Métodos Formales y Análisis de Herramientas para la Producción de Software

Aristides Dasso, Ana Funes

{arisdas, afunes}@unsl.edu.ar
Universidad Nacional de San Luis

Universidad Nacional de San Luis
2006

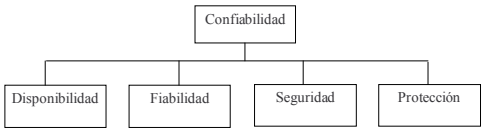
Objetivos del Curso

- Introducción a los métodos formales con especial énfasis en el uso del lenguaje de especificación del método RAISE (RSL).
- Ganar experiencia práctica en la escritura de especificaciones formales en RSL, en particular haciendo uso de un estilo aplicativo secuencial.
- Obtener experiencia en el uso de las herramientas de RAISE.

Bibliografía básica

- Transparencias
  Sitio de la maestría: url: http//www.sel.unsl.edu.ar
- Libros:
  - The RAISE Development Method
  - The RAISE Specification Language
  - Specification Case Studies in RAISE
- Sitio del UNU/IIST (herramientas, reports, etc.):
  url: http://www.iist.unu.edu.ar

Confiabilidad

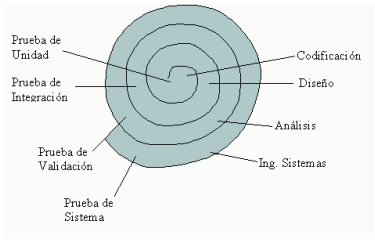


- ¿Con qué frecuencia colapsa el sistema?
- ¿Hace lo que se supone que debe hacer?
- ¿Cuán probable que produzca algún tipo de daño es?
- ¿Cuán susceptible es a los ataques?

¿Cómo lograrla?

- Aseguramiento de la calidad del SW
  - Gestión de la configuración del SW
  - Planificación y control
  - Evaluación y control de riesgos
  - Revisiones del sw
  - Pruebas del sw.

Testing en el Proceso de Desarrollo de Software



- Uso de FM:
  - Encontrar errores cuanto antes.
    - Concentrarnos en las etapas iniciales: captura y análisis de requerimientos.
      - FM ayudan a eliminar inconsistencias e incompletitud.
  - Razonar sobre las propiedades del sistema.
    - FM permiten "demostrar" propiedades.

Terminología

- Prueba (test, testing). Acto de 'ejercitar' el software.
- Demostración ≡ Proof

Métodos Formales (**F**ormal **M**ethods)

"Los métodos formales que se utilizan para desarrollar sistemas de computadoras son técnicas de base matemática para describir propiedades del sistema. Estos métodos formales proporcionan marcos de referencia en el seno de los cuales las personas pueden especificar, desarrollar y verificar los sistemas de manera sistemática, en lugar de hacerlo ad-hoc".

Software Engineering Encyclopedia (Marcianiack, JJ; 1994).

Métodos de Desarrollo de Software

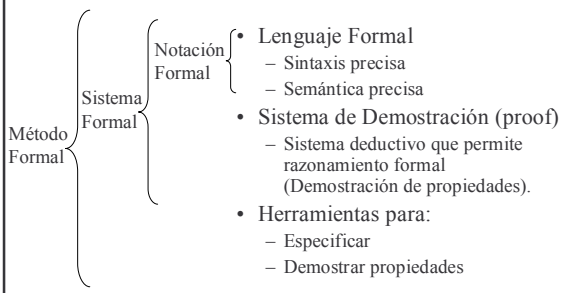
- Ad-hoc
- Sistemáticos
- Rigurosos
- Formales

Métodos Formales (**F**ormal **M**ethods)

"Mathematically based techniques for the specification, development and verification of software and hardware systems."

http://foldoc.doc.ic.ac.uk/foldoc.

FM, Sistema Formal y Notación Formal *(Alagar & Periyasamy)*



Métodos Formales: Estilos de Desarrollo

- Transformacional
  (p.e. SETS)
- "Invent and Verify"
  (p.e. VDM, Z, RAISE, B)

Grados de Rigor (1) (NASA's Langley Research Center)

Nivel 1: Especificación formal de todo o parte del sistema.

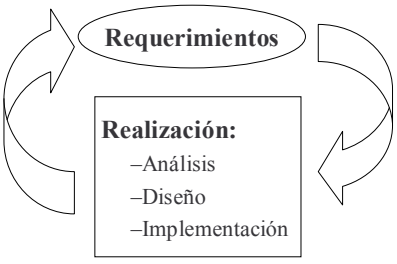
Grados de Rigor (2) (NASA's Langley Research Center)

Nivel 2: Especificación formal en dos o más niveles de abstracción y pruebas en lápiz y papel de que la especificación detallada implica la más abstracta.

Grados de Rigor (3) (NASA's Langley Research Center)

Nivel 3: Prueba formal realizada usando un probador de teoremas semi-automático.

Especificaciones de Software



Especificación de Requerimientos (1)

¿Cómo especificar los requerimientos?

Informalmente

Lenguaje ambigüo
(p.e. lenguaje natural)

Especificación de Requerimientos (2)

¿Cómo especificar los requerimientos?

Semiformalmente

(sintaxis más ó menos precisa,
semántica informal,
p.e. UML)

Especificación de Requerimientos (3)

¿Cómo especificar los requerimientos?

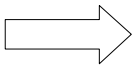
Formalmente

(sintaxis y semántica formal)

Especificación formal: modelo matemático que describe el mundo real.

Especificación

Especificación (qué) (abstracto)	Implementación (cómo) (concreto)

Especificación  Programa

Estilos de Especificaciones Formales

Orientado al modelo. Basado en dominios matemáticos. Por ej. Números, funciones, conjuntos, secuencias, etc. Concreto.	**Orientado a la propiedad.** Basado en definiciones axiomáticas. Abstracto.
Aplicativo. No permite el uso de variables.	**Imperativo u Orientado al Estado**. Permite el uso de variables.
Estático. No provee medios para manejar el tiempo.	**Acción**.El Tiempo puede ser considerado en la especificación. Varias formas de hacerlo: considerar el Tiempo como lineal o branching, sincrónico, asincrónico, etc.

FM, Pro y Contras (1)

- Especificaciones no ambiguas. ☺
- Demostración formal de propiedades. ☺
- Comprensión profunda del problema => reducción de errores y omisiones. ☺ => incremento en la confiabilidad. ☺
- Generación automática de código. ☺

FM, Pro y Contras (2)

- Verificación es costosa y consume tiempo. ☹
- No se adaptan a todo tipo de sistema. ☹
- Pocos expertos. ☹
- Dificultan la validación con el cliente. ☹

Combinación de técnicas 

Integración de notaciones formales con otras técnicas

- *Suplemental*, notaciones informales enriquecidas con conceptos formales.
- *Extensión*, notaciones formales extendidas con conceptos de otros paradigmas, por ejemplo del paradigma oo.
- *Interface*, se provee a las notaciones formales con interfaces gráficas para ayudar en el desarrollo de modelos.
- *Semántica,* la semántica de un lenguaje de modelado semi-formal viene dada por un lenguaje formal.

Los 10 mandamientos de los FM (1)

(Bowen y Hinchley; 1995)

I. Seleccionarás la notación adecuada.
II. Formalizarás, pero no de más.
III. Estimarás los costes.
IV. Poseerás un experto en FM a tu disposición.
V. No abandonarás tus métodos formales de desarrollo.

Los 10 mandamientos de los FM (2)

VI. Documentarás suficientemente.
VII. No comprometerás los estándares de calidad.
VIII. No serás dogmático.
IX. Probarás, probarás y volverás de probar.
X. Reutilizarás cuanto puedas.

Algunas aplicaciones de FM

- T800 Transputer.
- IBM CICS transaction processing system.
- Tektronic Co., oscilloscopes.
- Airbus A330/340.
- Darlington Nuclear Facility.
- NASA (http://shemesh.larc.nasa.gov/fm/fm-main-research.html)
  - Small Aircraft Transportation System (SATS).
  - Formal Analysis of Airborne Information for Lateral Spacing (AILS).

RAISE

¿Qué es **RAISE**?

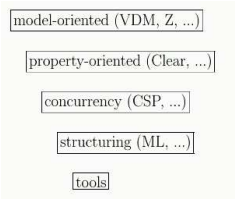
Rigorous **A**pproach to **I**ndustrial **S**oftware **E**ngineering

Consiste de:

- Un método para desarrollar software.
- Un lenguaje de especificación formal (RSL).
- Conjunto de herramientas.

Antecedentes de RAISE (1)

- ESPRIT Project RAISE, 1985-90.



Antecedentes de RAISE (2)

- ESPRIT Project LaCoS (**La**rge **S**cale **Co**rrect **S**ystems), 1990-95.
  - Evolución del método, lenguaje y herramientas.
  - Aplicaciones industriales de RAISE.

BNR Europe (UK):	Network design toolset
Lloyd's Register (UK):	Ship engine monitoring; security
Bull (F):	Database; security
MATRA Transport (F):	Automatic train protection
Inisel Espacio (E):	Image processing
Space Software Italia (I):	Tethered satellite; air traffic control
Technisystems (GR):	Shipping transaction processing

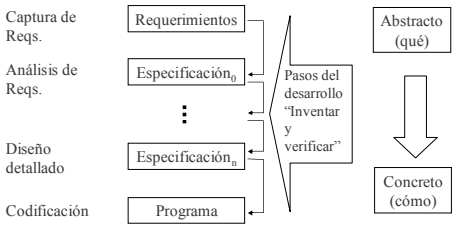
Características del método RAISE

- Desarrollo separado
- Desarrollo incremental
- Inventar y Verificar
- Riguroso

Grados de Rigor en RAISE

1. ***Sólo especificación formal*:** formalidad aplicada sólo al proceso de especificación.
2. ***Especificación formal y desarrollo riguroso*:** 1 + rigor en el proceso de desarrollo (se mantienen las relaciones de desarrollo y se las examina, sin embargo no son justificadas).
3. ***Especificación y desarrollo formal*:** 2 + más justificación.

Método RAISE (1)



Método RAISE (2)

- Especificación Inicial E_0 ,Crítica.
  - No somos expertos en el domino.
  - Reqs. en lenguaje natural, por personas diferentes.
    - Ambiguos
    - Inconsistentes
    - Incompletos
    - Vagos
- ¿Qué hacer para construir una buena especificación inicial?

Método RAISE (3)

- Ser abstracto.
- Usar el vocabulario del cliente.
- Hacerla legible.
- Buscar problemas.
- Identificar condiciones de consistencia y políticas.

Validación y Verificación

- Validación: ¿Estamos construyendo el producto correcto?
- Verificación: ¿Estamos construyendo correctamente el producto?

Validación (1)

- Paso importante.
- No puede ser formal.
  - Ppal. técnica: Chequear que cada requerimiento ha sido tenido en cuenta.
    - Si, no hay problema.
    - No, necesitamos cambiar la especificación.
    - No porque pensamos que no es una buena idea (imposibilidad, inconsistencia).
    - No porque es diferido para más adelante (requerimientos no funcionales, p.e. Sist. Op., lenguaje, performance, etc.)

Validación (2)

- Otras técnicas:
  - Buscar en la especificación propiedades que deberían estar en los reqs. pero que no están.
  - Desarrollar casos de prueba (y resultados esperados) junto con la especificación para el cliente.
  - Re-escribir los reqs. a partir de la especificación.
  - Construir un prototipo de todo o parte del sistema para correr algunos casos de prueba.
- Objetivo: buscar el compromiso del usuario.

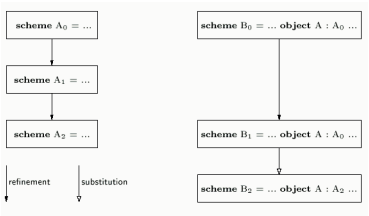
Verificación (1)

- Controlar que estamos desarrollando el sistema correctamente.
- Viene después de la validación. Asume la correctitud de de E_0.
- Basada en "Inventar y Verificar"
  - Relación de Refinamiento o Implementación.

Verificación (2)

Relación de Refinamiento o Implementación

- Transitividad
- Monotonicidad
  - B_2 es un ref. de B_1.
  - A_0 es un contrato entre los des. de A y B.



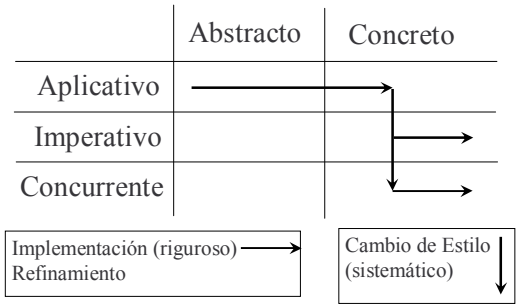
Relación de Refinamiento, condiciones

- El nuevo modelo (A_1) debe incluir TODAS las entidades del viejo (A_0)
  types, values, objects, variables y channels con los mismos nombres y tipos maximales.
- A_1 puede tener más entidades que A_0.
- Implementación estática, garantiza monotonicidad.
- Puede ser chequeada estáticamente.

Relación de Refinamiento, condiciones

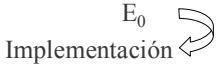
- Todas las propiedades de A_0 se mantienen en A_1.
  - axiomas, defs. de funciones, constantes, restricciones en subtipos.
- No puede ser chequeada estáticamente, necesita demostración (Recordar que RAISE es riguroso.)

Estilos y Ruta de Desarrollo en RAISE



Lightweight FM

- Uso de FM sin pruebas ni refinamientos.



- Especificaciones iniciales simples.

Herramientas de RAISE

- Type-checker
- Confidence condition generator
- Pretty printer
- Traductores (C++, Ada, SML ...)
- Demostrador

Confidence Conditions (1)

Propiedades que deberían ser probablemente verdaderas si el módulo no es inconsistente pero que no pueden determinarse como verdades o falsas por medio de una herramienta automática.

Confidence Conditions (2)

- Precondiciones de funciones parciales son satisfechas.
- Valores que se suponen deben ser subtipos, son subtipos.
- Definiciones de subtipos no vacíos.
- La definición de una función parcial sin pre condición.
- La definición de una función total con una pre condición.
- …


```
1.  scheme CC =
2.    class
3.      value
4.        x1 : Int = hd <..>,
5.        x2 : Int = f1(-1),
6.        x3 : Nat = -1,
7.        f1 : Nat -~-> Nat
8.        f1(x) is -x pre x > 0
9.      type None = {| i : Nat :- i < 0 |}
10.     value
11.       x4 : Nat :- x4 < 0,
12.       f2 : Nat -> Nat
13.       f2(x) as r post r + x = 0
14.   end
```



```
CC.rsl:4:21: CC:
-- application arguments and/or precondition
let x = <..> in x ~= <..> end

CC.rsl:5:20: CC:
-- application arguments and/or precondition
-1 >= 0 /\ let x = -1 in x > 0 end

CC.rsl:6:16: CC:
-- value in subtype
-1 >= 0

CC.rsl:8:7: CC:
-- function result in subtype
all x : Nat :- (x > 0 is true) => -x >= 0
```



```
CC.rsl:9:31: CC:
-- subtype not empty
exists i : Nat :- i < 0

CC.rsl:11:10: CC:
-- possible value in subtype
exists x4 : Nat :- x4 < 0

CC.rsl:13:7: CC:
-- possible function result in subtype
all x : Nat :- exists r : Nat :- r + x = 0
```


Ventajas en el uso de RAISE

- Precisión
- Abstracción
- Razonamiento formal
- Re-uso

⇒ Menor cantidad de errores.

Resumen (3)

- La mayor parte del curso usará el estilo aplicativo, el cual es cercano a la programación funcional y a la matemática.
- El estilo imperativo es similar a la programación imperativa tradicional, con variables, asignaciones, secuencias, loops, etc.
- El estilo concurrente soporta la especificación de aspectos concurrentes.