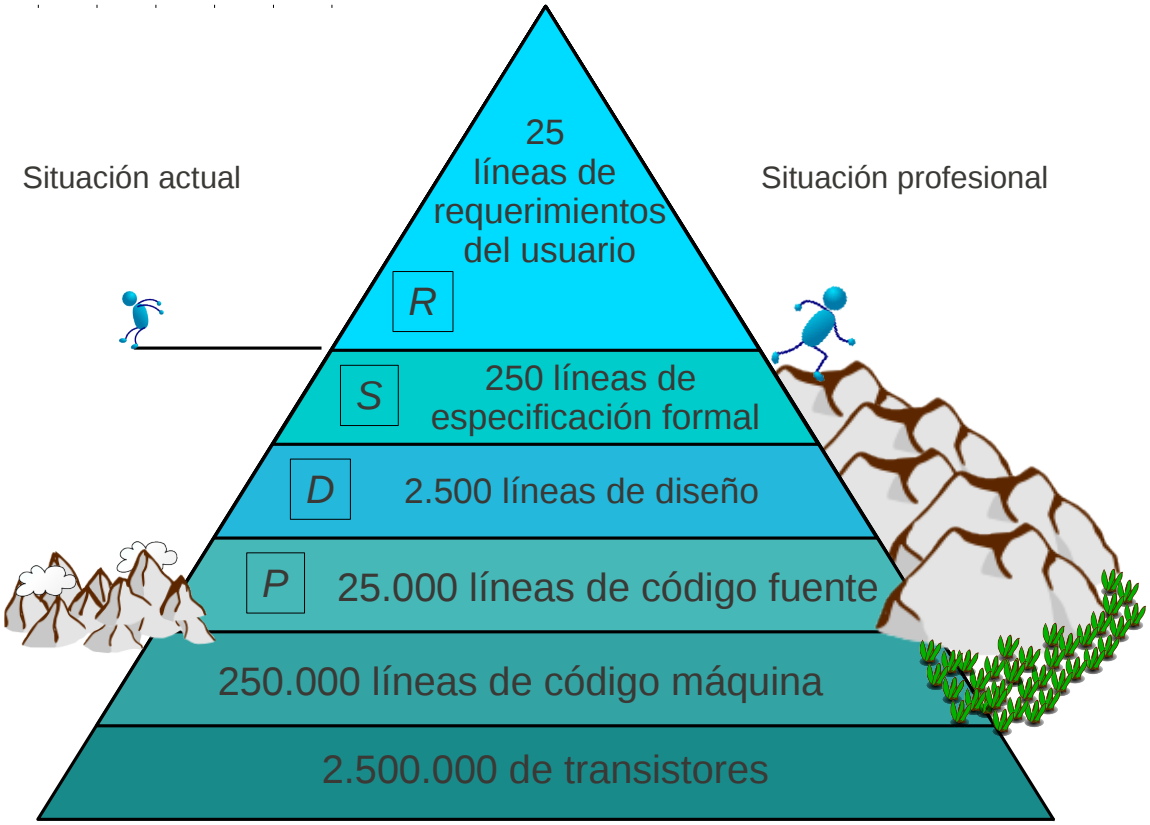
¿Por qué **cuatro** descripciones fundamentales?



Porque con menos la tarea del desarrollador es muy riesgosa

Porque con menos aumentan los costos de producción

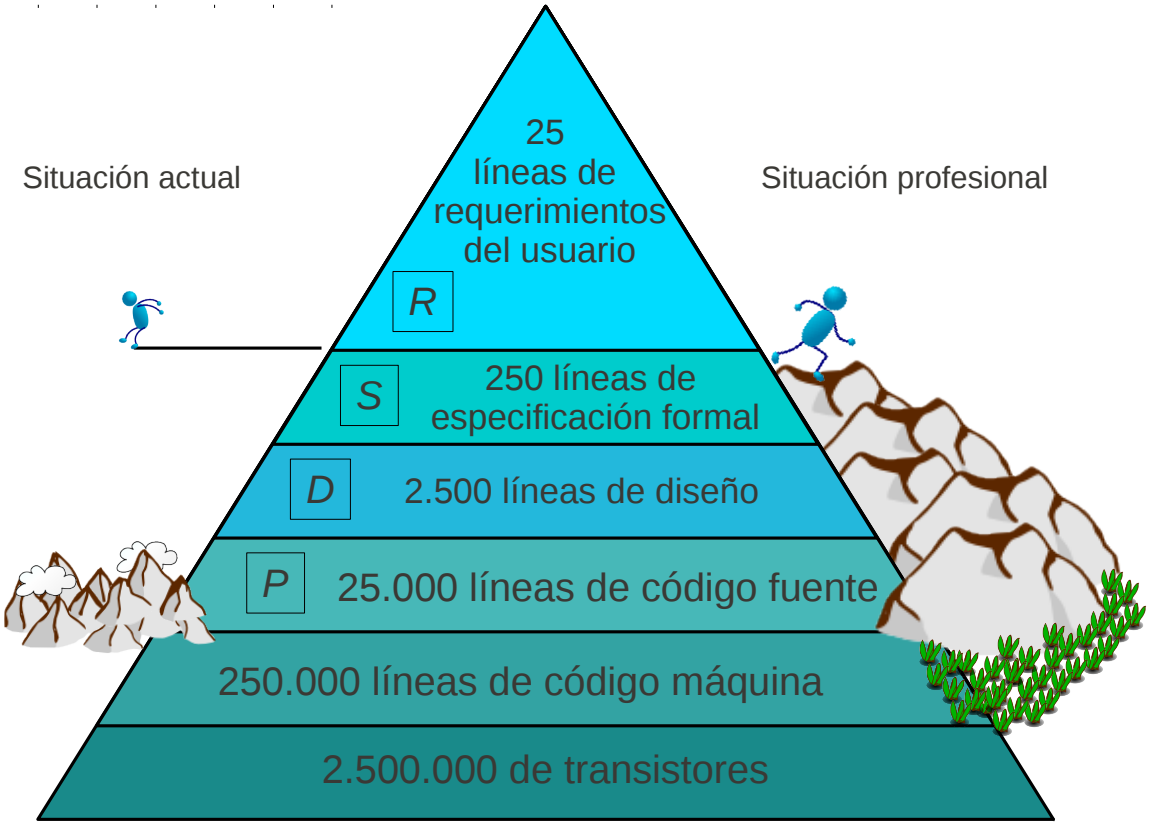
¿Por qué **cuatro** descripciones fundamentales?



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
- Entender 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
- Entender 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? (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

CONICET
CIFASIS

¿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?

- 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?

- 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?

En 2018 y 2019 dos aviones Boeing 737 MAX 8 se accidentaron produciendo la muerte de 346 personas.

Los accidentes fueron producidos por un error en el software conocido como MCAS (Maneuvering Characteristics Augmentation System).

Boeing se vio afectada de varias formas

- corregir el software y distribuirlo en todos los aviones
- pagar por el entrenamiento de los pilotos de sus clientes
- las acciones de la compañía bajaron 12 % en pocos días produciendo una descapitalización de USD 28.000 M
- para mejorar su imagen entregó a sus clientes la *disagree light*, que antes era opcional y paga

¿Es ingeniería?

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?

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?

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?

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?

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?

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?

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?

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?

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?

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?

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 ingeniería?

 Reino Unido: el escándalo de la oficina de correos – DW

 Mr Bates vs The Post Office – miniserie británica

¿Es ingeniería?

CHAOS Report is based on the collection of project case information from real-life IT environments and software projects. This version and past versions have used eight different instruments in the collection of this information, which includes project profiles, project tracking, individual project surveys, case interviews, general surveys, project postmortems, and other instruments. CHAOS research encompasses 21 years of data on why projects succeed or fail, representing more than 100,000 completed IT projects. However, for our new database we eliminated cases from 1994 through 2002, since they did not match the current requirements for analysis. The new CHAOS database has just under 50,000 projects.

CHAOS RESOLUTION FOR ALL PROJECTS

	1994	2009	2010	2011	2012	2013
SUCCESSFUL	31%	36%	38%	37%	41%	36%
CHALLENGED	53%	44%	40%	46%	40%	48%
FAILED	16%	20%	22%	17%	19%	16%

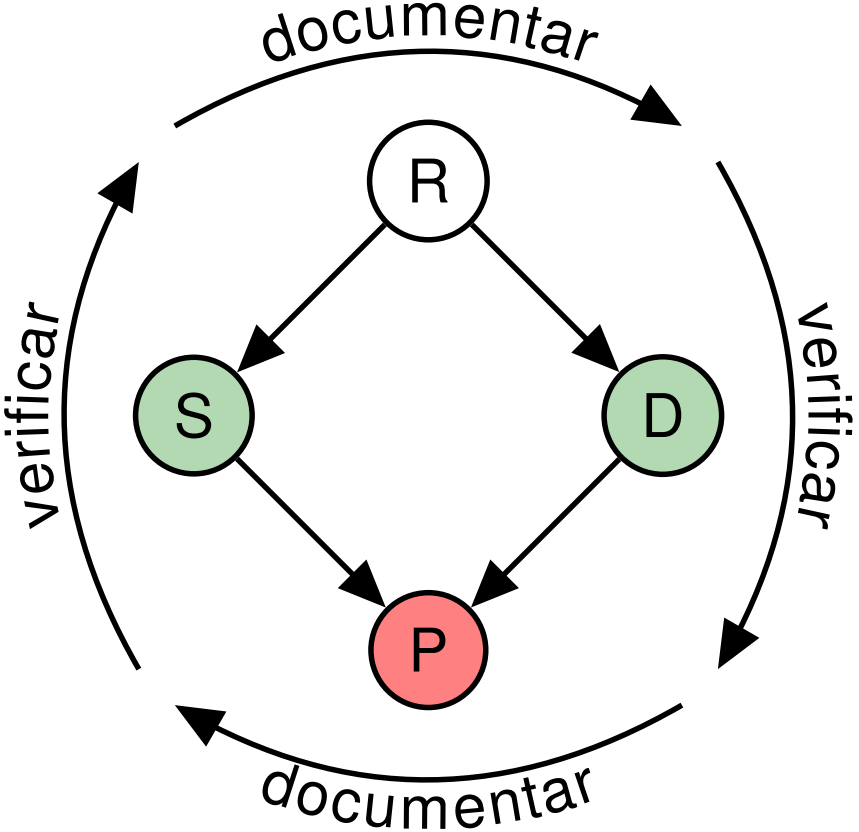
CONICET CIFASIS

¿Cómo podemos evitar los errores de programación?

CONICET
CIFASIS

Métodos Formales

¿Cómo podemos evitar los errores de programación?



Métodos formales, ingeniería aplicada al desarrollo de software

Escribir S como una fórmula **lógica**

Escribir P como una fórmula **lógica**

Demostrar $P \Rightarrow S$

Es por esto que necesitan aprender matemática

Métodos formales

Ingeniería aplicada al desarrollo de software

Escribir S como una fórmula **lógica** usando una notación formal

S será más corta que P

S será más simple que P

S será más evidentemente correcta que P

Es más fácil escribir S que P

Es más fácil escribir P partiendo de S

Si te parece difícil escribir S, entonces te va a resultar más difícil escribir P

CONICET CIFASIS

¿Métodos formales?

CDIS – AdaCore (1993)

- Parte del sistema de control de tráfico aéreo de UK
- Lenguajes de especificación: VDM y CSP
- 197K LOC
- Esfuerzo total 15K d-h; igual o mejor a proyectos similares.
- Defectos encontrados 0.75 cada 1K LOC; mucho mejor que proyectos comparables.

¿Métodos formales?

Tokeneer ID Station – AdaCore (2003)

- Sistema de control de acceso seguro
- Lenguaje de especificación Z (2K LOC)
- 9K LOC de SPARK
- Esfuerzo total 260 d-h
- Defectos encontrados 0.4 cada 1K LOC
- Productividad y calidad mejores que CDIS

¿Métodos formales?

CompCert – proyecto académico (2005), AbsInt (2015)

- Compilador C certificado, correcto por construcción, libre de errores
- Lenguaje de especificación Coq (100K LOC, incluyendo pruebas formales)
- Esfuerzo total 2.2K d-h
- Defectos encontrados **0** cada 1K LOC
- 90 % del desempeño de GCC v 4 en nivel 1

¿Métodos formales?

seL4 – General Dynamics C4 Systems (2015)

- Micronúcleo operativo de alta seguridad
- Lenguaje de especificación Isabelle (480K LOC, incluye pruebas formales)
- Esfuerzo total 7K d-h
- Defectos encontrados **0** cada 1K LOC
- Es posible realizar verificación formal de propiedades hasta el nivel código máquina a un costo razonable.

¿Métodos formales?

Parallel Commits – Cockroach (2019)

- CockroachDB's new atomic commit protocol
- Reduce latency of transactions down to only a single round-trip of distributed consensus
- Lenguaje de especificación TLA+ (1K con verificación)
- Demostraron propiedades de *vitalidad* y *seguridad*
- `https://www.cockroachlabs.com/blog/parallel-commits/#parallel-commits`

TLA+: DynamoDB, S3, Raft Consensus Algorithm

CONICET CIFASIS

¿Métodos formales?

Météor Project – Alstom (1998)

- Línea 14 del subte de París no tiene conductor
- Software de control
- Lenguaje de especificación B (110K LOC, dos especificaciones)
- 86K LOC de Ada
- Defectos encontrados **0** cada 1K LOC (desde 1998)
- El sistema fue entregado a tiempo y dentro del presupuesto.

CONICET
CIFASIS

¿Métodos formales?

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).

¿Métodos formales?

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).

¿Métodos formales?

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.

CONICET
CIFASIS

¿Métodos formales?

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.

CONICET
CIFASIS

¿Métodos formales?

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.

¿Métodos formales?

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

¿Métodos formales?

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

¿Métodos formales?

Back to the building blocks: a path toward secure and measurable software — The White House (2024)

Given the complexities of code, testing is a necessary but insufficient step in the development process to fully reduce vulnerabilities at scale.
While formal methods have been studied for decades, their deployment remains limited; further innovation in approaches to make formal methods widely accessible is vital to accelerate broad adoption. Doing so enables formal methods to serve as another powerful tool to give software developers greater assurance that entire classes of vulnerabilities, even beyond memory safety bugs, are absent.

`https://www.whitehouse.gov/wp-content/uploads/2024/02/Final-ONCD-Technical-Report.pdf`

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?

Mucho software no es “industrial”

- Sin restricciones económicas – software libre
- Se venden servicios, no software, not software – publicidad
- Nuevos productos sin mercados asegurados – redes sociales

Los usuarios está forzados a tolerar los errores, no hay alternativas

Los proveedores de software de un sector son todos igualmente malos

Las buenas prácticas, como MF, rara vez se enseñan en las universidades

CONICET CIFASIS

¿Por qué la producción de software es así?

El foco está en programar, producir software **es** programar

- Es así? Está bien? Por qué? Por qué no?

Los cambios tecnológicos adoptados por la industria atacan las dificultades aparentes de la producción de software

- Los MF atacan las esenciales
- ¿Cuáles son las dificultades esenciales de la producción de software?

La producción de software es inmadura porque es joven

- ¿Ingeniería electrónica? ¿Ingeniería aeroespacial? Biotecnología?

La industria del software es esencialmente diferente de la industria clásica

- Ciencias fácticas vs. formales; física vs. lógica y matemática

CONICET CIFASIS

Dificultades esenciales de la producción de software

F. Brooks, *No silver bullet*, 1986

La esencia del software es una relación entre diversos conceptos.

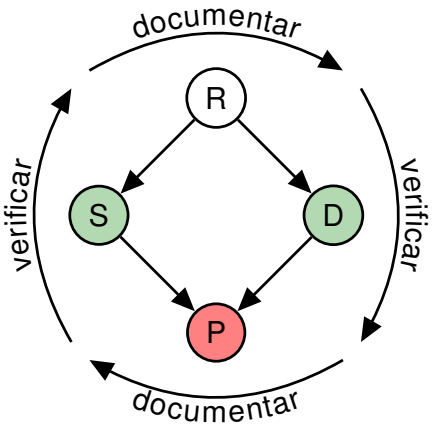
conjuntos de datos, relaciones entre datos, algoritmos

Esta esencia es abstracta; es la misma bajo diferentes representaciones.

La **parte difícil de construir software** es la

- **especificación** (S),
- **diseño** (D),
- y **verificación** (V)

de esta relación conceptual, **no** la tarea de representarla (P).



El foco de la industria es la programación

Los cambios tecnológicos adoptados por la industria atacan las dificultades aparentes de la producción de software.

La industria cree que las claves para producir software son

- lenguajes de programación nuevos y más sofisticados
- herramientas para facilitar la programación
  IDE, automatización de testing, etc.

La **parte difícil es** la especificación, diseño y verificación, **no la programación**.

Los lenguajes y herramientas de programación no atacan la parte difícil de la producción de software.

CONICET CIFASIS

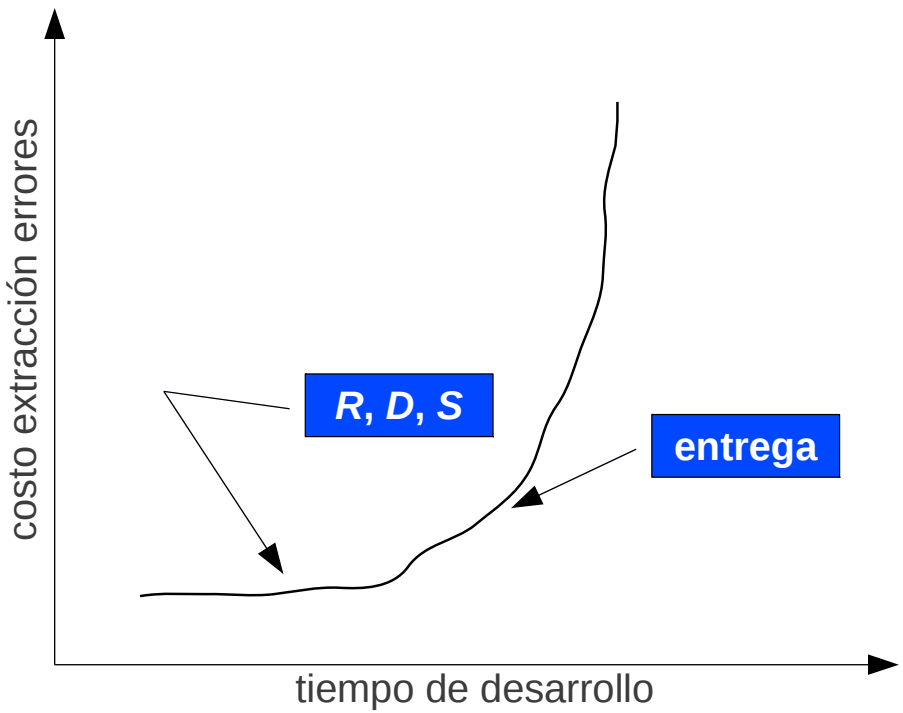
El foco de la industria es la programación

¡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



CONICET
CIFASIS

Los MF atacan las dificultades esenciales de producir de software

Atacan dos de las partes difíciles de construir software: especificación y verificación

El proceso de desarrollo de 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?

CONICET
CIFASIS

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

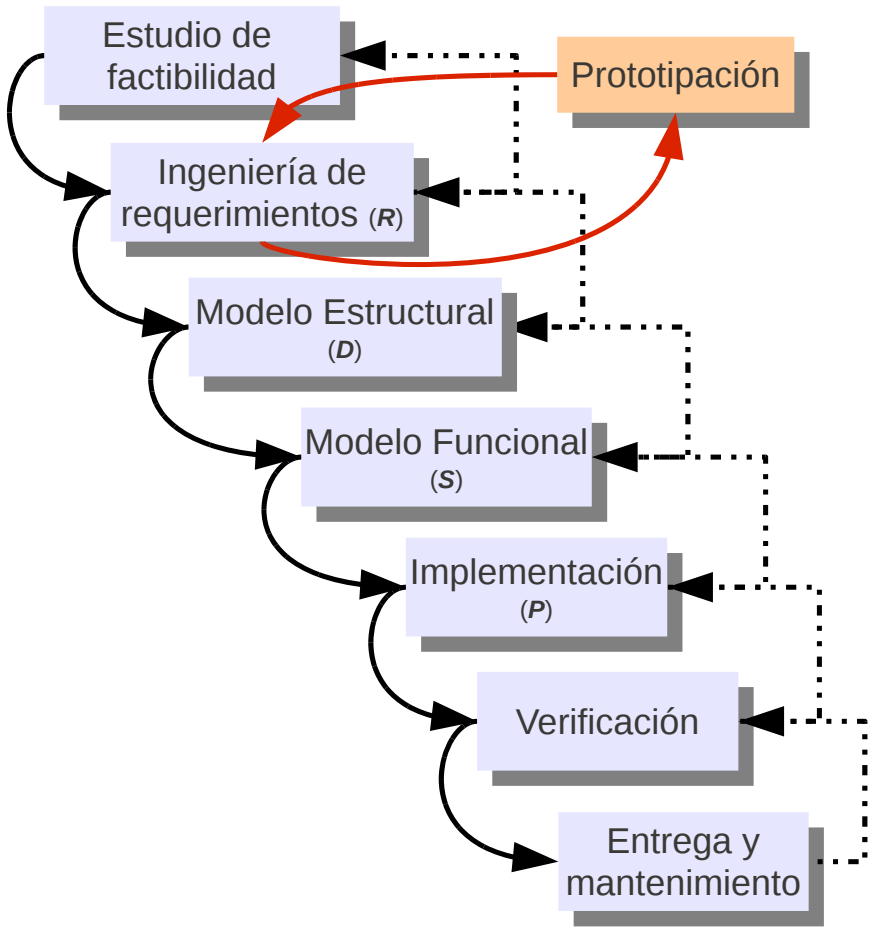
Modelo de cascada

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

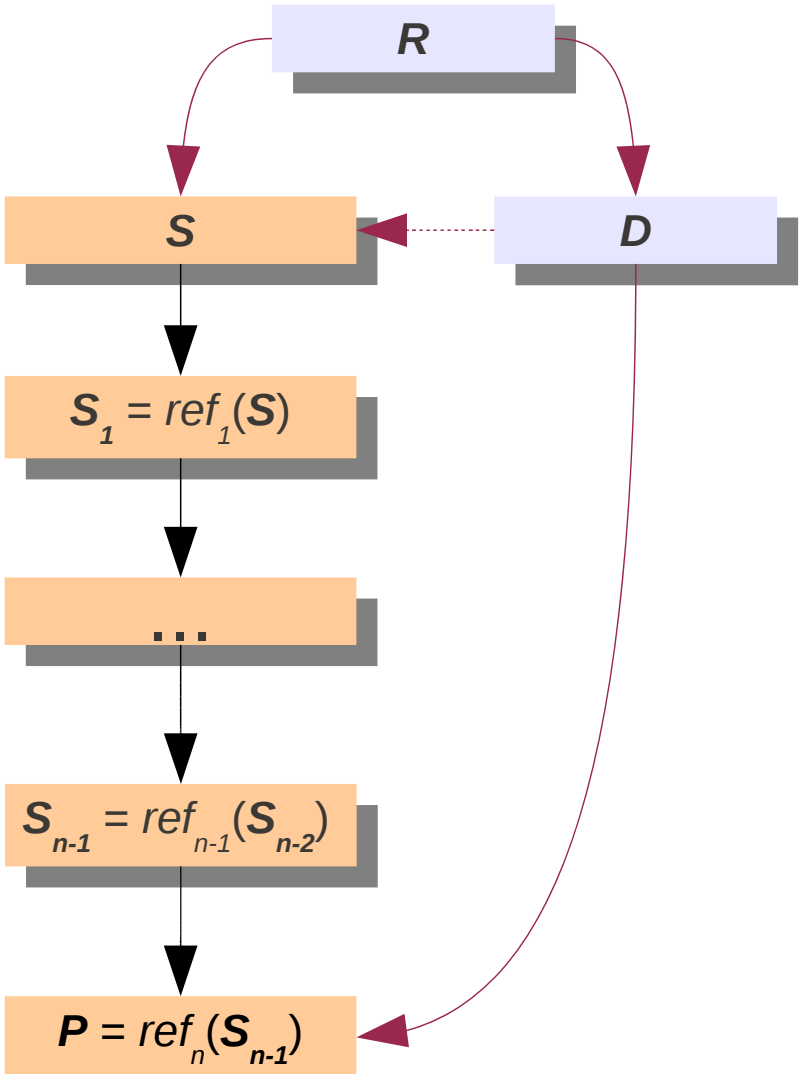
Versión actual

- 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



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

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

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

Banco

$cajas : NCTA \pfun SALDO$

DepositarOk

$\Delta Banco$

$num? : NCTA$

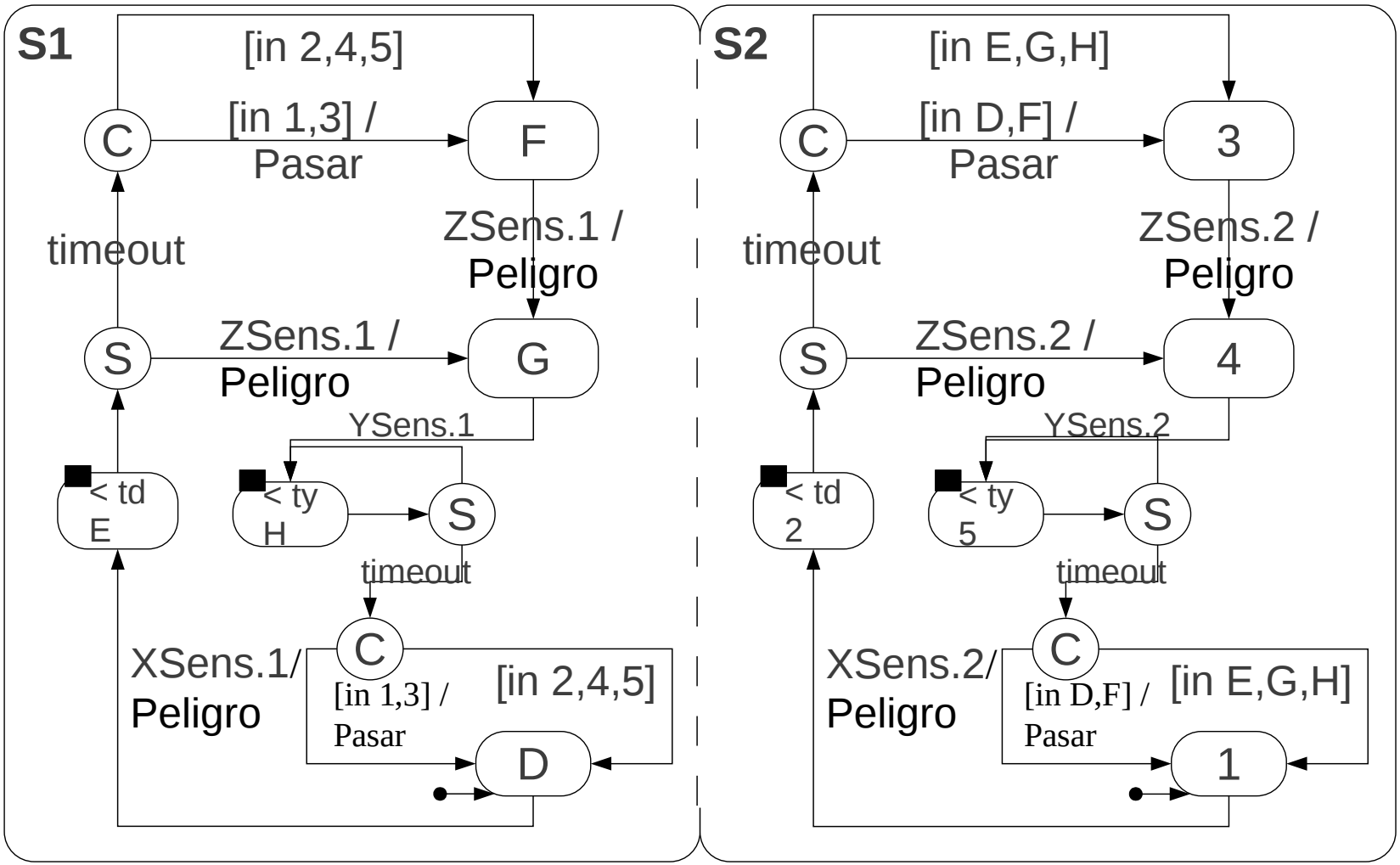
$monto? : \mathbb{Z}$

$num? \in \dom cajas$

$monto? > 0$

$cajas' = cajas \oplus \{num? \mapsto cajas\ num? + monto?\}$

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

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:

$$F = m \times a$$

No se asigna $m \times a$ 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:

$$F = m \times a$$

No se asigna $m \times a$ 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:

$$F = m \times a$$

    No se asigna $m \times a$ a F; m no es un `int` ni un `float`.
- Describa sólo los fenómenos esenciales de la interfaz entre el entorno y el sistema

Formalizar o no formalizar, esa **no** es la cuestión

Porque siempre van a formalizar...
cuando escriban el programa

La pregunta es: ¿cuántas veces van a formalizar?

CONICET
CIFASIS

Formalizar o no formalizar, esa **no** es la cuestión

Porque siempre van a formalizar...

cuando escriban el programa

La pregunta es: ¿cuántas veces van a formalizar?

Formalizar o no formalizar, esa **no** es la cuestión

Porque siempre van a formalizar...
cuando escriban el programa

La pregunta es: ¿cuántas veces van a formalizar?

Formalizar o no formalizar, esa **no** es la cuestión

Porque siempre van a formalizar...
cuando escriban el programa

La pregunta es: ¿cuántas veces van a formalizar?

CONICET
CIFASIS

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.