TEMA 5. Modelado y Verificación Formal
Parte 1 de 2

Calidad del Software

Dr. José Luis Abellán Miguel

Grado en Ingeniería Informática

Dr. José Luis Abellán Miguel - Tlf: (+34) 968 278 657
Universidad Católica San Antonio de Murcia www.ucam.edu

Índice

- Introducción
- Estrategia de cuarto limpio
- Especificación funcional
- Diseño de cuarto limpio
- Conceptos de métodos formales
- Lenguajes de especificación formal

Bibliografía

- Pressman, R. ***Ingeniería del Software: Un enfoque práctico***. 7ª edición. Madrid: McGraw Hill, 2010. ISBN: 9701054733 (disponible en la biblioteca UCAM) → **Capítulo 21**

Introducción (1/2)

- Métodos de modelado y verificación formal se aplican antes/durante el desarrollo de los modelos y código
  - A diferencia de las revisiones y pruebas que ocurren después
  - Es una actividad llevada a cabo por los ingenieros de SW
- Proporcionar un enfoque basado en matemáticas para programar el modelado y la capacidad de verificar que el modelo del productos SW es correcto
  - Reducir el número de errores (bugs) mientras el SW se diseña y construye

Introducción (2/2)

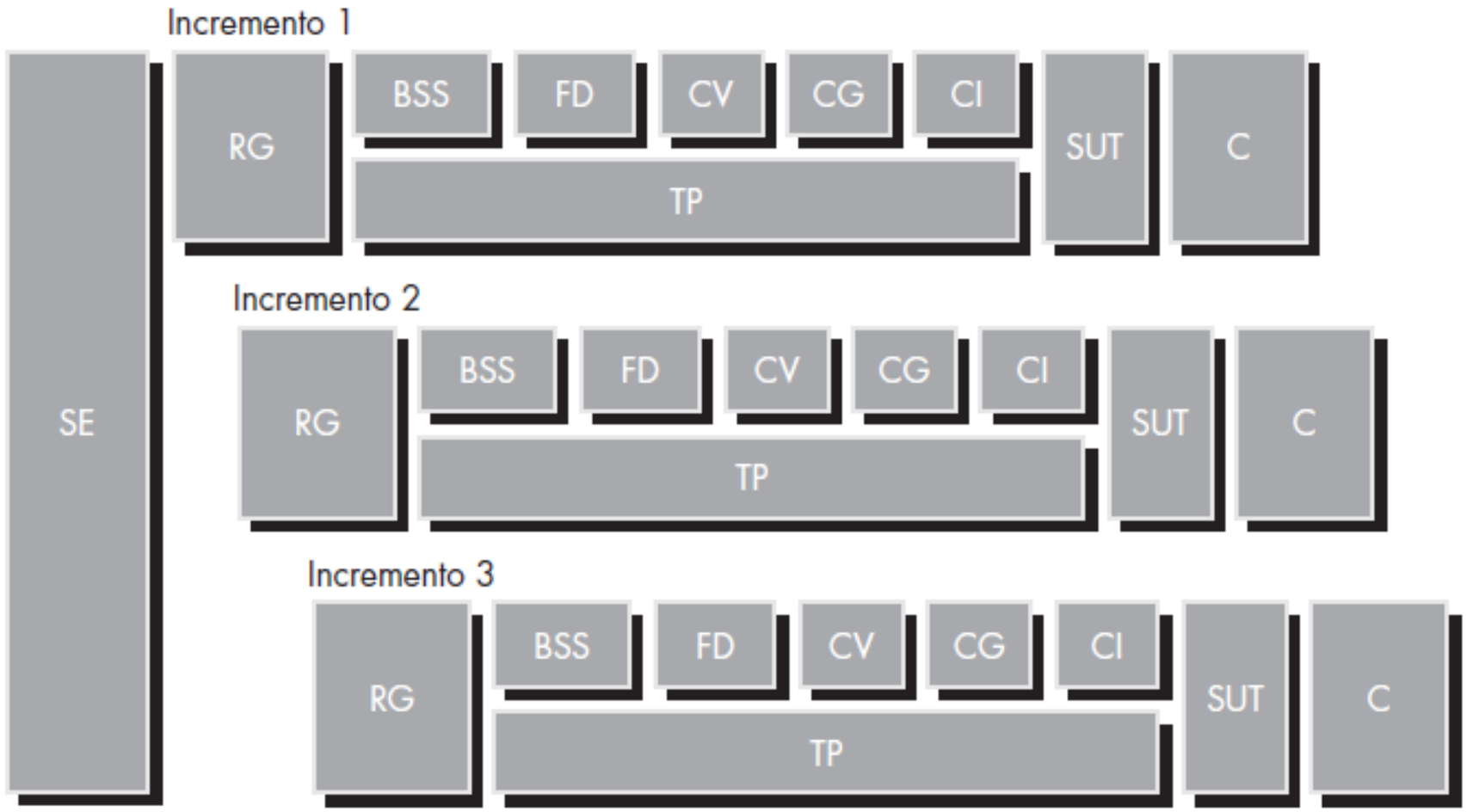
- Los modelos de requerimientos y de diseño se crean usando notación especializada que es susceptible de verificación matemática
  - Prueba de exactitud formal se aplica al modelo de requerimientos
  - Prueba de uso estadístico ejercita los escenarios de uso para garantizar que se descubren y corrigen los errores en la funcionalidad del usuario.
- ¿Cuál es el producto final?
  - Modelo formal especializado de requerimientos.
  - Se registran los resultados de las pruebas de exactitud y de uso estadístico

Estrategia de cuarto limpio (1/7)

- Enfoque que enfatiza la necesidad de construir con exactitud el software conforme este se desarrolla
  - Su modelo de proceso incorpora la certificación de calidad estadística de los incrementos de código conforme se acumulan en un sistema
  - Enfatiza la necesidad de probar la “exactitud”
- Versión especializada del modelo de software incremental de los métodos ágiles
  - Pequeños equipos de software independientes desarrollan “una tubería de incrementos de software”
  - Conforme cada incremento se certifica, se integra en el todo

Estrategia de cuarto limpio (2/7)

- Modelo de proceso de cuarto limpio:



SE — ingeniería de sistema
RG — recopilación de requerimientos
BSS — especificación de estructura de caja
FD — diseño formal
CV — verificación de exactitud
CG — generación de código
CI — inspección de código
SUT — prueba de uso estadística
C — certificación
TP — planeación de prueba

Estrategia de cuarto limpio (3/7)

- Dentro de la tubería de incrementos de cuarto limpio, ocurren las siguientes tareas:
  - **Planeación del incremento (SE).** Se desarrolla un plan de proyecto que adopte la estrategia incremental. Se crea la funcionalidad de cada incremento, su tamaño proyectado y un calendario de desarrollo de cuarto limpio. Debe tenerse especial cuidado para garantizar que los incrementos certificados se integraran en forma oportuna.
  - **Recopilación de requerimientos (RG).** Se desarrolla una descripción mas detallada de los requerimientos del cliente (para cada incremento)

Estrategia de cuarto limpio (4/7)

- **Especificación de estructura de caja (BSS).** Se usa un método de especificación que utilice unas *estructuras de caja* para describir la especificación funcional. Las estructuras de caja "se aíslan, y separan la definición creativa de comportamiento, datos y procedimientos en cada nivel de refinamiento".
- **Diseño formal (FD).** Al usar el enfoque de estructura de cajas, el diseño de cuarto limpio es una extensión de la especificación (requerimientos).
  - Las especificaciones (llamadas cajas negras) se refinan iterativamente (dentro de un incremento) para convertirse en análogas de los diseños arquitectónicos y en el nivel de componente (llamados cajas de estado y cajas claras, respectivamente).

Estrategia de cuarto limpio (5/7)

- **Verificación de exactitud (CV).** El equipo de cuarto limpio realiza una serie de rigurosas actividades de verificación de exactitud sobre el diseño y, luego, el código.
  - La verificación comienza con la estructura de caja (especificación) de nivel mas alto y avanza hacia el detalle y el código de diseño.
  - El primer nivel de la verificación de exactitud ocurre al aplicar un conjunto de “preguntas de exactitud”. Si esto no demuestra que la especificación es correcta, se usan métodos mas formales (matemáticos) para la verificación.

Estrategia de cuarto limpio (6/7)

- **Generación, inspección y verificación de código (CG, CI).** Las especificaciones de estructura de caja, representadas en un lenguaje especializado, se traducen al lenguaje de programación adecuado.
  - Las revisiones técnicas se usan entonces para asegurar la conformidad semántica del código y las estructuras de código, así como la exactitud sintáctica del código.
  - Luego se realiza la verificación de exactitud para el código fuente.
- **Planeación de prueba estadística (TP).** Se analiza el uso proyectado del software y se planea y diseña una suite de casos de prueba que ejercitan una “distribución de probabilidad” de uso.
  - Esta actividad de cuarto limpio se realiza en paralelo con la especificación, la verificación y la generación de código.

Estrategia de cuarto limpio (7/7)

- **Prueba de uso estadístico (SUT).** La prueba exhaustiva del software es imposible siempre es necesario diseñar un numero finito de casos de prueba.
  - Las técnicas de uso estadístico ejecutan una serie de pruebas derivadas de una muestra estadística (la distribución de probabilidad anotada anteriormente) de todas las posibles ejecuciones de programa efectuadas por todos los usuarios de una población objetivo
- **Certificación (C).** Una vez completadas la verificación, inspección y prueba de uso (y todos los errores corregidos), el incremento se certifica como listo para su integración

Especificación Funcional (1/5)

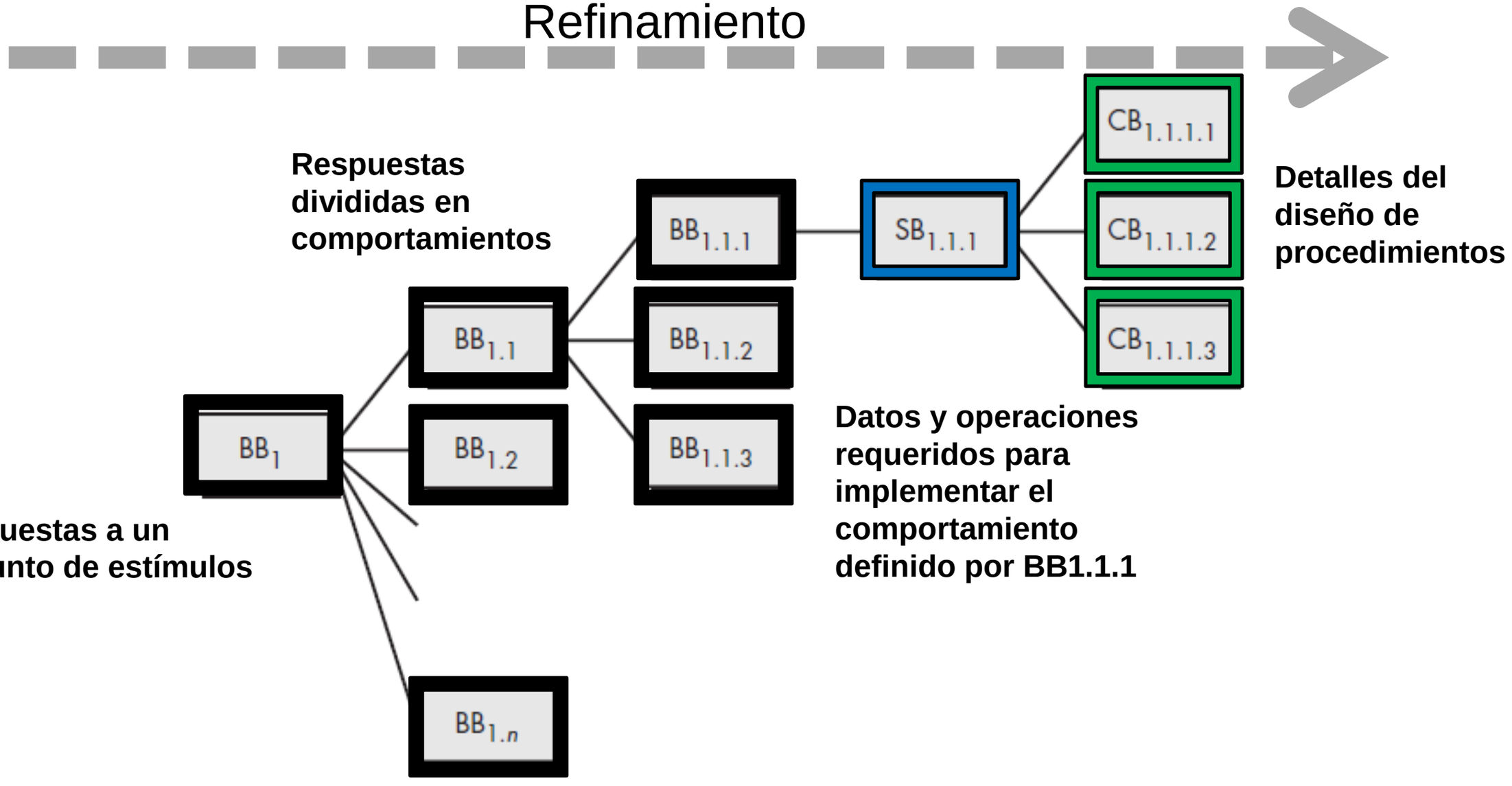
- El enfoque de modelado en la ingeniería del software de cuarto limpio usa un método llamado *especificación de estructura de caja*
  - Una “caja” encapsula el sistema (o algún aspecto del mismo) en algún nivel de detalle
  - A través de un proceso de elaboración o refinamiento por pasos, las cajas se refinan en una jerarquía donde cada caja tiene transparencia referencial (la funcionalidad encapsulada no depende de ninguna otra caja)

Especificación Funcional (2/5)

- Tipos de caja:
  - **Caja negra.** Especifica el comportamiento de un sistema o de una parte de un sistema. El sistema (o su parte) responde a estímulos específicos (eventos) al aplicar un conjunto de reglas de transición que mapean el estimulo en una respuesta
  - **Caja de estado.** Encapsula los datos y operaciones de estado en una forma análoga a los objetos. En esta visión de especificación, se representan las **entradas (estímulos)** y **salidas (respuestas)** a la caja de estado. La caja de estado también representa la "historia de estímulos" de la caja negra, es decir, los datos encapsulados en la caja de estado que deben conservarse entre las transiciones implicadas
  - **Caja clara.** Las funciones de transición que se implican mediante la caja de estado se definen en la caja clara. Contiene el diseño de procedimientos para la caja de estado

Especificación Funcional (3/5)

- Esquema de refinamiento de estructura de cajas



Caja negra **Caja de estado** **Caja clara**

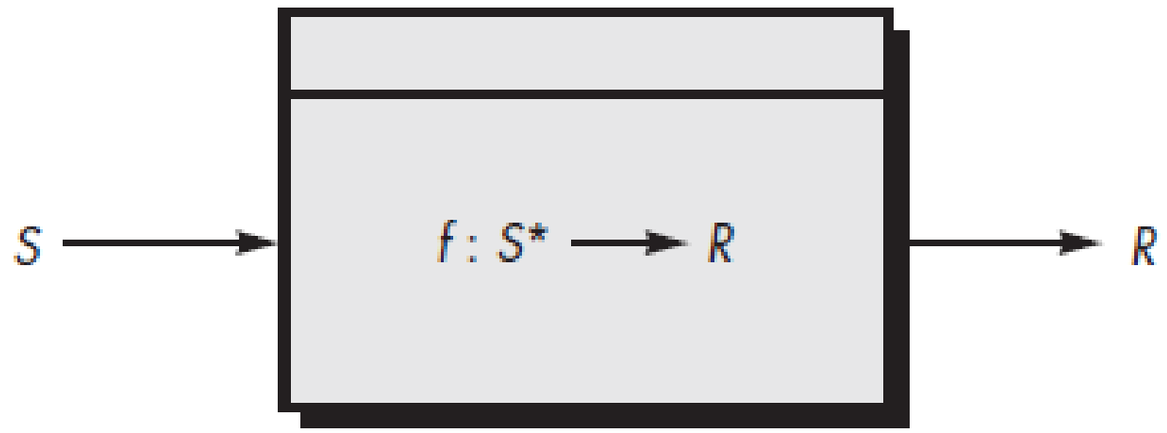
Especificación Funcional (4/5)

- Conforme ocurre cada uno de estos pasos de refinamiento, también ocurre la verificación de la exactitud.
  - Las especificaciones de caja de estado se verifican para asegurar que cada una se conforma de acuerdo con el comportamiento definido por la especificación de la caja negra padre.
  - Las especificaciones de caja clara se verifican contra la caja de estado padre.

Especificación Funcional (5/5)

Especificación de caja negra

- Una especificación de *caja negra* describe una abstracción, estímulos y respuesta, usando la siguiente notación:



- La función *f* se aplica a una secuencia *S** de entradas (estímulos) *S* y las transforma en una salida (respuesta) *R*
  - Para componentes de software simples, *f* puede ser una función matemática, pero, en general, *f* se describe usando lenguaje natural (o un lenguaje de especificación formal).
  - Pueden haber jerarquías de uso: cajas de nivel inferior heredan las propiedades de las cajas superiores en la jerarquía de cajas

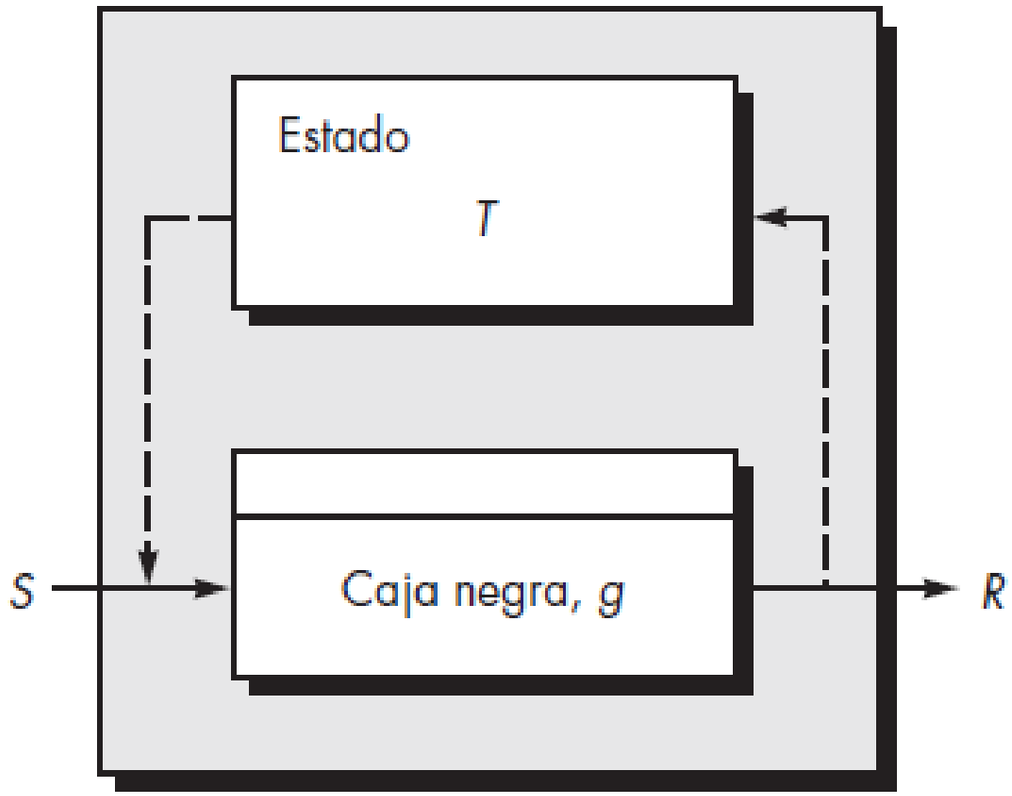
Especificación Funcional (5/5)

Especificación de caja de estado (1/2)

- "Simple generalización de una maquina de estado"
- Un estado es un modo observable de comportamiento del sistema
  - Durante el procesamiento, un sistema responde a los eventos (estímulos), haciendo una transición desde el estado actual hasta algún estado nuevo.
  - Conforme se realiza la transición, puede ocurrir una acción.
- La caja de estado usa una abstracción de datos para determinar la transición al siguiente estado y la acción (respuesta) que ocurrirá como consecuencia de la transición.

Especificación Funcional (5/5)

Especificación de caja de estado (2/2)

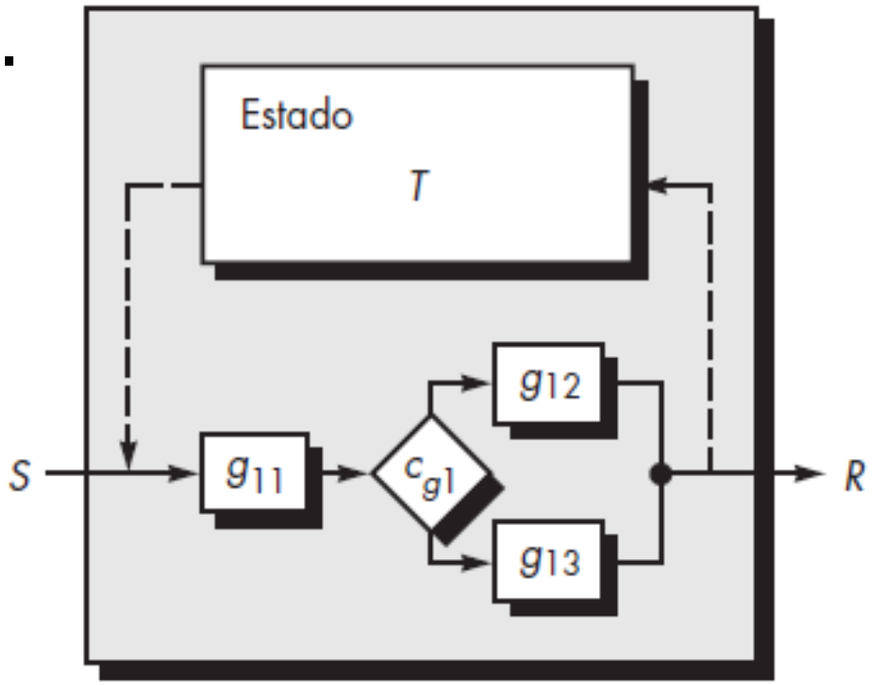


- La caja de estado incorpora una caja negra *g*
- El estimulo *S* que es entrada a la caja negra llega desde alguna fuente externa y desde un conjunto de estados internos del sistema *T*

Especificación Funcional (5/5)

Especificación de caja clara

- Se basa en el diseño procedural y con la programación estructurada
- La subfunción *g* dentro de la caja de estado se sustituye con los constructos de programación estructurada que implementan *g*.



La caja negra *g* se sustituye por un constructo secuencia que incorpora un condicional. Este, a su vez, puede refinarse en las cajas claras de nivel inferior conforme avanza el refinamiento por pasos

Diseño de Cuarto Limpio (1/2)

- Técnica de diseño formal basada en la programación estructurada
  - Las funciones de procesamiento básico se refinan usando:
    1) Expansión en pasos de funciones matemáticas en estructuras de conectivos lógicos (if-then-else)
    2) Subfunciones, donde la expansión se realiza hasta que todas las subfunciones identificadas puedan enunciarse directamente en el lenguaje de programación utilizado para la implementación
  - Los datos de programa se encapsulan como un conjunto de abstracciones que son atendidas por subfunciones

Diseño de Cuarto Limpio (2/2)

Refinamiento de diseño (1/2)

- Cada especificación de caja clara representa el diseño de un procedimiento (subfunción) requerido para lograr una transición de caja de estado
  - Dentro de la caja clara se usan constructos de programación estructurada y refinamiento por pasos para representar detalles procedurales
    - Una función de programa *f* se refina en una secuencia de subfunciones *g* y *h*.
    - Se refinan en constructos condicionales (por ejemplo, if-then-else y do-while).
    - El refinamiento continua hasta que hay suficiente detalle procedural para crear el componente en cuestión

Diseño de Cuarto Limpio (2/2)

Refinamiento de diseño (2/2)

- El equipo de cuarto limpio realiza una **verificación formal de exactitud**.
  - A los constructos de programación estructurada se une un conjunto de condiciones de exactitud genéricas
    - Si una función ***f*** se expande en una **secuencia *g* y *h***, la condición de exactitud para toda entrada a *f* es:

      ¿*g* seguida de *h* hace *f*? → **1 Condición a verificar**
    - Cuando una función ***p*** se refina en una condicional de la forma **"if *c* then *q*, else *r*"**, la condición de exactitud para toda entrada a *p* es:

      Siempre que la condición *c* es verdadera, ¿*q* hace *p*?; y siempre que *c* es falsa, ¿*r* hace *p*? → **2 Condiciones a verificar**
    - La función ***m*** se refina como un **ciclo**, las condiciones de exactitud para toda entrada a *m* son:

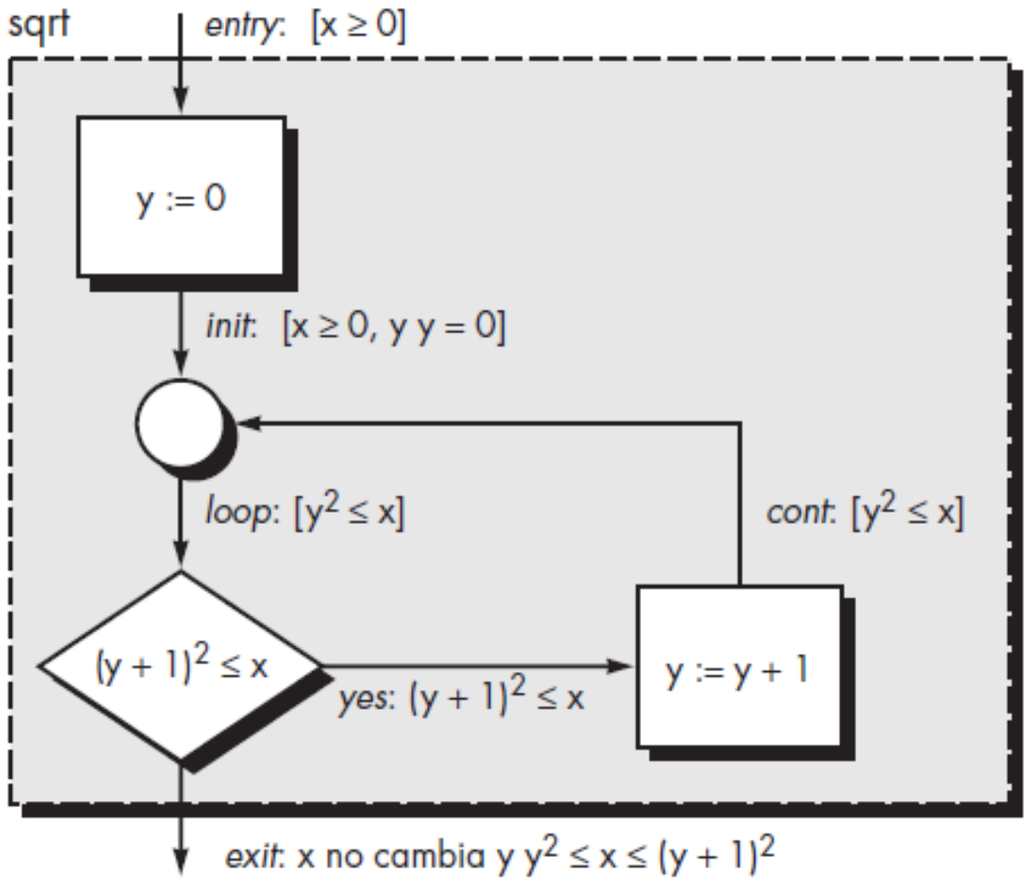
      ¿Esta garantizada la finalización? → 1 condición + …

      Siempre que *c* es verdadera, ¿*n* seguida por *m* hace *m*?; y siempre que *c* es falsa, ¿saltar el ciclo todavía hace *m*? → … + 2 condiciones = **3 conds**

Diseño de Cuarto Limpio (2/2)

Verificación de diseño (1/3)

Diseño procedural para el cálculo de la parte entera de una raíz cuadrada



- Para probar que el diseño es correcto, es necesario probar que las condiciones *init*, *loop*, *cont*, *yes* y *exit,* son verdaderas en todos los casos:

  (1 condición para las secuencias, 2 condiciones para if-then-else y 3 para ciclos)

Diseño de Cuarto Limpio (2/2)

Verificación de diseño (2/3)

- La condición *init* demanda que [$x \geq 0$ y $y = 0$]:
  - Como no tiene sentido un valor negativo para la raíz cuadrada, se satisface $x \geq 0$
  - En el diagrama de flujo, el enunciado inmediatamente anterior a la condición init establece y =0 → OK.

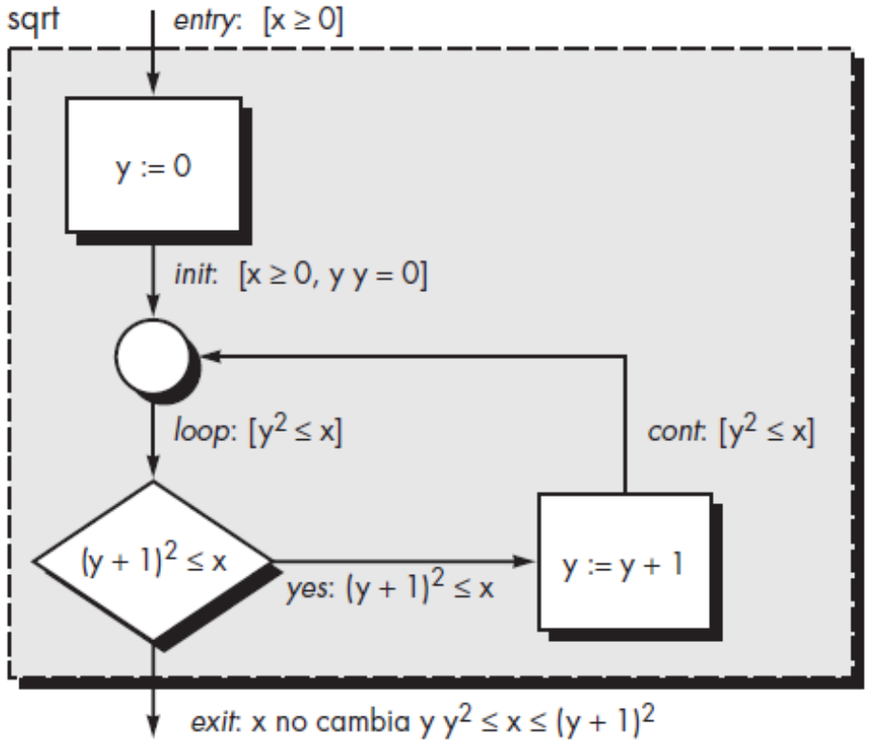
    ➔ init es verdadera

- Para la condición loop demanda que [y^2 ≤ x]:
  - Directamente de *init* → OK
  - Por medio del flujo de control: (*loop* = condición *cont*) → OK

    ➔ loop es verdadera

- Para la condición cont demanda que [y^2 ≤ x]:
  - Se encuentra cuando el valor de *y* se aumenta en 1
  - La ruta de flujo de control que conduce a *cont* puede invocarse solo si la condición *yes* también es verdadera

    Si (*y* + 1)^2 ≤ *x* → *y*^2 ≤ *x*. ➔ *cont* es verdadera



Diseño de Cuarto Limpio (2/2)

Verificación de diseño (3/3)

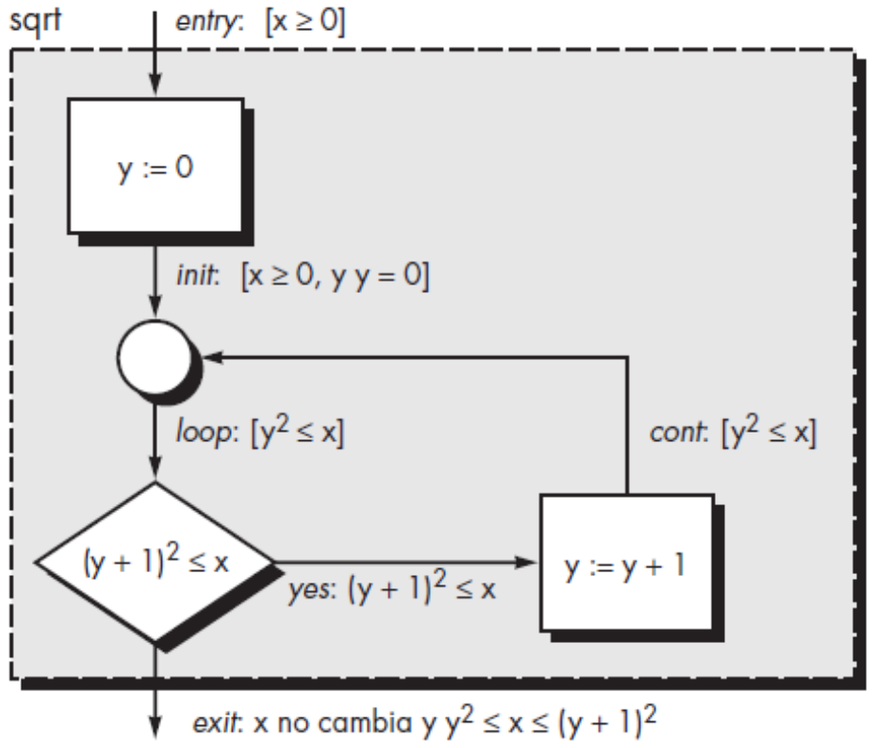
- La condición *yes* demanda que [(*y* + 1)^2 ≤ *x*]:
  - Se prueba en la lógica condicional y cuando se recorre el arco de izquierda a derecha ➔ condición yes es verdadera
- La condición *exit* demanda primero que *x* permanezca invariable
  - Como no hay instrucciones/funciones que modifiquen X

➔ Primera condición es verdadera

- Segunda condición del *exit* : [*y*^2 ≤ *x* ≤ (*y* + 1)^2]
  - Puesto que la prueba condicional (y + 1)^2 ≤ x debe fallar para alcanzar la condición exit → (y + 1)^2 ≤ x.
  - La condición *loop* debe incluso ser verdadera → *y*^2 ≤ *x*

➔ (y + 1)^2 > x & *y*^2 ≤ *x* ➔ la condición exit es verdadera

- Adicionalmente, debe asegurarse de que el ciclo termina. Un examen de la condición *loop* indica que, dado que *y* se incrementa y *x* ≥ 0, el ciclo finalmente debe terminar



Pruebas de Cuarto Limpio (1/2)

- La estrategia y tácticas de las pruebas de cuarto limpio son fundamentalmente diferentes a las de los enfoques de prueba convencionales.
  - Los métodos convencionales derivan un conjunto de casos de prueba para descubrir errores de diseño y codificación.
- La meta de la prueba de cuarto limpio es validar los requerimientos de software al demostrar que una muestra estadística de casos de uso se ejecuta exitosamente

Pruebas de Cuarto Limpio (2/2)

Pruebas de uso estadístico (1/4)

- El comportamiento del programa visible al usuario se activa con entradas y eventos que con frecuencia son producidos por el usuario
- En los sistemas complejos, el posible espectro de entrada y eventos (casos de uso) puede ser extremadamente amplio
  - **Pruebas de uso estadístico**: subconjunto de casos de uso verificara de manera adecuada el comportamiento del programa

Pruebas de Cuarto Limpio (2/2)

Pruebas de uso estadístico (2/4)

- Examinar el software de la forma en la que los usuarios pretenden usarlo
- Los equipos de prueba de cuarto limpio (equipos de certificación) deben determinar una distribución de probabilidad de uso para el software.
- La especificación (caja negra) para cada incremento del software se analiza a fin de definir un conjunto de estímulos (entradas o eventos) que hacen que el software cambie su comportamiento

Pruebas de Cuarto Limpio (2/2)

Pruebas de uso estadístico (3/4)

- En la creación de escenarios de uso y en una comprensión general del dominio de aplicación, a cada estímulo se le asigna una probabilidad de uso con base en entrevistas con usuarios potenciales
- Para cada conjunto de estímulos se generan casos de prueba de acuerdo con la distribución de probabilidad de uso

Estímulo del programa	Probabilidad	Intervalo
Armar/desarmar (AD)	50%	1–49
Establecer zona (EZ)	15%	50–63
Consulta (C)	15%	64–78
Prueba (P)	15%	79–94
Alarma de pánico	5%	95–99

Pruebas de Cuarto Limpio (2/2)

Pruebas de uso estadístico (4/4)

- Se generar una secuencia de casos de prueba de uso que se ajusten a la distribución de probabilidad de uso
  - Números aleatorios entre 1 y 99

13-94-22-24-45-56 → AD–T–AD–AD–AD–ZS

81-19-31-69-45-9 → T–AD–AD–AD–Q–AD–AD

38-21-52-84-86-4 → AD–AD–ZS–T–T–AD

Estímulo del programa	Probabilidad	Intervalo
Armar/desarmar (AD)	50%	1–49
Establecer zona (EZ)	15%	50–63
Consulta (C)	15%	64–78
Prueba (P)	15%	79–94
Alarma de pánico	5%	95–99

- El equipo de prueba los ejecuta, y verifica el comportamiento del software contrastándolo con la especificación para el sistema.
  - La temporización de las pruebas se registra de modo que puedan determinarse los intervalos de tiempo. Al usar intervalos de tiempo, el equipo de certificación puede calcular el tiempo medio hasta el fallo (TMHF).
  - Si una larga secuencia de pruebas se realiza sin fallas, el TMHF es bajo y la confiabilidad del software puede suponerse alta.

Pruebas de Cuarto Limpio (2/2)

Certificación (1/2)

- Dentro del contexto del enfoque de ingeniería del software de cuarto limpio, la *certificación* implica que la confiabilidad (medida por el tiempo medio hasta el fallo o TMHF) puede especificarse para cada componente
- El enfoque de certificación involucra cinco pasos:

1. Se crean escenarios de uso
2. Se especifica un perfil de uso
3. Se generan casos de prueba a partir del perfil
4. Las pruebas se ejecutan y los datos de fallo se registran y analizan
5. …. (siguiente diapositiva)

Pruebas de Cuarto Limpio (2/2)

Certificación (2/2)

5. Se calcula la confiabilidad y se certifica

- Creación de tres modelos para ejecutar el paso 5.
  - **Modelo de muestreo.** La prueba de software ejecuta *m* casos de prueba aleatorios (*m* ha de ser lo suficientemente grande como para dar un valor de confiabilidad válido) y se certifica si no ocurren fallos o un numero especifico de ellos.
  - **Modelo de componente.** Se certifica un sistema compuesto de *n* componentes.
    - Probabilidad de que el componente *i* falle
  - **Modelo de certificación.** La confiabilidad global del sistema se proyecta y se certifica
- El equipo de certificación tiene la información requerida para entregar el software que tenga un TMHF certificado