GRADO

GUÍA DE ESTUDIO DE LA ASIGNATURA

PRUEBAS DE SOFTWARE

2ª PARTE | PLAN DE TRABAJO Y ORIENTACIONES PARA SU DESARROLLO

| Equipo docente de Pruebas de Software

GRADO EN INGENIERÍA EN INFORMÁTICA

1.- PLAN DE TRABAJO

El equipo docente utilizará el curso virtual de la asignatura para la comunicación con los alumnos. Es muy recomendable que el alumno acceda periódicamente a este portal para estar informado de cualquier novedad que se pudiera producir.

La asignatura de Pruebas de Software requiere un estudio sistemático y continuado a lo largo del cuatrimestre, dado que se han de asimilar conceptos teóricos y prácticos simultáneamente. Para facilitar su superación es conveniente planificar las etapas de estudio desde el principio, teniendo en cuenta los plazos de entrega y dedicando semanalmente el tiempo necesario, ya que es difícil asimilar la asignatura si se deja el trabajo para el final del cuatrimestre. El objetivo de esta asignatura es que el estudiante adquiera los conocimientos y competencias reflejados en la guía general de la asignatura en los 6 ETCS (créditos europeos) que tiene asignados. Un crédito equivale a 25 horas, lo que implica unas 150 horas de estudio y trabajo a lo largo de las 12 semanas disponibles para el curso.

Los créditos asignados están en consonancia con los contenidos, distribuidos en tres unidades didácticas. Además, en el curso virtual se publicará el enunciado de dos Prácticas de Evaluación Continua (PECs). Las PECs son voluntarias y su realización podrá incrementar la nota del examen presencial hasta un máximo de un punto.

El cuadro siguiente muestra el cronograma que marca unas pautas adecuadas para que el alumno medio alcance los objetivos al final del curso. Este cronograma incluye los contenidos que recomendamos estudiar cada semana y sus respectivas referencias bibliográficas.

CONTENIDOS	PLAN DE ACTIVIDADES
UNIDAD DIDÁCTICA I: Fundamentos de las pruebas de software *3 Semanas*	
1ª Semana	
a) Imposibilidad de las pruebas exhaustivas b) Error, defecto o falta y fallo c) Caso de prueba d) Objetivo de las pruebas	1. Estudiar el capítulo 1 del libro de texto base
2ª Semana	
a) Pruebas de caja negra b) Pruebas estructurales o de caja blanca c) Pruebas unitarias d) Pruebas de integración e) Pruebas de sistema	1. Estudiar el capítulo 2 del libro de texto base
3ª Semana	
a) Criterios de cobertura b) Utilidad de los criterios de cobertura c) Un posible modelo de trabajo d) Criterios de cobertura para código fuente e) Criterios de cobertura para máquinas de estado f) Limitaciones de los criterios de cobertura	1. Estudiar el capítulo 3 del libro de texto base 2. Leer el artículo "Faults of Omission" de Brian Marick, publicado en "Software Testing and Quality Engineering Magazine", Jan. 2000. El artículo está disponible en el curso virtual y en http://www.exampler.com/testing-com/writings/omissions.html 3. Leer el artículo "How to Misuse Code Coverage" de Brian Marick, disponible en el curso virtual y en http://www.exampler.com/testing-com/writings/coverage.pdf

CONTENIDOS	PLAN DE ACTIVIDADES
UNIDAD DIDÁCTICA II: Valores de prueba *3 Semanas*	
1ª Semana	
a) Clases o particiones de equivalencia b) Valores límite c) Conjetura de errores d) Aplicación de las técnicas al conjunto de datos de salida e) Criterios de cobertura para valores de prueba	1. Estudiar el capítulo 4 del libro de texto base
2ª Semana	
a) Identificación sistemática de valores de prueba b) Dimensiones principales y secundarias de variables de prueba	1. Estudiar el documento "Ejemplo de PEC1: Identificación de los valores de prueba" anexo a esta guía
3ª Semana	
a) Aplicación práctica de lo estudiado	1. Realizar la PEC 1

CONTENIDOS	PLAN DE ACTIVIDADES
UNIDAD DIDÁCTICA III: Combinación de valores de prueba *5 Semanas*	
1ª Semana	
a) Estructura de un caso de prueba b) El oráculo c) Estrategias de combinación	1. Estudiar el capítulo 5 del libro de texto base
2ª Semana	
a) Justificación empírica de los métodos combinatorios b) Métodos combinatorios para la prueba de configuraciones c) Métodos combinatorios para la prueba de parámetros de entrada d) Medición de la cobertura combinatoria e) Métodos combinatorios frente a métodos aleatorios	1. Estudiar los capítulos 1-4, 6, 7; y los apéndices A-C del documento "Practical Combinatorial Testing", de R. Kuhn et al., disponible en el curso virtual y en http://csrc.nist.gov/groups/SNS/acts/documents/SP800-142-101006.pdf
3ª Semana	
a) Encontrar un "covering array" mínimo es un problema NP-completo b) Combinaciones con valores mixtos de *t* c) Supresión de combinaciones indeseables mediante la inclusión de restricciones d) Reutilización de combinaciones de prueba a medida que se aumenta *t*	1. Estudiar el artículo "Pairwise Testing in the Real World: Practical Extensions to Test-Case Scenarios" de J. Czerwonka, Microsoft Corporation, Feb. 2008. El artículo está disponible en el curso virtual y en https://msdn.microsoft.com/en-us/library/cc150619.aspx

4ª Semana	
a) Ejemplo práctico de testing combinatorio b) Herramientas para testing combinatorio	1. Estudiar el documento "Ejemplo de PEC2: Combinación de valores de prueba" anexo a esta guía 2. Codificar y ejecutar el ejemplo tratado en el documento "Ejemplo de PEC2: Combinación de valores de prueba" con las herramientas ACTS y PIC. • ACTS es gratuita. Puede obtenerse (i) en el curso virtual o (ii) solicitando una copia a R. Kuhn (autor de ACTS) en http://csrc.nist.gov/groups/SNS/acts/index.html • PICT es gratuita. Puede obtenerse (i) en el curso virtual o (ii) solicitando una copia a R. Kuhn (autor de ACTS) en http://download.microsoft.com/download/f/5/5/f55484df-8494-48fa-8dbd-8c6f76cc014b/pict33.msi
5ª Semana	
a) Aplicación práctica de lo estudiado	1. Realizar la PEC 2

Se recomienda emplear una semana adicional en repasar lo estudiado en el curso.

2.- ORIENTACIONES PARA EL ESTUDIO DE LOS CONTENIDOS

2.1 Contextualización

En Pruebas de Software se estudiarán las técnicas básicas para comprobar el correcto funcionamiento de aplicaciones informáticas.

En las siguientes asignaturas del Grado en Ingeniería Informática se estudian los fundamentos del desarrollo de software:

- Primer curso: (i) Fundamentos de Programación, (ii) Estrategias de Programación y Estructuras de Datos, y (iii) Programación Orientada a Objetos
- Segundo curso: (i) Programación y Estructuras de Datos Avanzadas, y (ii) Introducción a la Ingeniería de Software
- Tercer curso: Diseño del Software

Pruebas de Software complementa a las citadas asignaturas, aportando las técnicas necesarias para detectar errores en sistemas software.

2.2 Requisitos previos

Se aconseja tener los conocimientos básicos de programación y algoritmia que se estudian en las asignaturas de (i) Fundamentos de Programación y (ii) Estrategias de Programación y Estructuras de Datos del primer curso del Grado en Ingeniería Informática.

2.3 Descripción detallada de los contenidos

A continuación se hace una enumeración de los contenidos de cada una de las Unidades Didácticas:

UNIDAD DIDÁCTICA I: Fundamentos de las pruebas de software

1.1. Conceptos fundamentales

1.1.1. Imposibilidad de las pruebas exhaustivas
1.1.2. Error, defecto o falta y fallo
1.1.3. Caso de prueba
1.1.4. Objetivo de las pruebas

1.2. Niveles de prueba

1.2.1. Pruebas de caja negra
1.2.2. Pruebas estructurales o de caja blanca
1.2.3. Pruebas unitarias
1.2.4. Pruebas de integración
1.2.5. Pruebas de sistema

1.3. Criterios de cobertura para artefactos software

1.3.1.Criterios de cobertura
1.3.2. Utilidad de los criterios de cobertura
1.3.3. Un posible modelo de trabajo
1.3.4. Criterios de cobertura para código fuente
1.3.5. Criterios de cobertura para máquinas de estado
1.3.6. Limitaciones de los criterios de cobertura

UNIDAD DIDÁCTICA II: Valores de prueba

2.1. Introducción a los valores de prueba

2.1.1. Clases o particiones de equivalencia
2.1.2. Valores límite
2.1.3. Conjetura de errores
2.1.4. Aplicación de las técnicas al conjunto de datos de salida
2.1.5. Criterios de cobertura para valores de prueba

2.2. Determinación de los valores de prueba

2.1.1. Identificación sistemática de valores de prueba
2.1.2. Dimensiones principales y secundarias de variables de prueba

UNIDAD DIDÁCTICA III: Combinación de valores de prueba

3.1. Introducción al testing combinatorio

3.1.1. Estructura de un caso de prueba
3.1.2. El oráculo
3.1.3. Estrategias de combinación

3.2. Testing combinatorio desde un enfoque práctico

3.2.1. Estructura de un caso de prueba
3.2.2. El oráculo
3.2.3. Estrategias de combinación

3.3. Limitaciones y funcionalidades básicas de una herramientas de testing combinatorio

3.3.1. Encontrar un "covering array" mínimo es un problema NP-completo
3.3.2. Combinaciones con valores mixtos de *t*
3.3.3. Supresión de combinaciones indeseables mediante la inclusión de restricciones
3.3.4. Reutilización de combinaciones de prueba a medida que se aumenta *t*

2.4 Prácticas de Evaluación Continua

Los alumnos podrán realizar dos Prácticas de Evaluación Continua (PECs). Las PECs son voluntarias y su realización podrá incrementar la nota del examen presencial hasta un máximo de un punto.

Como anexo a esta guía se incluyen dos PECs de ejemplo completamente resueltas.

2.5 Resultados del aprendizaje asociados a los contenidos

En la asignatura se aprenderán:

- Dos de las técnicas más populares para la simplificación de un espacio de prueba potencialmente infinito: la partición del espacio en clases de equivalencia y la identificación de valores límite para dichas clases.

- Los fundamentos del testing combinatorio, que facilita la creación de juegos de prueba que exploran la interacción entre distintos valores de prueba.

- Los fundamentos algorítmicos del testing combinatorio y cómo utilizar herramientas que dan soporte automático de este tipo de pruebas.

2.6 Bibliografía básica

El libro de texto base de la asignatura es:

Autores: Macario Polo Usaola, Beatriz Pérez Lamancha, Pedro Reales Mateo
Título: Técnicas combinatorias y de mutación para testing de sistemas software
Editorial: Ra-Ma
Año de publicación: 2012

Además, los alumnos deberán estudiar los siguientes documentos gratuitos:

- "Faults of Omission" de Brian Marick, artículo disponible en el curso virtual y en http://www.exampler.com/testing-com/writings/omissions.html

- "How to Misuse Code Coverage" de Brian Marick, artículo disponible en el curso virtual y en http://www.exampler.com/testing-com/writings/coverage.pdf

- "Practical Combinatorial Testing" de R. Kuhn et al., documento disponible en el curso virtual y en http://csrc.nist.gov/groups/SNS/acts/documents/SP800-142-101006.pdf

- "Pairwise Testing in the Real World: Practical Extensions to Test-Case Scenarios" de J. Czerwonka, Microsoft Corporation, Feb. 2008. Este artículo está disponible en el curso virtual y en https://msdn.microsoft.com/en-us/library/cc150619.aspx

2.7 Bibliografía complementaria

El siguiente libro está enteramente dedicado a los contenidos de la Unidad Didáctica 2:

Cem Kaner, Sowmya Padmanabhan, and Douglas Hoffman. The Domain Testing Workbook. Context Driven Press, 2013.

Los siguientes libros cubren de manera introductoria las Unidades Didácticas 1 y 2:

- Paul C. Jorgensen. Software Testing: A Craftsman's Approach, 4th Edition. Auerbach Publications, 2013.

- Glenford J. Myers, Corey Sandler, and Tom Badgett. *The Art of Software Testing, 3rd Edition*. Wiley, 2011.

3.- ORIENTACIONES PARA LA REALIZACIÓN DEL PLAN DE ACTIVIDADES

La metodología seguida para el aprendizaje de esta asignatura es la propia de una universidad a distancia, que se caracteriza por el empleo conjunto de medios impresos, audiovisuales y de las nuevas tecnologías. Los materiales docentes específicos, las comunidades virtuales de aprendizaje, la asistencia presencial a los estudiantes a través de los profesores tutores de los Centros Asociados y el uso de los diversos sistemas de comunicación (teléfono, videoconferencia, radio, televisión, correo electrónico, etc.) son los medios con que cuenta la UNED para la enseñanza a distancia y todos ellos son utilizados en esta asignatura.

3.1 Medios y recursos

Exceptuando el libro de texto base, el resto del material bibliográfico básico, así como las aplicaciones ACTS y PICT, estarán disponibles en el curso virtual de la asignatura.

3.2 Evaluación

La evaluación de la asignatura se realizará mediante pruebas presenciales y una evaluación continua basada en la realización de dos Prácticas de Evaluación Continua (PECs).

3.2.1 Evaluación continua: Prácticas

Las PECs son voluntarias y su realización podrá incrementar la nota de la prueba presencial hasta un máximo de UN PUNTO. Es necesaria la realización correcta de las DOS PECs para optar al incremento del punto. La entrega y evaluación de las PECs sólo se podrá realizar antes de presentarse al examen de la convocatoria de junio con la fecha tope que se publicará en el curso virtual de la asignatura. Si no se entregan las prácticas antes de dicha fecha se entiende que se está renunciando a la evaluación continua del curso.

3.2.2 Pruebas Presenciales: Exámenes

En el examen, el alumno deberá resolver un supuesto práctico aplicando los contenidos de la asignatura. Para realizar la prueba personal no se permitirá el uso de ningún material auxiliar. El examen se evaluará de 0 a 9. Por tanto, para obtener la máxima calificación en la asignatura (10) es imprescindible haber resuelto satisfactoriamente las dos PECS voluntarias.

4.- ANEXO: EJEMPLOS DE PEC

A continuación se incluyen dos PECs, completamente resueltas, similares a las que el alumno deberá resolver.

Pruebas de Software

Ejemplo de PEC-1: Identificación de los valores de prueba

Dpto. de Ingeniería de Software y Sistemas Informáticos

Rubén Heradio, rheradio@issi.uned.es

ETSI Informática, Universidad Nacional de Educación a Distancia

Índice

1. Introducción

Este documento incluye un ejemplo resuelto de Prueba de Evaluación Continua 1 (PEC-1) para la asignatura de Pruebas de Software impartida en la Universidad Nacional de Educación a Distancia.

Este tipo de PEC persigue los siguientes objetivos:

1. Que el alumno ejercite dos técnicas fundamentales en la prueba de software: la partición de un espacio de pruebas en clases de equivalencia y la determinación de valores límite para dichas clases. Pese a que conceptualmente estas técnicas pueden parecer sencillas, en la práctica su aplicación puede llegar a ser bastante compleja. Con el fin de facilitar la resolución progresiva de este tipo de problemas, se propone seguir el conjunto de pasos descritos en la Sección 3. Además, recomendamos la lectura del magnifíco libro de Kaner et al. [3], que está dedicado integramente a la utilización de estas técnicas.
2. Concienciar al alumno de la dificultad que supone probar a fondo una variable con dominio infinito. Para ello, la Sección 3 introduce e ilustra el concepto de *dimensiones secundarias* para variables de prueba.

2. Enunciado

En esta PEC, el *System Under Test* (SUT) será el formulario web de la Figura 1, cuya URL es:

`http://www.stikets.com/etiquetas/etiquetas-ropa/ropa-pequenas.html`

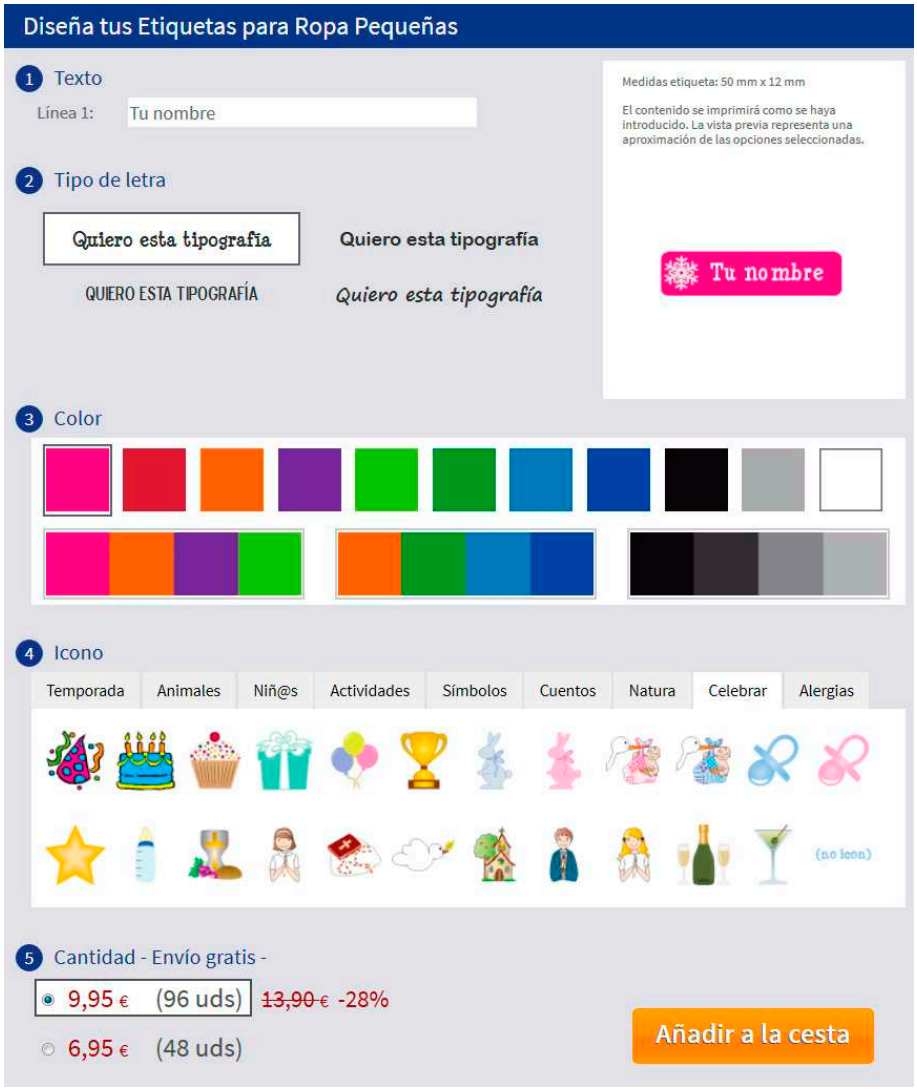

Figura 1: Formulario para la creación de etiquetas

Dicho formulario facilita la creación de etiquetas para ropa, que pueden configurarse según los siguientes cuatro parámetros:

1. *Texto de la etiqueta*
2. *Tipografía del texto*
3. *Color de la etiqueta*
4. *Iconos*. Opcionalmente, las etiquetas pueden contener un icono. Si lo hacen, el texto no puede sobrepasar los 25 caracteres. En cambio, cuando las etiquetas carecen de iconos podrán incluir hasta 30 caracteres.

Además, el formulario incluye un último parámetro para especificar la cantidad de etiquetas que el cliente desea comprar.

 El problema consiste en determinarproponer un conjunto de valores de prueba para el SUT, mediante la partición del espacio de prueba en clases de equivalencia y la determinación de valores límite para dichas clases.

 Se aconseja que, para familiarizarse con el problema, el alumno visite la URL y "juegue" con el formulario antes de intentar resolver la PEC.

 Para realizar esta PEC, se recomienda la consulta de las siguientes referencias bibliográficas:

1. El capítulo 4 del texto base de la asignatura [5].
2. Kaner et al. [3] es el texto más completo sobre partición de espacios de prueba en clases de equivalencia y el uso de valores límite.
3. Los libros [1] y [4] son clásicos de la prueba de software. El contenido de esta PEC se trata en los capítulos 5 y 6 de [1], y en el capítulo 4 de [4].

3. Solución

Para determinar de una manera sistemática las clases de equivalencia y sus respectivos valores de prueba, se propone seguir los siguientes pasos:

Paso 1 *Identificar qué variables interesa probar*. En nuestro caso, las variables se corresponden con los campos del formulario de la Figura 1:

- Texto de la etiqueta (T)
- Tipo de Letra (TL)
- Color de la etiqueta (C)
- Icono (I)
- Cantidad de Etiquetas (CE)

Paso 2 *Determinar el tipo de cada variable*:

- T: String
- TL: Enumerado
- C: Enumerado
- I: Enumerado
- CE: Enumerado

Paso 3 *Determinar las dimensiones principales y secundarias de cada variable*. Algunas variables, como las booleanas y los tipos enumerados, tienen un dominio finito. Por ejemplo, TL, C, I y CE son de tipo enumerado.

Sin embargo, en la mayoría de las ocasiones, tendremos que lidiar con variables cuyo dominio de valores será infinito. Por ejemplo, el texto T de las etiquetas es de tipo String. Para limitar los casos de prueba de estos tipos de variables, identificaremos las *dimensiones* a los largo de las cuales pueden variar sus valores. Por ejemplo, para T se podrían considerar las siguientes dimensiones:

- *Contenido de T*. Es decir, ¿cuál es comportamiento del SUT cuando varía el contenido del texto?. Por ejemplo, ¿que pasa si se incluyen acentos, ñ's, etc.?
- *Longitud de T*. Es decir, ¿cuál es comportamiento del SUT cuando varía la longitud del texto?. Por ejemplo, ¿que pasa cuando no introducimos ningún texto (longitud 0)? ¿y cuando excedemos la longitud máxima?

Una vez que hemos identificado las dimensiones de las variables con dominio infinito, pasaremos a determinar su *dimensión principal*. De acuerdo con Kaner et al. [3], la dimensión principal de una variable depende de cual sea el uso que hace el SUT de la variable. Para su determinación, deberíamos preguntarnos ¿qué aprende, consigue o controla el SUT con esta variable? El resto de las dimensionas se considerarán secundarias.

El objetivo de nuestro SUT es imprimir etiquetas. Cuando introducimos un texto, el SUT trata de formatearlo adecuadamente para que quepa en la etiqueta y se muestre de la manera más clara posible. Además, según el enunciado, el SUT ajusta el texto de manera distinta si la etiqueta contiene un icono (en cuyo caso, el texto puede contener hasta 25 caracteres) o si no lo incluye (en cuyo caso, el texto puede albergar hasta 30 caracteres). En conclusión, los tests deben orientarse fundamentalmete a estudiar como reacciona el SUT frente a distintos tamaños de T y, por tanto, consideraremos que la dimensión principal de T es su longitud (Longitud del Texto, LT).

Paso 4 *Particionar la dimensión principal de cada variable.* El objetivo de la partición en clases de equivalencia es evitar tests redundantes cuando tenemos un conjunto infinito (o muy grande) de valores posibles. La idea fundamental puede expresarse como: "si parece que el SUT procesa de la misma manera cierto conjunto de valores, debería bastar probar con cualquiera de dichos valores".

Como el dominio de TL, C, I y CE es muy reducido, probaremos con todos sus valores (dicho de otro modo, "consideraremos que cada valor es una clase de equivalencia").

El tamaño máximo permitido para el texto depende de si la etiqueta incluye un icono o no.

- Si la etiqueta contiene un icono, distinguiremos las siguientes clases:
  - $LT < 1$ (el texto no incluye caracteres)
  - $1 \leq \text{LT} \leq 25$
  - $LT > 25$ (el texto rebasa el límite)
- Si la etiqueta no incluye icono, distinguiremos las siguientes clases:
  - $LT < 1$ (el texto no incluye caracteres)
  - $1 \leq \text{LT} \leq 30$
  - $LT > 30$ (el texto rebasa el límite)

Paso 5 *Determinar si existe alguna relación entre los valores primarios de las variables.* Dado que la inclusión de iconos limita el máximo número de caracteres de las etiquetas, existen las siguientes relaciones entre I y LT:

- I = con icono $\Rightarrow LT \leq 25$
- I = sin icono $\Rightarrow LT \leq 30$

Paso 6 *Diseñar la tabla de clases de equivalencia y valores límite para la dimensión principal.* La Tabla 1 recopila las clases de equivalencia detectadas en el Paso 4 y propone valores de prueba para cada una de esas clases. La tabla sigue el formato propuesto por Kaner et al. [3], que utiliza una fila para cada valor de prueba, indicando a qué variable y clase de equivalencia corresponde. Dada la tendencia de los SUT a fallar en las fonteras de las clases, siempre que sea posible escogeremos como valores de prueba los *valores límite* de las clases. Por ejemplo, probaremos $1 \leq \text{LT} \leq 25$ con los valores 1 y 25.

Se recomienda que la tabla sea lo más legible posible. Por ello, es conveniente repetir valores de prueba para una misma variable cuando se correspondan a clases de equivalencia distintas. Posteriormente, y para evitar redundancias, utilizaremos un identificador único para cada valor. Por ejemplo, el valor $LT = 1$, con identificador $T1$, se utiliza en las clases $1 \leq \text{LT} \leq 25$ y $1 \leq \text{LT} \leq 30$ y , por tanto, se incluye dos veces en la Tabla 1.

Variable	Clase válida	Clase inválida	Valor	Id. del valor
Longitud del Texto (LT) CON icono	$1 \leq \text{LT} \leq 25$		1	T1
			25	T2
		$\text{LT} < 1$	0	T3
		$\text{LT} > 25$	26	T4
Longitud del Texto (LT) SIN icono	$1 \leq \text{LT} \leq 30$		1	T1
			30	T5
		$\text{LT} < 1$	0	T3
		$\text{LT} > 30$	31	T6
Tipo de Letra (TL)	Tipo 1		Tipo 1	TL1
	Tipo 2		Tipo 2	TL2
	Tipo 3		Tipo 3	TL3
	Tipo 4		Tipo 4	TL4
Color de la etiqueta (C)	El texto se imprime en blanco		Negro	C1
	El texto se imprime en negro		Blanco	C2
Icono (I)	Con icono		Icono de la Tarta	I1
	Sin icono		Sin icono	I2
Cantidad de Etiquetas (CE)	48 unidades		48	CE1
	96 unidades		96	CE2

Tabla 1: Clases de equivalencia para la dimensión principal

Paso 7 *Reescribir las restricciones entre valores primarios utilizando los identificadores definidos en la tabla creada en el Paso 6*:

- $I1 \Rightarrow (\neg T5 \wedge \neg T6)$
- $I2 \Rightarrow (\neg T2 \wedge \neg T4)$

Paso 8 *Determinar las dimensiones secundarias de interés para cada variable*. Las dimensiones secundarias de una variable de dominio infinito pueden ser muy numerosas. Por ejemplo, en Kaner et al. [3] se proponen 34 dimensiones secundarias para cadenas de texto.

Sólo deberán considerarse las dimensiones secundarias que impliquen un riesgo para nuestro SUT. A este respecto, conviene consultar los textos [3] y [2]. [3] incluye un catálogo de valores de prueba para dimensiones secundarias de distintos tipos de variables (números enteros, reales, cadenas de caracters, etc.). [2] incluye una lista de errores que ocurren con frecuencia en aplicaciones software. Esta lista nos puede dar una idea de posibles dimensiones secundarias y de los riesgos que conllevan.

A modo de ejemplo y sin pretender ser exhaustivos, sino sólo ilustrar lo complejo que es probar a fondo un sistema software real, consideraremos cuatro tipos de dimensiones secundarias para el contenido de *T*. Estudiaremos cómo reacciona el SUT frente a:

- cadenas vacías (strings que únicamente incluyen espacios en blanco, tabuladores, saltos de línea, etc.)
- cadenas con caracteres propios del español
- cadenas con caracteres ASCII[1] de control
- cadenas con caracteres ASCII no alfanuméricos

Paso 9 *Repetir los pasos 4-7 por cada dimensión secundaria*. La Tabla 2 resume las clases de equivalencia y los valores de prueba que utilizaremos para las dimensiones secundarias de *T*. Para evitar que el ejemplo se complique en exceso, sólo hemos incluido una clase y un valor de prueba para cada dimensión secundaria.

Variable	Riesgo (fallo potencial)	Clase que NO debería producir fallos	Clase que SÍ debería producir fallos	Valor	Nota aclaratoria	Id. del valor
Texto	Sólo espacios en blanco		Campo en blanco	" "		T7
		Caracteres propios del español		"Begoña"	Incluye la letra ñ	T8
				"María"	Incluye acentos	T9
		Caracteres ASCII de control		ASCII 24	Cancelar	T10
		Caracteres ASCII no alfanuméricos		ASCII 169	®	T11

Tabla 2: Clases de equivalencia para las dimensiones secundarias de *T*

[1]En el Apéndice 1 se describe qué es el código ASCII y como escribir caracteres de este tipo.

◎ **Conclusión**:

Como resultado de la partición del espacio de prueba en clases de equivalencia, se han identificado:

1. Las siguientes variables y sus respectivos valores de prueba:
   a) $T = \{T1, T2, \ldots, T11\}$
   b) $TL = \{TL1, TL2, \ldots, TL4\}$
   c) $C = \{C1, C2\}$
   d) $I = \{I1, I2\}$
   e) $CE = \{CE1, CE2\}$
2. Las siguientes restricciones entre valores:
   a) $I1 \Rightarrow (\neg T5 \wedge \neg T6)$
   b) $I2 \Rightarrow (\neg T2 \wedge \neg T4)$

4. Apéndice 1: la tabla de caracteres ASCII

El *American National Standards Institute* (*ANSI*) creó el código *ASCII* (*American Standard Code for Information Interchange*) en 1963. Dicho código nació a partir del conjunto de símbolos y caracteres que por entonces usaba la compañía Bell en telegrafía.

En un primer momento, el código sólo incluía letras mayúsculas y números. En 1967 se agregaron las letras minúsculas y algunos caracteres de control, formando así lo que se conoce como *US-ASCII*.

El US-ASCII constaba de tan sólo 128 caracteres y estaba pensado para la escritura de textos en inglés. En 1981, la empresa IBM desarrolló una extensión del código ASCII, donde se reemplazaban algunos caracteres de control obsoletos por nuevos caracteres gráficos. Además, se incorporaban 128 caracteres nuevos para la escritura de textos en otros idiomas, como por ejemplo el español. Así fue como el código ASCII se amplió hasta incluir 255 caracteres.

Hoy día, casi todos los sistemas informáticos utilizan esta versión extendida del código ASCII, que se puede consultar en:

`http://www.elcodigoascii.com.ar/`
`http://www.ascii-code.com/`

En los sistemas operativos Microsoft Windows, los caracteres ASCII se escriben manteniendo pulsada la tecla Alt y tecleando los códigos correspondientes con el **teclado numérico**. Por ejemplo, imaginemos que deseamos escribir el símbolo ®, cuyo código ASCII es el 169. Para ello, seguiremos los siguientes pasos:

1. Presionaremos la tecla Alt
2. Sin dejar de presionar Alt, pulsaremos en el teclado numérico los números 1, 6 y 9
3. Soltaremos Alt

Referencias

[1] Paul C. Jorgensen. *Software Testing: A Craftsman's Approach, 4th Edition*. Auerbach Publications, 2013.

[2] Cem Kaner, Jack Falk, and Hung Quoc Nguyen. *Testing Computer Software, 2nd Edition*, chapter Appendix: common software errors[2]. Wiley, 2012.

[3] Cem Kaner, Sowmya Padmanabhan, and Douglas Hoffman. *The Domain Testing Workbook*. Context Driven Press, 2013.

[4] Glenford J. Myers, Corey Sandler, and Tom Badgett. *The Art of Software Testing, 3rd Edition*. Wiley, 2011.

[5] Macario Polo, Beatriz Pérez, and Pedro Reales. *Técnicas combinatorias y de mutación para testing de sistemas software*. Ra-Ma, 2012.

[2] disponible en `http://www.testingeducation.org/BBST/testdesign/Kaner_Common_Software_Errors.pdf`

Pruebas de Software

Ejemplo de PEC-2: Combinación de valores de prueba

Dpto. de Ingeniería de Software y Sistemas Informáticos

Rubén Heradio, rheradio@issi.uned.es

ETSI Informática, Universidad Nacional de Educación a Distancia

Índice

1. Introducción

Este documento incluye un ejemplo resuelto de Prueba de Evaluación Continua 2 (PEC-2) para la asignatura de Pruebas de Software impartida en la Universidad Nacional de Educación a Distancia.

Este tipo de PEC persigue los siguientes objetivos:

1. Ejercitar el testing combinario. Se trata de que el alumno utilice herramientas gratuitas que después manejará en su futuro profesional.
2. Ilustrar el crecimiento combinatorio. En pruebas de software, el aumento de la cobertura tiene un coste muy alto.
3. Hacer notar que el testing combinatorio es un problema NP-completo. Como consecuencia, (i) es un campo de investigación abierto, y (ii) no hay una herramienta que siempre produzca los mejores resultados. Por ello, el alumno deberá acostumbrase a aplicar varias herramientas en cada problema para seleccionar la que mejores resultados produzca.

2. Enunciado

Como resultado de la PEC: "*Ejemplo de PEC-1: Identificación de los valores de prueba*", se han obtenido:

1. Las siguientes variables y sus respectivos valores de prueba:
   a) $T = \{T1, T2, \ldots, T11\}$
   b) $TL = \{TL1, TL2, \ldots, TL4\}$
   c) $C = \{C1, C2\}$
   d) $I = \{I1, I2\}$
   e) $CE = \{CE1, CE2\}$
2. Las siguientes restricciones entre valores:
   a) $I1 \Rightarrow (\neg T5 \wedge \neg T6)$
   b) $I2 \Rightarrow (\neg T2 \wedge \neg T4)$

En lo que sigue, al igual que Kuhn et al. [2], denotaremos t al número de valores combinados en cada test siguiendo una estrategia t-way[1]. Así:

[1] también llamada t-wise

- $t = 1$ equivale a "cada elección"[2] en [4]
- $t = 2$ equivale a "todos los pares"[3] en [4]
- En nuestro caso, al haber 5 variables (T, TL, C, I y CE), $t = 5$ equivale a la estrategia "todas las combinaciones"[4] en [4]

Realice las siguientes actividades:

1. Calcule un juego de pruebas que combine con $t = 1$ las variables T, TL, C, I y CE, y respete las restricciones antes citadas. Para ello, utilice la herramienta ACTS[a].
2. Realice el cálculo equivalente para $t = 2$. El nuevo conjunto de pruebas debe extender el conjunto obtenido para $t = 1$.
3. Analice el crecimiento combinatorio al incrementar t. Para ello, compare el tamaño de los conjuntos de prueba obtenidos con ACTS para $t = 1, 2, ..., 5$
4. Compare el tamaño de los conjuntos obtenidos con ACTS para $t = 1, 2, ..., 5$ con los obtenidos con otra herramienta[b], por ejemplo PICT[c].

[a] ACTS se puede obtener gratuitamente en los cursos virtuales de la UNED, así como en http://csrc.nist.gov/groups/SNS/acts/index.html

[b] puede encontrar un listado de herramientas para testing combinatorio en http://www.pairwise.org/tools.asp

[c] PICT se puede obtener gratuitamente en los cursos virtuales de la UNED, así como en http://download.microsoft.com/download/f/5/5/f55484df-8494-48fa-8dbd-8c6f76cc014b/pict33.msi

[2] en inglés, "all singles" o "each choice"

[3] en inglés, "all pairs" o "pairwise"

[4] en inglés, "all combinations"

3. Solución

3.1. Juego de pruebas para $t = 1$

ACTS es una aplicación java disponible en dos versiones: una con interfaz gráfica de usuario y otra que funciona por línea de comandos. En este documento se utilizará esta segunda versión.

La Figura 1 es la codificación en ACTS de nuestro problema:

- Las líneas 1-2 describen el SUT para el que deseamos producir el juego de pruebas
- Las líneas 4-9 especifican las variables y sus respectivos valores de prueba
- Las líneas 11-13 codifican las restricciones de combinación de los valores de prueba

```
1  [System]
2  Name: stikets
3
4  [Parameter]
5  T (enum) : T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11
6  TL (enum) : TL1, TL2, TL3, TL4
7  C (enum) : C1, C2
8  I (enum) : I1, I2
9  CE (enum) : CE1, CE2
10
11 [Constraint]
12 (I="I1") => ((T != "T5" && T != "T6"))
13 (I="I2") => ((T != "T2" && T != "T4"))
```

Figura 1: Especificación de los valores de prueba en ACTS

A continuación, ejecutaremos ACTS escribiendo por línea de comandos:

```
java -jar -Ddoi=1 -Doutput=csv acts_cmd_2.92.jar ActsConsoleManager stikets.txt resultado.txt
```

- ACTS viene empaquetado en un .jar (`acts_cmd_2.92.jar`)
- `stikets.txt` y `resultado.txt` son los ficheros de entrada y salida, respectivamente
- Con el parámetro `-Ddoi` se especifica el valor de t
- Con el parámetro `-Doutput` se especifica el formato de salida, que puede ser CSV (Comma Separated Values), EXCEL, NIST y numérico.

La Tabla 1 muestra el juego de pruebas obtenido para $t = 1$.

El objetivo de las combinaciones tipo “todos los valores” es poner a prueba los valores de cada variable. Colateralmente, también se ponen a prueba algunas combinaciones aleatorias de valores.

T	TL	C	I	CE	Id. del test
T1	TL2	C2	I2	CE2	Test 1
T2	TL3	C1	I1	CE1	Test 2
T3	TL4	C2	I2	CE1	Test 3
T4	TL1	C2	I1	CE2	Test 4
T5	TL1	C2	I2	CE1	Test 5
T6	TL2	C1	I2	CE2	Test 6
T7	TL4	C2	I1	CE2	Test 7
T8	TL1	C1	I1	CE1	Test 8
T9	TL2	C1	I2	CE1	Test 9
T10	TL1	C2	I1	CE2	Test 10
T11	TL3	C1	I1	CE2	Test 11

Tabla 1: Combinaciones obtenidas con ACTS para $t = 1$

Como estas combinaciones no son sistemáticas, no hay garantía de que se detecten los fallos que impliquen la interacción de varias variables. Por ejemplo, uno de los fallos del formulario de stikets es que el tipo de letra condiciona la cantidad de caracteres que pueden aparecer en las etiquetas. Así, con el tipo de letra 2 la longitud máxima no es 25 sino 20. Según la notación propuesta en la Tabla 1 de la PEC anterior ("*Ejemplo de PEC-1: Identificación de los valores de prueba*"), la combinación que pondría de manifiesto este error es T2-TL2, que no está incluida en la Tabla 1. Para hacernos una idea de la gravedad del error, imaginemos que estamos diseñando una etiqueta que incluye 22 caracteres con el tipo de letra 1. Si en el último momento decidimos cambiar al tipo de letra 2 y pulsamos "Añadir a la cesta", la etiqueta que recibiremos en casa será erronea. Para detectar estos problemas sistemáticamente, es necesario utilizar valores de t mayores que 1.

3.2. Juego de pruebas para $t = 2$

En escenarios reales, es muy común aumentar el valor de t incrementalmente. Por ejemplo, se empieza a probar el SUT con $t = 1$, a continuación con $t = 2$, etc. Para minimizar el esfuerzo de realizar las pruebas, éstas suelen automatizarse usando frameworks como JUnit, que ejecutan el SUT con los valores de prueba y comparan los resultados obtenidos con los resultados esperados. Para poder reutilizar la codificación de las pruebas en cada incremento de t interesa que los test combinatorios no se generen desde cero, sino utilizando como punto de partida los test calculados en el paso anterior.

Muchas herramientas, como ACTS y PICT soportan esta funcionalidad. En concreto, utilizaremos la Figura 2 para que ACTS calcule las combinaciones correspondientes a $t = 2$ utilizando como punto de partida las combinaciones obtenidas en $t = 1$. Note que las líneas 15-27 de la Figura 2 incluyen los resultados resumidos en la Tabla 1.

Para obtener el nuevo juego de pruebas ejecutaremos ACTS escribiendo por línea de comandos:

```
java -jar -Ddoi=2 -Dmode=extend -Doutput=csv acts_cmd_2.92.jar ActsConsoleManager stikets.txt resultado.txt
```

- Se utiliza `-Dmode=extend` para indicar a ACTS que debe extender los tests recogidos en `stikets.txt`

```
[System]
Name: stikets

[Parameter]
T (enum) : T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11
TL (enum) : TL1, TL2, TL3, TL4
C (enum) : C1, C2
I (enum) : I1, I2
CE (enum) : CE1, CE2

[Constraint]
(C="C1") => ((T != "T5" && T != "T6"))
(C="C2") => ((T != "T2" && T != "T4"))

[Test Set]
T,TL,C,I,CE
T1,TL2,C2,I2,CE2
T2,TL3,C1,I1,CE1
T3,TL4,C2,I2,CE1
T4,TL1,C2,I1,CE2
T5,TL1,C2,I2,CE1
T6,TL2,C1,I2,CE2
T7,TL4,C2,I1,CE2
T8,TL1,C1,I1,CE1
T9,TL2,C1,I2,CE1
T10,TL1,C2,I1,CE2
T11,TL3,C1,I1,CE2
```

Figura 2: Especificación de los valores de prueba en ACTS, extendiendo las combinaciones obtenidas para $t = 1$

- Como t pasa a ser 2, el valor de `-Ddoi` es 2

La Tabla 2 muestra las combinaciones obtenidas con ACTS para $t = 2$. Antes decíamos que con el tipo de letra 2, el número máximo de catacteres erróneamente pasa a ser 20 en lugar de 25. El Test 19, al combinar los valores T2 y TL2, facilitaría la detección de este problema.

T	TL	C	I	CE	Id. del test
T1	TL2	C2	I2	CE2	Test 1
T2	TL3	C1	I1	CE1	Test 2
T3	TL4	C2	I2	CE1	Test 3
T4	TL1	C2	I1	CE2	Test 4
T5	TL1	C2	I2	CE1	Test 5
T6	TL2	C1	I2	CE2	Test 6
T7	TL4	C2	I1	CE2	Test 7
T8	TL1	C1	I1	CE1	Test 8
T9	TL2	C1	I2	CE1	Test 9
T10	TL1	C2	I1	CE2	Test 10
T11	TL3	C1	I1	CE2	Test 11
T1	TL1	C1	I1	CE1	Test 12
T2	TL1	C2	I1	CE2	Test 13
T3	TL1	C1	I1	CE2	Test 14
T6	TL1	C2	I2	CE1	Test 15
T7	TL1	C1	I2	CE1	Test 16
T9	TL1	C2	I1	CE2	Test 17
T11	TL1	C2	I2	CE1	Test 18
T2	TL2	C2	I1	CE1	Test 19
T3	TL2	C2	I1	CE2	Test 20
T4	TL2	C1	I1	CE1	Test 21
T5	TL2	C1	I2	CE2	Test 22
T7	TL2	C2	I2	CE2	Test 23
T8	TL2	C2	I2	CE2	Test 24
T10	TL2	C1	I2	CE1	Test 25
T11	TL2	C2	I1	CE2	Test 26
T1	TL3	C2	I2	CE1	Test 27
T3	TL3	C1	I1	CE1	Test 28
T4	TL3	C1	I1	CE2	Test 29
T5	TL3	C1	I2	CE1	Test 30
T6	TL3	C2	I2	CE1	Test 31
T7	TL3	C1	I2	CE1	Test 32
T8	TL3	C1	I1	CE2	Test 33
T9	TL3	C2	I2	CE2	Test 34
T10	TL3	C1	I2	CE2	Test 35
T1	TL4	C1	I2	CE2	Test 36
T2	TL4	C2	I1	CE2	Test 37
T4	TL4	C1	I1	CE2	Test 38
T5	TL4	C2	I2	CE1	Test 39
T6	TL4	C1	I2	CE1	Test 40
T8	TL4	C2	I2	CE2	Test 41
T9	TL4	C1	I1	CE1	Test 42
T10	TL4	C1	I1	CE1	Test 43
T11	TL4	C2	I1	CE2	Test 44

Tabla 2: Combinaciones obtenidas con ACTS para $t = 2$

3.3. Análisis del crecimiento combinatorio

Los Apéndices A y B de [2] tratan en detalle el crecimiento combinatorio debido a t.

En nuestro caso de estudio, la Figura 3 refleja el crecimiento del número de tests según se incrementa t. Mientras que ACTS proporciona 11 tests para $t = 1$, para $t = 5$ genera 288. Note que, en ausencia de restricciones, el número de tests para $t = 5$ (todas las combinaciones) sería $11 \cdot 4 \cdot 2^3 = 352$.

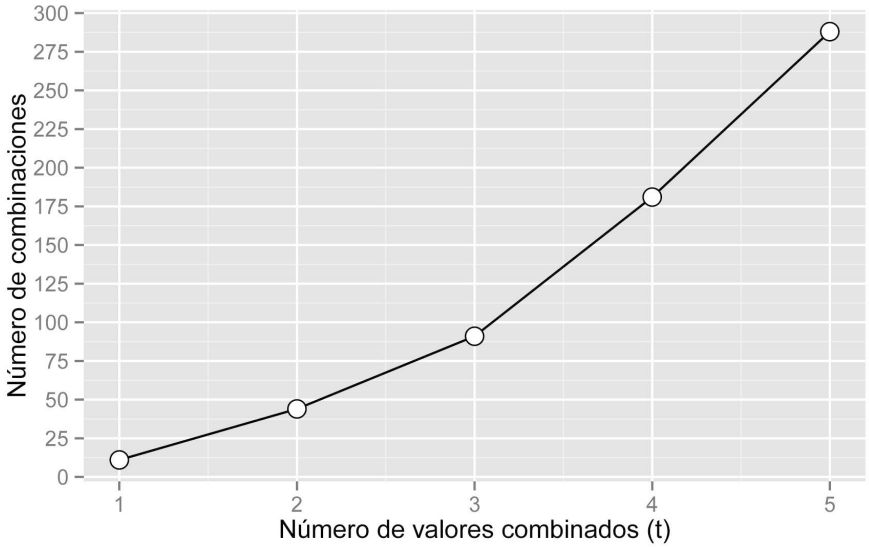

Figura 3: Crecimiento del número de combinaciones según la cantidad de valores combinados

3.4. Análisis comparativo de herramientas: ACTS vs PICT

La obtención de un conjunto mínimo de pruebas para un t dado es un problema NP-completo [3, 1]. Es decir, no existe ningún algoritmo de testing combinatorio que pueda asegurar que el conjunto de pruebas que calcula es el menor de todos los posibles. Como consecuencia, cada herramienta producirá juegos de prueba distintos.

Vamos a comparar el tamaño de los juegos de prueba que generan ACTS y la herramienta PICT de Microsoft.

La Figura 4 muestra la codificación en PICT, análoga a la que utilizamos con ACTS (Figura 1).

Para que PICT genere el juego de prueba para $t = 1$ en un fichero `resultado.txt`, escribiremos el siguiente texto por línea de comandos:

```
pict stikets.txt /o:1 >resultado.txt
```

```
T: T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11
TL: TL1, TL2, TL3, TL4
C: C1, C2
I: I1, I2
CE: CE1, CE2

IF [I] = "I1" THEN (([T] <> "T5") AND ([T] <> "T6"));
IF [I] = "I2" THEN (([T] <> "T2") AND ([T] <> "T4"));
```

Figura 4: Especificación de los valores de prueba en PICT

Si sólo deseamos conocer el tamaño del juego de prueba, basta con escribir:

```
pict stikets.txt /o:1 /s
```

Note que con el parámetro `/o` se especifica el tamaño de t. Por ejemplo, para conocer el número de tests con $t = 3$, escribiremos:

```
pict stikets.txt /o:3 /s
```

La Tabla 3 compara los tamaños obtenidos con ACTS y PICT. Si el SUT fuera más complejo y la cantidad de variables y valores de prueba fueran mayores, la diferencia de los tests sería más notoria.

t	**Número de Tests**	
	ACTS	**PICT**
1 (“all singles”)	11	11
2 (“all pairs”)	44	44
3	91	92
4	181	182
5 (“all combinations”)	288	288

Tabla 3: Comparativa ACTS vs PICT

Referencias

[1] Jacek Czerwonka. Pairwise testing in the real world: Practical extensions to test-case scenarios[5]. *Microsoft Corporation*, 2008.

[2] D. Richard Kuhn, Raghu N. Kacker, and Yu Lei. *Practical Combinatorial Testing*[6]. NIST, 2010.

[3] Yu Lei and K.-C. Tai. In-parameter-order: a test generation strategy for pairwise testing. In *3rd IEEE International Symposium High-Assurance Systems Engineering*, pages 254–261, Nov 1998.

[4] Macario Polo, Beatriz Pérez, and Pedro Reales. *Técnicas combinatorias y de mutación para testing de sistemas software*. Ra-Ma, 2012.

[5] disponible en `https://msdn.microsoft.com/en-us/library/cc150619.aspx`

[6] disponible en `http://csrc.nist.gov/groups/SNS/acts/documents/SP800-142-101006.pdf`