UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

FACULTAD DE
INGENIERÍA
UDELAR

Inferencia Inductiva de la Especificación: una propuesta para evaluar pruebas de caja negra

Jorge Triñanes

Tesis de Maestría presentada a la Facultad de Ingeniería de la
Universidad de la República
en cumplimiento parcial de los requerimientos para la obtención del título de
Magíster en Informática

Tutor
Dr. Diego Vallespir

Tribunal
Dr. Eduardo Miranda
Dr. Héctor Cancela
Dr. Martín Solari

Montevideo, Uruguay
Septiembre de 2017

A Gabriela, Mariana,
Guillermo, Inés y Manuela

Agradecimientos

A Diego Vallespir por su incansable aliento, apoyo, paciencia y dedicación.

Al Grupo de Ingeniería de Software y en especial a los organizadores y asistentes a los seminarios que resultaron muy importantes para aclarar mis ideas.

A Patricia, compañera de mi vida.

Resumen

Las pruebas de software constituyen una actividad relevante para producir software de calidad. En la ejecución de las pruebas juega un rol importante la selección de los casos de prueba, actividad para la que se han propuesto diversas técnicas. El estudio de la efectividad de cada una de esas técnicas para detectar defectos en los programas también constituye un área de investigación abierta. Se han llevado a cabo diversos experimentos con el objetivo de evaluar esa efectividad, pero hasta el momento estos no han permitido extraer conclusiones claras. Existen dos grandes grupos de técnicas de selección de casos de prueba: las que toman en cuenta el código del programa (llamadas de caja blanca) y las que no lo toman en cuenta y solo consideran la especificación de lo que el programa debe hacer (llamadas de caja negra). Las técnicas de caja negra no hacen uso del código del programa por lo que pueden ser usadas incluso antes de que este exista. Cada técnica de caja blanca tiene asociado un criterio de aceptación de conjuntos de casos de prueba basado en el comportamiento del programa cuando ejecuta cada caso del conjunto. El criterio de aceptación permite evaluar si el conjunto de casos de prueba se puede considerar adecuado o no. Para evaluar la efectividad de las técnicas de caja blanca es posible generar de forma automática casos y evaluar los criterios de aceptación, lo que habilita a la experimentación sin recurrir a sujetos que apliquen la técnica. Por el contrario, los experimentos destinados a evaluar la efectividad de técnicas de selección de caja negra han requerido de la participación de sujetos que utilicen esas técnicas. En los experimentos llevados a cabo hasta el momento en los que participaron sujetos aplicando una técnica dada, en general no se ha llevado a cabo un análisis de cuán bien fue aplicada dicha técnica. Sin este análisis los resultados de los experimentos pueden estar brindando información acerca de cuán efectiva es la técnica aplicada por sujetos con determinadas características y en cierto contexto, pero esta pudiera resultar diferente de la efectividad de la técnica en sí. En este trabajo se define un criterio de aceptación de conjuntos de casos de prueba que no toma en cuenta el programa, sino tan solo la especificación. Un conjunto de casos de prueba cumple el criterio si una máquina de inferencia inductiva es capaz de inferir un programa correcto respecto a la especificación a partir de ese conjunto. Este criterio de aceptación puede ser aplicado a técnicas de caja negra y permite evaluar si el conjunto de casos de prueba generado resulta o no adecuado. Este criterio fue aplicado en la evaluación de conjuntos de casos de prueba generados por sujetos en un experimento destinado a estudiar la efectividad de técnicas de selección de casos de prueba de caja negra. A partir de esa evaluación se mostró la factibilidad de la aplicación del criterio con tecnologías actualmente disponibles, al menos para su uso en un ambiente académico. La aplicación del criterio y el análisis de los conjuntos de casos de prueba mostró una gran dispersión en la cantidad y características de los casos. Varios sujetos aplicaron la técnica de forma inadecuada y algunos sujetos generaron casos más allá de lo requerido por la técnica, lo que muestra la conveniencia de que en experimentos de ese tipo se evalúe la forma en que fue aplicada la técnica si se pretende extraer conclusiones sobre la efectividad de la misma. A partir de los resultados obtenidos se identifican diversas posibilidades de trabajo futuro referidas al perfeccionamiento en la aplicación del criterio, a su aplicación en la enseñanza de técnicas de selección de casos de prueba y al desarrollo teórico.

Índice General

Índice de tablas

Índice de ilustraciones

Capítulo 1: Introducción

Los computadores están siendo usados en una creciente variedad de áreas de aplicación y su uso y funcionamiento correctos con frecuencia resultan críticos para el éxito en el funcionamiento de las organizaciones y/o para la integridad física de las personas [ISO25]. Una parte creciente del presupuesto destinado a la producción de software se destina a tareas de aseguramiento de la calidad y testing [WQR15]. El esfuerzo de testing, dependiendo del tamaño del producto y de la calidad en el proceso de desarrollo puede variar entre el 10 y 50% del esfuerzo total de desarrollo [JON11].

1.1 Motivación

En la ejecución de las pruebas juega un rol importante la selección de los casos de prueba, ya que no es factible evaluar el comportamiento siquiera de programas simples con la totalidad de valores de entrada posibles [DIJ70, MYE04]. Se han propuesto diversas técnicas para realizar esa selección. El estudio de la efectividad de cada una de esas técnicas para detectar defectos en los programas constituye un área de investigación abierta. Se han llevado a cabo diversos experimentos con el objetivo de evaluar esa efectividad, pero hasta el momento estos no han permitido extraer conclusiones claras [JUR04, BER07].

Existen dos grandes grupos de técnicas de selección de casos de prueba: las que toman en cuenta el código del programa (llamadas de caja blanca) y las que no lo toman en cuenta y solo consideran la especificación de lo que el programa debe hacer (llamadas de caja negra). Las técnicas de caja negra no hacen uso del código del programa por lo que pueden ser usadas incluso antes de que este exista [MYE04].

Un criterio de aceptación de conjunto de casos de prueba es un medio para evaluar si un conjunto de casos de prueba se puede considerar o no adecuado. Cada técnica de caja blanca tiene asociado un criterio de aceptación de conjuntos de casos de prueba basado en el comportamiento del programa al ejecutar los distintos casos del conjunto [BER03]. Para evaluar la efectividad de las distintas técnicas de selección de casos de prueba se han llevado a cabo numerosos experimentos en los que sujetos aplican distintas técnicas. Para el caso de las técnicas de caja blanca es posible utilizar el criterio de aceptación asociado para evaluar si la técnica fue aplicada de forma adecuada. Este enfoque en general no resulta posible aplicarlo con las técnicas de caja negra. En los experimentos llevados a cabo hasta el momento en los que participaron sujetos aplicando una técnica dada de caja negra, salvo escasas excepciones, no se ha llevado a cabo un análisis de cuán bien fue aplicada dicha técnica [JUR04]. Sin este análisis los resultados de los experimentos pueden estar brindando información acerca de cuán efectiva es la técnica aplicada por sujetos con determinadas características y en cierto contexto, pero esta pudiera resultar diferente de la efectividad de la técnica en sí.

1.2 Contexto

Este trabajo se llevó a cabo en el Grupo de Ingeniería de Software de la Facultad de Ingeniería de UDELAR (Gris) que ha estado trabajando desde 2009 en experimentación para evaluar la efectividad de técnicas de selección de casos de prueba.

1.3 Planteamiento del problema

Los experimentos destinados a evaluar la efectividad de las técnicas de selección de caja negra requieren de la participación de sujetos que aplican esas técnicas. Los resultados obtenidos en distintos expe-

rimentos no han arrojado resultados claros. La participación de sujetos en estos experimentos plantea desafíos para asegurar su validez ya que en cada instancia de un experimento podrían estar influyendo factores de contexto, fuera del control de los experimentadores y que inciden en los resultados del mismo en cuanto a la aplicación de la técnica por parte de los sujetos. Para las técnicas de caja blanca se dispone de criterios de aceptación que permiten evaluar si una técnica fue aplicada o no de forma adecuada, no así para las técnicas de caja negra. Si se dispusiera de un procedimiento que permitiera evaluar si una técnica de caja negra fue aplicada de forma adecuada o no, se podría estimar el impacto sobre los resultados de un experimento de los factores de contexto no controlados.

1.4 Objetivos

El trabajo se plantea a partir de las siguiente pregunta de investigación:

¿Cómo podemos evaluar si una técnica de selección de casos de prueba de caja negra ha sido aplicada de forma adecuada?

La evaluación se puede llevar a cabo considerando el proceso de aplicación de la técnica o considerando el producto de su aplicación independientemente del proceso mediante el cual se obtuvo. Estos dos enfoques a priori resultan complementarios y plantean desafíos distintos.

Se plantean entonces las dos preguntas siguientes:

a) Qué factores pueden incidir en la calidad de la aplicación de una técnica de selección de casos de prueba de caja negra por parte de un sujeto? y

b) ¿Cómo es posible evaluar si un sujeto aplicó una técnica de selección de casos de prueba de caja negra de forma adecuada?

El trabajo se plantea objetivos, tratando de responder a cada una de las preguntas de investigación.

El objetivo general es:

Disponer de un mecanismo que permita evaluar si una técnica de selección de casos de prueba de caja negra fue aplicada de forma adecuada.

Este objetivo se abre en dos sub-objetivos:

a) Estudiar los factores de contexto que pueden incidir en la calidad con la que sujetos aplican una técnica de selección de casos de prueba y,

b) definir un procedimiento análogo a los criterios de aceptación de caja blanca que resulte útil para evaluar si una técnica de caja negra fue aplicada de forma adecuada o no.

1.5 Trabajo realizado y resultados obtenidos

Se hicieron dos revisiones de la literatura. La primera referida a técnicas de selección de casos de prueba y su efectividad. De esta revisión quedó claro que es poco lo que se sabe acerca de la efectividad de las distintas técnicas de selección de casos de prueba y también permitió identificar un conjunto de factores que pueden incidir en la efectividad de la aplicación de una técnica. La otra revisión de la literatura es-

tuvo referida a factores de contexto que inciden en la efectividad de las pruebas, lo que permitió completar la identificación de este tipo de factores.

Del análisis de los factores en su conjunto, se detectó que estos se encuentran relacionados entre sí de manera compleja. Se elaboró un diagrama que muestra las relaciones entre esos factores el que puede ser utilizado para evaluar si el contexto en el que fue aplicada una técnica favorece o no su adecuada aplicación.

Se definió un criterio de aceptación de conjuntos de casos de prueba que únicamente depende de la especificación del programa. Este criterio se evaluó con los conjuntos de casos de prueba elaborados por sujetos en un experimento controlado.

1.6 Estructura

El capítulo 2 presenta conceptos relativos a las pruebas de software y a la selección de casos de prueba. El capítulo 3 presenta la Ingeniería de Software empírica que busca obtener evidencias empíricas que justifiquen (o rechacen) la utilización de técnicas, métodos, herramientas y procedimientos en la construcción de software, más allá de las opiniones y prácticas en boga. Es en este marco que se llevaron a cabo experimentos para evaluar la efectividad de técnicas de selección de casos de prueba. El capítulo 4 presenta los estudios que se han hecho respecto a la efectividad de las distintas técnicas de selección de casos de prueba y analiza factores de contexto que pueden afectar la forma en la que se aplica una de estas técnicas y la detección de defectos. El capítulo 5 presenta el criterio de aceptación de conjuntos de casos de prueba que proponemos. El capítulo 6 presenta una evaluación de distintas alternativas y dos propuestas de maneras de medir el grado de cumplimiento del criterio de aceptación introducido en el capítulo 5. El capítulo 7 describe la aplicación de ese criterio de aceptación en la evaluación de los conjuntos de casos de prueba generados por sujetos en un experimento llevado a cabo con anterioridad destinado a evaluar la efectividad de una técnica de selección de caja negra. El capitulo 8 analiza el criterio de aceptación y presenta una evaluación del mismo. Por último, el capítulo 9 presenta las conclusiones y el trabajo futuro.

Capítulo 2: Pruebas de software

En 2004 la organización IEEE Computer Society, presentó la guía conocida como SWEBOK (Guide to the Software Engineering Body of Knowledge) con el objetivo de compilar el conjunto de conocimientos referidos a la ingeniería de software que se fueron desarrollando y evolucionando en el transcurso de las cuatro décadas anteriores.

2.1 Defecto y Falla

Conviene distinguir estos dos conceptos que están íntimamente relacionados. Un programa presenta una ***falla*** cuando tiene un comportamiento que no resulta adecuado. Por ejemplo no hace lo que tiene que hacer, o lo hace mal, o hace cosas que no debiera [SWE04].

La causa que dio origen a que se diera la falla se denomina ***defecto***. Un defecto puede permanecer sin manifestarse durante múltiples ejecuciones de un programa (inclusive sin que llegue a manifestarse nunca), hasta que se activa y el programa queda en un estado que si se propaga hasta la salida observable deriva en una falla [SWE04].

2.2 Verificación y Pruebas de Software

En la construcción de software una parte importante del esfuerzo total se dedica a la verificación de que el producto final o los productos intermedios cumplen con determinados atributos de calidad [BEI90, BER07]. En general la mayor parte del esfuerzo de verificación consiste en evaluar el comportamiento de los programas [BER07], actividad que en inglés se denomina "testing" y que en español denominamos "prueba". En español el término prueba refiere tanto a demostrar que algo es correcto como a realizar una evaluación o experimento de las cualidades. En lo que sigue los términos "prueba" y "probar", cuando refieran a software deben entenderse con el segundo de los significados y la frase "prueba de software" corresponde a la traducción del inglés "software testing". Asimismo utilizaremos de manera indistinta los términos "prueba" y "test" para referirnos a una prueba en particular de software.

En una primera aproximación, la prueba de software es el proceso de ejecutar un sistema de software para determinar si cumple con su especificación y funciona en el ambiente previsto [WHI00] y es el único procedimiento que nos permite evaluar el comportamiento del software en su ejecución en ese ambiente [BER03].

La actividad de verificación intenta asegurar que el producto fue construido de forma correcta, en el sentido de que cumple con las especificaciones impuestas al producto como resultado de actividades previas y la prueba de software es "la verificación *dinámica* del comportamiento de un programa en un conjunto *finito* de casos, adecuadamente *seleccionados* del dominio de ejecuciones que usualmente es infinito, respecto al comportamiento *esperado*" [SWE04].

En esa definición el término "dinámica" refiere a que la verificación se realiza ejecutando el programa y observando su comportamiento a partir de entradas valoradas. Para ser precisos, el valor de la entrada por sí solo no resulta suficiente para determinar una prueba en la medida que el comportamiento de un sistema de software depende en general del estado del sistema.

Por ejemplo, en los programas interactivos las entradas y salidas se entremezclan y el comportamiento (la salida) depende de la última entrada pero también de las anteriores. En programas que almacenan información en una base de datos es frecuente que ante la misma entrada el comportamiento sea diferente.

En lo que sigue el término "entrada" asociado a un programa debe entenderse que incluye también un estado especificado, en los casos en que el estado resulte relevante para definir su comportamiento.

Corresponde notar que sistemas de software no determinísticos pueden reaccionar de forma distinta a una misma entrada en el mismo estado. Es el caso por ejemplo de sistemas con componentes que ejecutan en paralelo o de forma distribuida en los que el comportamiento del sistema depende de la entrada, del estado y del tiempo de ejecución de cada componente. En este tipo de software para una entrada y un estado dados puede haber más de una salida válida. En lo que sigue solo vamos a tomar en cuenta sistemas de software determinísticos.

Si consideramos un programa como un artefacto que recibe datos de entrada y a partir de estos genera una salida, cada entrada posible puede ser considerada como un caso de prueba.

El término "finito" es importante porque aún para programas simples es tan grande la cantidad de casos teóricamente posibles de probar que su sola ejecución (sin considerar la evaluación de los resultados) llevaría años para completar una prueba exhaustiva - por ejemplo Dijkstra calculó que la prueba exhaustiva de un multiplicador de dos enteros de 27 bits en que cada operación insumía en su momento decenas de microsegundos, llevaría más de 10000 años [DIJ70]. En términos prácticos, el conjunto de casos de prueba posibles de un programa puede considerarse como infinito. El conjunto de casos de prueba a incluir en una prueba, además de ser finito, debe poder ser ejecutado y evaluado en un tiempo que resulte razonable. Por lo tanto los casos que se incluyan en una prueba constituyen necesariamente una muestra del conjunto total.

Que un programa se comporte de forma adecuada con cierto valor de entrada no permite asegurar nada respecto al comportamiento del mismo programa con un valor distinto de entrada. Y dados dos valores de entrada para los que el programa se comporta de forma adecuada no es posible afirmar nada respecto al comportamiento con valores intermedios debido a lo que Ghezzi denomina *falta de continuidad* en el comportamiento [GHE03]. De esto se deriva que "la prueba de un programa puede ser usada para mostrar la presencia de defectos, pero nunca para mostrar su ausencia" [DIJ70].

El término "seleccionados" es importante porque distintos criterios de selección de casos de prueba para construir la muestra pueden generar resultados completamente diferentes en el sentido de que algunas muestras pueden permitir detectar defectos y otras no.

El término "esperados" refiere a que debe ser posible decidir si los resultados obtenidos resultan aceptables o no. En lo que nos ocupa, vamos a considerar que el comportamiento esperado está especificado y el resultado obtenido se compara contra lo especificado.

La prueba de software se aplica usualmente a lo largo de los procesos de desarrollo y mantenimiento a distintos tipos de objeto a evaluar. Este puede variar entre un único módulo, un grupo de módulos o todo un sistema, por lo que se distinguen tres niveles desde el punto de vista del alcance de la prueba: unitaria, de integración y del sistema [SWE04].

En la prueba unitaria se verifica el funcionamiento de una porción de software aislada que puede ser probada de forma separada. Dependiendo del contexto podría ser un programa individual o un componente a su vez compuesto por unidades fuertemente relacionadas. En la prueba de integración el objeto a probar es un conjunto de programas o componentes que interactúan entre sí y en la prueba del sistema se prueba un sistema completo compuesto por un conjunto de programas que interactúan entre sí [SWE04].

Las pruebas también se clasifican en función de los objetivos que persiguen. Desde este punto de vista, se puede querer evaluar algún atributo de calidad en particular como correctitud o conformidad con las especificaciones y requerimientos funcionales, o atributos no funcionales como confiabilidad, desempeño (tiempo de respuesta y consumo de recursos), facilidad de uso, etc. [SWE04].

Otra forma de clasificar las pruebas es en función de objetivos específicos en el marco del proceso de desarrollo. Así por ejemplo el test de regresión persigue evaluar si se produjo una "regresión" o involución del producto respecto a un estado anterior luego de incorporar cambios, o dicho de otra forma, evaluar si algo que funcionaba bien dejó de hacerlo. Otros casos están dados por el test de aceptación con participación del cliente para que este pueda evaluar si el producto cumple con lo solicitado y el test de instalación para evaluar si un producto quedó correctamente instalado [SWE04].

Como la prueba consiste en evaluar el comportamiento de un producto de software al ejecutarlo con una muestra de los casos posibles, la efectividad de la prueba - esto es la medida en que permite lograr los objetivos establecidos para esa prueba- va a depender de los objetivos de la prueba y de las características de la muestra.

2.3 Selección de casos de prueba

En general un criterio de aceptación sirve para decidir cuán "bueno" es un conjunto de casos de prueba (CCP). Se puede definir formalmente como sigue.

Un criterio de aceptación C es un predicado de decisión definido sobre triples (P, RM, T), en donde P es un programa, RM es un modelo de referencia relacionado a P y T es un conjunto de casos de prueba. Cuando se cumple C(P,RM,T), se dice que T satisface el criterio C para P y RM [BER03].

Un criterio de test puede ser utilizado de forma proactiva como técnica para guiar la selección de casos de prueba de forma tal que al terminar la selección el criterio se cumple de forma automática o se puede utilizar a posteriori para determinar si el conjunto de casos de prueba seleccionado de alguna otra forma resulta suficiente [BER03].

Un criterio de test y una técnica de selección de casos de prueba responden entonces a la misma definición formal, aunque difieren en su aplicación práctica, por lo que podremos afirmar que un conjunto de casos de prueba satisface una técnica de selección si el conjunto satisface el criterio de test correspondiente [ZHU97].

En pruebas de correctitud (también denominadas como de conformidad con las especificaciones) de un producto de software, la efectividad de la prueba estará dada por la cantidad de defectos que permita identificar de los defectos que contenga el producto. En este contexto la selección de los casos de prueba debe estar orientada a poner de manifiesto (generar) fallas originadas por distintos defectos [KAN99].

La cantidad de casos de una prueba tiene impacto sobre el esfuerzo necesario para definir, ejecutar y evaluar los resultados de la prueba. La decisión de agregar un caso adicional a una prueba va a estar dada esencialmente por una solución de compromiso entre el esfuerzo adicional y la probabilidad de encontrar un nuevo defecto [KAN99].

Una técnica de selección de casos de prueba sirve para obtener un "buen" conjunto de casos de prueba. Sin embargo, esto puede tener más de un significado, dependiente del contexto, de la aplicación específica y de los objetivos de la prueba. La interpretación más común es que "buen" sería "permitir detectar muchos defectos" pero resulta conveniente aclarar qué tipos de defectos ya que diferentes técnicas suelen permitir detectar distintos tipos de defectos [BER03, BAS87]. Un estudio de Adams muestra que muchos defectos presentes en el software en producción no son efectivos en el sentido de que no generan fallas a lo largo de su vida y llega a afirmar que, si pudiéramos predecir qué defectos van a generar fallas, tan solo valdría la pena corregir el diez por ciento de los defectos [ADA84]. Tanto los tipos de defectos como la probabilidad de generar una falla durante el uso del software son factores que limitan la validez de la cantidad de defectos detectados como un indicador de la "bondad" de un método de selección de casos de prueba [BER03].

2.4 Técnicas de Caja Blanca y de Caja Negra

Existen dos grandes grupos de técnicas de selección de casos de prueba. Por un lado están aquellas que utilizan como fuente de información el código del programa, denominadas de "caja blanca" (o "estructurales") para las que el código es visible y por otro aquellas para las que el código es una "caja negra" del que no se tiene información (a las que también se denomina como "funcionales").

Entre las técnicas de caja blanca están: cobertura de sentencias, cobertura de condiciones, cobertura de condiciones múltiples, cobertura de definiciones y usos de variables.

La técnica de cobertura de sentencias está basada en el criterio de aceptación de cobertura de sentencias que consiste en que dado un programa P, un conjunto de casos de prueba cumple el criterio si toda sentencia del programa es ejecutada al menos una vez al ejecutar el conjunto de casos de prueba completo. De forma análoga, la técnica de cobertura de condiciones está basada en el criterio de aceptación de cobertura de condiciones. En este caso un conjunto de casos de prueba cumple el criterio si toda condición del programa del tipo A>B es evaluado al menos una vez con cada uno de sus valores posibles (verdadero y falso).

A partir de un criterio de aceptación, la técnica asociada responde al esquema:

Partir del CCP vacío

  Mientras el CCP no cumpla el criterio
    Agregar un caso de prueba al CCP que mejore la cobertura
  Fin Mientras

Las técnicas de Caja Negra parten de la especificación del programa y tratan de cubrir cada una de las condiciones posibles de la especificación con al menos un caso. Las técnicas Partición en Clases de Equivalencia y Análisis de Borde son técnicas de caja negra.

2.5 Partición en Clases de Equivalencia y Análisis de Borde

La técnica de Partición en Clases de Equivalencia consiste en dividir el conjunto de datos de entrada en clases para las que se espera un comportamiento análogo respecto a la detección de un defecto. Si un elemento de una clase detecta un defecto, espero que cualquier otro elemento de la misma lo detecte. A la inversa, si un elemento de una clase no detecta un defecto, esperamos que cualquier otro elemento de la clase tampoco lo detecte. Sin embargo sí podría detectar otro defecto, ya que las clases pueden solaparse.

La técnica se compone de dos pasos:

a) identificar las clases de equivalencia y
b) definir casos de prueba que cubran todas las clases de equivalencia.
Hay dos tipos de clases de equivalencia: las clases válidas y las clases inválidas.

Las clases válidas son aquellas que contienen datos de entrada válidos para el programa y las inválidas son las que contienen algún dato inválido.

La identificación de clases de equivalencia a partir de la especificación es en gran medida un proceso heurístico. Si hay alguna razón para suponer que el programa no trata elementos de una clase de manera idéntica, corresponde partir esa clase en clases más pequeñas.

El segundo paso consiste en usar las clases de equivalencia para identificar los casos de prueba. Este proceso es como sigue:

1. Asignar un número único a cada clase de equivalencia

2. Hasta que todas las clases de equivalencia válidas hayan sido cubiertas por (incorporadas en) casos de prueba, escribir un nuevo caso de prueba que cubra tantas clases de equivalencia válidas como sea posible.

3. Hasta que los casos de prueba hayan cubierto todas las clases inválidas, escribir un caso de prueba que cubra una y solo una de las clases de equivalencia inválidas.

Para la técnica de partición en clases de equivalencia resulta indiferente la elección de cualquier caso que integre una clase, salvo en lo referido a la posibilidad de cubrir más de una clase con un mismo caso [MYE04].

La técnica de análisis de borde pone el acento en los valores de borde que cambian el comportamiento del programa. Para cada clase de equivalencia con borde se debe incluir al menos un caso en cada lado del borde y casos con valores alejados al borde. Por ejemplo sea un programa que calcula la suma de dos variables X e Y enteras y devuelve su suma si esta es menor a 100 y si esta es mayor o igual que 100 enciende una alarma. Para la técnica de partición en clases de equivalencia una clase válida sería X+Y < 100. Utilizando esta técnica cualquier pareja de enteros (X,Y) cuya suma sea menor que 100 sería apropiada. Utilizando la técnica de análisis de borde se debiera incluir un caso en el que X+Y=99 que es el mayor valor suma de dos enteros que es menor a 100, incluir también un caso en el que X+Y=100 y algún caso con X+Y <100 alejado de 100 y algún caso en que X+Y>100.

Capítulo 3: Ingeniería de Software Empírica

La ingeniería de software empírica se puede considerar como una escuela de pensamiento en el marco de la ingeniería de software que aboga por la validación empírica de las técnicas, los métodos, las herramientas y los procedimientos que se utilizan para construir, mantener y gestionar software. Esta línea de pensamiento considera que la construcción de software debiera estar basada en una mayor y mejor validación empírica de las ideas y opiniones respecto a la mejor forma de construir software [BAS86, BAS96, TIC97, JUR01, SJO07]. Este enfoque tuvo un desarrollo importante a partir del trabajo de Victor Basili y su grupo durante la década de 1970 en la Universidad de Maryland y este foco aparece reflejado en foros tales como Journal of Empirical Software Engineering, IEEE International Symposium on Software Metrics (METRICS desde 1993), Empirical Assessment and Evaluation (EASE desde 1997), IEEE International Symposium on Empirical SE (ISESE desde 2002) [SJO05]. ISESE y METRICS se fundieron en 2007 en International Symposium on Empirical Software Engineering and Measurement (ESEM)..

Sjoberg et al. identifican los siguientes tipos de estudios empíricos:

a) Experimentos (controlados)
b) Encuestas (“surveys” en inglés)
c) Estudio de casos
d) Investigación-acción.

Consideran como “experimento” a un estudio en el que se realiza una intervención deliberada para observar sus efectos. Debido a que en la literatura de ingeniería de software el término “experimento” se usa frecuentemente de forma inconsistente como si fuera sinónimo de “estudio empírico”, agregaron el adjetivo “controlado” para evitar confusiones.

Uno de los hallazgos principales de su estudio es que los informes con frecuencia resultan vagos y carecen de una terminología consistente. Terminan recomendando que los investigadores informen de manera precisa los puntos siguientes:

- Tipo y número de sujetos, incluyendo la tasa de mortalidad (sujetos que dejan de participar en la prueba).
- Variables de contexto tales como experiencia general en ingeniería de software y experiencia específica para las tareas de los experimentos.
- Forma en que los sujetos fueron reclutados.
- Las áreas de aplicación y los tipos de tareas.
- La duración de las tareas.
- Validez interna y externa de los experimentos, siendo específicos respecto a la muestra y la población objetivo del experimento [SJO05].

A partir de esta caracterización queda claro que la ingeniería de software empírica, si bien ha crecido mucho en los últimos 40 años, es una disciplina muy poco madura y que tiene por delante múltiples desafíos. El uso predominante de estudiantes en los estudios empíricos es un factor limitante de la validez externa de los resultados, esto es la medida en que los resultados de los experimentos se pueden extrapolar a lo que sucede en la industria.

En el marco de la conferencia "International Conference on Empirical Software Engineering Issues, Critical Assessment and Future Directions 2006" se llevó a cabo un trabajo en grupo para analizar el rol de los experimentos controlados en la investigación en ingeniería de software [JED07]. Como resultado de este trabajo se lograron una serie de acuerdos, los que se reseñan a continuación.

a) Las principales características que deben tener los experimentos controlados son: muestras pequeñas, asignación al azar a los tratamientos (no necesariamente muestreo al azar), control de factores extraños, instalaciones artificiales de laboratorio, tareas (pequeñas) y artefactos (artificiales) bien definidos, entrenamiento adecuado para las tareas, estudiantes o profesionales como sujetos.
b) Las motivaciones para diseñar y ejecutar experimentos controlados deberían ser: investigar/comprender relaciones de causa-efecto, buen mecanismo para obtener evidencia inicial acerca de esas relaciones, útiles como preparación para estudios de campo, para evaluar la conveniencia de llevar a cabo un estudio de campo, proporcionan un mayor control sobre variables extrañas, pueden ser utilizados como un mecanismo pedagógico, útiles en caso de tecnologías que no están en uso en la industria, para ciertas cuestiones presentan menos amenazas a la validez que otros tipos de estudios, tienden a ser más fáciles de replicar.

3.1 Conceptos básicos referidos a experimentos

Los experimentos constituyen una vía para obtener información acerca de la realidad y generar conocimiento acerca de esta, en particular acerca de relaciones de tipo causa-efecto. Un tipo de *experimento* en ingeniería de software consiste en que *sujetos* apliquen distintas *alternativas* (también llamadas *niveles* o *tratamientos*) para diversos *factores* (también denominados *predictores* o *variables independientes*) sobre *unidades experimentales* (también denominados *objetos experimentales*) con el objetivo de evaluar el impacto sobre una *variable de respuesta* (también llamada *variable dependiente*), en un ambiente definido por un conjunto de *parámetros*. El objetivo de un experimento de este tipo es evaluar una hipótesis referida al efecto de la aplicación de distintas *alternativas* por parte de *sujetos* para cada uno de los *factores* en un ambiente definido por un conjunto de *parámetros* sobre la *variable de respuesta* asociada a los *objetos* [JUR01].

En un experimento de ingeniería de software referido a la efectividad de técnicas de verificación, las unidades experimentales podrían ser programas, las alternativas estarían dadas por cada una de las técnicas de verificación a aplicar para el único factor "técnica de verificación" y los sujetos serían las personas que aplican las técnicas de verificación sobre los programas. La variable de respuesta podría estar dada por la cantidad de defectos encontrados por los sujetos a partir de la aplicación de cada una de las técnicas. El ambiente o contexto de aplicación de las técnicas estaría dado por un conjunto de parámetros, entre los que se podrían mencionar a modo de ejemplo: entrenamiento de los sujetos en el uso de las técnicas, herramientas de apoyo, duración de las actividades, incentivos para realizar la tarea.

3.2 Factores que inciden en las evaluaciones empíricas

Kitchenham analiza las dificultades que enfrenta la experimentación en ingeniería de software. Refiere que en medicina la regla de oro para la experimentación es el método de doble ciego en el que tanto el médico como el paciente ignoran qué tratamiento está recibiendo. En el caso de ingeniería de software los métodos y técnicas a evaluar deben ser llevados a cabo por personal con ciertas habilidades lo que impide que estas personas y que los experimentadores sean ciegos al tratamiento al que están siendo sometidos.

El protocolo del doble ciego se impone para evitar que las expectativas de pacientes y doctores sesguen los resultados. Este tipo de protocolos son imposibles en experimentos que están basados en un sujeto que lleva a cabo una tarea que utiliza de forma intensiva el factor humano. Propone dos enfoques complementarios para abordar este problema:
1) Desarrollar y adoptar protocolos que reducen los sesgos originados por experimentadores y sujetos.
2) Aceptar que los experimentos en ingeniería de software están limitados a ser menos rigurosos que los de medicina y tratar de calificar los experimentos de manera apropiada [KIT04].

Al informar el resultado de un experimento se deben considerar todas las amenazas que podrían tener un impacto sobre la validez de los resultados y se deben incluir al menos los aspectos siguientes: validez de la construcción, validez interna, validez externa y si resulta aplicable, validez de las conclusiones [JED08].

La validez de la construcción refiere al grado en que la especificación de las mediciones en un estudio permite representar de forma adecuada las construcciones del mundo real. Por ejemplo para medir la efectividad de una técnica de selección de casos de prueba se puede utilizar el porcentaje de defectos encontrados mediante esa técnica respecto a la cantidad total de defectos de un artefacto. La validez de construcción de esta medida es el grado en que la efectividad de la técnica está relacionada con ese porcentaje.

La validez interna refiere a la medida en que el tratamiento o las variables independientes fueron realmente responsables por los efectos observados en la variable dependiente. Factores desconocidos pueden haber influenciado sobre los resultados y por lo tanto limitar la validez interna del estudio. Un estudio puede tener validez interna pero carecer de validez de construcción, podría ser cierto que las manipulaciones en el estudio realmente afectaron el resultado, pero las manipulaciones no se correspondieron o representaron de forma adecuada la entidad deseada en el mundo real.

La validez externa refiere al grado en que los hallazgos de un estudio se pueden generalizar a otras poblaciones participantes o ambientes.

La validez de las conclusiones refiere a si las conclusiones del estudio son correctas. En experimentos controlados la validez de las conclusiones está directamente relacionada con la aplicación de tests estadísticos a los datos. Si estos se aplican de forma inadecuada se tiene una amenaza a la validez de las conclusiones.

No basta con mencionar que existe una amenaza a la validez, se deben aclarar las implicaciones de cada amenaza respecto a los hallazgos.

Si existen otros tipos de amenazas a la validez, estas también deben ser informadas, tales como participación de partes interesadas en el estudio o cuestiones éticas relacionadas con la selección de los participantes y en especial dependencias entre sujetos y experimentadores [JED08].

Capítulo 4: Estudio de técnicas de selección de casos de prueba

4.1 Estudios empíricos de la efectividad de técnicas de selección de casos de prueba

En el artículo "Reviewing 25 Years of Testing Technique Experiments" [JUR04] se clasifican las técnicas de selección de casos de prueba en las siguientes familias:

- Aleatoria
- Funcional
- Flujo de control
- Flujo de datos
- Mutación
- Regresión
- Mejora.

Tomando como base esas familias, clasifica los estudios en las categorías siguientes:

a) Estudios intra-familia
   1. Sobre técnicas de flujo de datos
   2. Sobre técnicas de mutación
   3. Sobre técnicas de regresión.

b) Estudios inter-familia
   1. Comparaciones entre las técnicas de flujo de control, flujo de datos y aleatorias
   2. Comparaciones entre las técnicas funcional y estructural de flujo de control
   3. Comparaciones entre las técnicas de mutación y de flujo de datos
   4. Comparaciones entre las técnicas de regresión y de mejora.

Los estudios intra-familia sobre técnicas de flujo de datos, de mutación y de regresión son realizados considerando en cada caso como sujetos experimentales a un conjunto de programas.

Lo mismo sucede en los estudios inter-familia de comparaciones entre las técnicas de flujo de control, flujo de datos y aleatorias, entre las técnicas de mutación y de flujo de datos y entre las técnicas de regresión y de mejora.

Todos estos estudios, dadas las características de las técnicas, admiten la posibilidad de partir de un conjunto de casos de prueba generados previamente o generar casos de forma aleatoria y evaluar a posteriori si un conjunto de casos cumple o no con un criterio dado, o para las pruebas de regresión, filtrar los casos de prueba ya existentes aplicando diversos criterios.

Los estudios inter-familia de comparaciones entre las técnicas funcional y estructural de flujo de control toman por el contrario como sujetos a personas que aplican las técnicas. Para las técnicas funcionales no existe la posibilidad de evaluar la cobertura a posteriori para un conjunto de casos de prueba dado.

En los experimentos referidos a técnicas de selección de casos de prueba aparecen dos factores potencialmente invalidantes:

1) los programas (se debe trabajar con un conjunto particular de programas) y
2) los datos de prueba (un conjunto particular de datos de prueba se debe generar para cada método).

La elección de los programas plantea una amenaza a la validez externa.

Si los datos de prueba son generados por una persona, esto requiere que utilice información acerca del programa o de su especificación, la que puede ser utilizada de una forma inconsciente que distorsione el experimento. Por ejemplo, independientemente de las directivas que haya recibido el sujeto para generar datos de prueba, podría utilizar su experiencia personal para elegir datos que a su juicio tienen alta probabilidad de detectar un defecto [HAM87], lo que plantea una amenaza a la validez de construcción.

4.2 Nuestro conocimiento acerca de la efectividad de las técnicas de selección de casos de prueba y factores que la afectan

Está reconocida la necesidad de desarrollar investigación adicional que proporcione evidencia analítica, estadística o empírica de la efectividad de los criterios de selección de casos de prueba para revelar defectos [HAR00, BER07].

Juristo et al. como resultado del análisis de los 25 años previos de experimentos referidos a técnicas de testing llegan a la conclusión de que el conocimiento al que se ha llegado referido a las técnicas de testing es muy limitado [JUR04]. Runeson y otros a partir de un análisis de 12 estudios empíricos llegan a conclusiones similares [RUN06].

4.2.1 Relación entre el uso de la técnica y la detección de defectos

A fines de las década de 1980 Hamlet planteó que una de las debilidades de las comparaciones entre métodos de testing realizadas hasta ese momento estaba dada porque solo tienen sentido si las propiedades comparadas están relacionadas con la calidad del software a partir de una teoría consistente [HAM89].

Un conjunto muy amplio de técnicas de selección de casos de prueba responden al modelo de partición en clases de equivalencia. Según este modelo, el conjunto de datos de entrada se divide en subconjuntos para los que se espera un comportamiento análogo (por ejemplo calcular un valor de acuerdo a una función o emitir un mensaje de error). Una vez identificados los subconjuntos, basta con ejercitar el comportamiento del programa con un representante cualquiera de cada uno de los subconjuntos. Esta descomposición puede llevarse a cabo utilizando toda la información disponible acerca de un programa como ser la especificación, lo que da origen a técnicas de prueba funcionales o de “caja negra” que no requieren de la información del código del programa, o utilizando el código del programa, que corresponde a técnicas estructurales o de “caja blanca”. Estos subconjuntos pueden ser disjuntos y formar una partición del conjunto de datos de entrada o, en otros casos como en métodos de cobertura y pruebas de mutantes pueden solaparse [HAM90].

Hamlet y Taylor contrastan la probabilidad de encontrar defectos utilizando un método que responda a ese modelo con la probabilidad de encontrar defectos realizando pruebas aleatorias y llegan a la conclusión de que en general los métodos de partición no resultan muy ventajosos sobre las pruebas aleatorias a menos que existan subconjuntos que tengan baja probabilidad de ser ejecutados de forma aleatoria y una alta probabilidad de contener defectos [HAM90]. En la práctica se ha detectado que los defectos suelen estar agrupados y escondidos en los “rincones” de los programas [HUM89] y que la probabilidad de que una

sección de un programa contenga más defectos es mayor cuanto mayor sea la cantidad de defectos ya encontrados en esa sección [MYE04], lo que justificaría el uso de los métodos de partición. La distribución de los defectos en un programa constituye por lo tanto un factor relevante en la efectividad de las técnicas de selección de casos de prueba, por lo que podría afectar la validez externa de un experimento.

Entre las comparaciones teóricas se destacan las relaciones de inclusión (subsume) entre las distintas técnicas de selección de casos de prueba. Se dice que un método A incluye (subsumes) a un método B si y sólo si un conjunto de casos de prueba que satisface A necesariamente satisface B [HAM89]. Esta relación establece un orden parcial entre las distintas técnicas. De forma intuitiva, podemos considerar que una técnica de selección A que incluye a otra B significa que A es más exigente que B (en sentido estricto sería que A es al menos tan exigente como B).

Estas relaciones de inclusión no aseguran nada respecto a la efectividad de conjuntos de casos de prueba específicos que cumplen con uno u otro criterio. Por ejemplo es posible tener un conjunto de casos de prueba que cumple con un criterio B que sí detecta defectos que no son detectados por un conjunto de casos de prueba A que cumple con un criterio más exigente que incluye al anterior [HAM89].

Frankl, Weyuker muestran que las relaciones de inclusión entre técnicas de selección de casos de prueba no permiten afirmar que la probabilidad de encontrar defectos utilizando una técnica A que incluye a una B es mayor que si se utilizara la técnica B [FRA93]. Zhu muestra que la relación de inclusión entre criterios de aceptación de casos de prueba (esto es utilizando los criterios a posteriori para validar un conjunto de casos de prueba seleccionado al azar) sí permite asegurar que la probabilidad de encontrar un defecto con un criterio de aceptación A que incluye a otro B es mayor que si se hubiera aplicado el criterio B [Zhu96].

En la práctica de testing de software quien prueba puede comenzar a generar casos de prueba sin tomar en cuenta ningún criterio de aceptación hasta que eventualmente evalúa el grado de cobertura alcanzado y en caso de resultar necesario utiliza el criterio de aceptación como técnica para seleccionar casos de prueba adicionales, por lo que en la aplicación práctica puede darse una combinación de las siguientes estrategias extremas:

a) usar la técnica de selección de casos de prueba de forma sistemática para guiar la identificación de casos de prueba, de forma tal que el criterio de aceptación correspondiente se cumple de forma automática y

b) generar casos de prueba de alguna manera y usar el criterio de aceptación para dar por terminada la generación de casos [Zhu96].

A partir de los resultados de Zhu, en un experimento que persiga evaluar la efectividad de una técnica de test resulta relevante distinguir si la técnica fue utilizada de forma proactiva para guiar la selección de los casos de prueba o solo para dar por terminada la tarea, lo que podría constituir una amenaza a la validez interna. Sin embargo, difícilmente se pueda discernir esto solo a partir del análisis de los casos de prueba generados.

Algunas técnicas están enteramente basadas en la intuición y comprensión humanas del comportamiento del sistema y se ha observado que distintos sujetos generan a partir de su aplicación resultados muy dispares a pesar de que el resto de las condiciones son idénticas [BRI07, BRIA07]

Hamlet advierte también acerca de lo que considera satisfacción "trivial" de un método de selección de casos de prueba. En sus palabras, "se satisface la letra del método pero con valores que tienen una baja probabilidad de exponer una falla" [HAM89].

Forgacs y Bertolino ponen al descubierto lo que denominan como el "sindrome de no estar probado" ("untestedness syndrome"), una debilidad que es común a un conjunto muy amplio de técnicas de selección de casos de prueba. Este se da cuando un conjunto de casos de prueba satisface el método pero sin embargo algunas instrucciones del programa permanecen sin ser probadas ("untested") en el sentido de que la salida observada no depende de esas instrucciones [FORG02].

4.2.2 Dificultades en la enseñanza de técnicas de selección de casos de prueba

Kaner y otro plantean dificultades en la enseñanza de la técnica de selección de casos de prueba de partición en clases de equivalencia (referido como "domain testing" en el artículo). Indican que esta técnica parece fácil de explicar pero los estudiantes pueden mostrarse como que la conocen y pensar que la conocen mucho antes de resultar competentes en su uso. En términos de la taxonomía de Bloom en el dominio cognitivo [KEN07], la enseñanza de testing requiere brindar conocimiento fáctico y conceptual en los niveles de memorización y comprensión, pero los estudiantes en su proceso de aprendizaje deben además analizar situaciones y problemas, aplicar técnicas y evaluar su propio trabajo y el de sus pares, aspectos estos que corresponden a niveles superiores de aprendizaje.

En la evaluación de los resultados de ejercicios llevados a cabo por estudiantes en cursos de corta duración de tipo seminario con algunos ejercicios prácticos, se detectaron insuficiencias de entidad en lo atinente a niveles superiores de pensamiento en la taxonomía de Bloom. Se propusieron entonces desarrollar un curso que mejorara sustancialmente al curso típico de testing, esperando mejorar sustancialmente las capacidades de los estudiantes para aplicar técnicas clave de testing, brindando a los estudiantes más ejemplos del uso de esas técnicas, más ejercicios de práctica y mayor retroalimentación respecto de los resultados de los ejercicios. El curso se centró en "domain testing" y tuvo una duración de 18 horas de clase. Las evaluaciones del curso por parte de los estudiantes fueron muy buenas y los estudiantes evaluaron que habían logrado un nivel de conocimiento mucho mejor que el que tenían al comienzo del curso. El curso incluyó una prueba final y los evaluadores llegaron a la conclusión de que los estudiantes aprendieron bien los procedimientos pero no aprendieron bien "domain testing". Concluyeron que este curso con mucha práctica y un estilo procedimental de instrucción puede preparar a los estudiantes para una evaluación que requiera los mismos análisis y aplicaciones tal como fueron practicadas. A partir de este resultado Kaner reestructuró sus cursos de testing de forma que las presentaciones están disponibles en la Web, los estudiantes las miran antes de asistir a clase y el tiempo de clase se dedica a discusiones y actividades, pero remarca que el desafío no es tan solo dejar tiempo libre para mucha actividad ya que el curso evaluado ya contenía mucha práctica, sino que se trata de desarrollar actividades que permitan a los estudiantes incorporar el conocimiento de forma que puedan aplicarlo de manera creativa en situaciones nuevas . [KAN07].

En dos replicaciones de un experimento en el que sujetos estudiantes fueron entrenados en el uso de técnicas de selección de casos de prueba para luego aplicarlas, Kamsties y Lott preguntaron a los sujetos por su nivel de dominio de cada técnica. En la primera replicación la pregunta se hizo antes del experimento y en la segunda una vez que estuvo terminado. En el análisis de los resultados encontraron una mucho mayor correlación entre el nivel de dominio informado por los sujetos con la efectividad en la detección de defectos en la segunda replicación que en la primera [KAM95], lo que permite suponer que los sujetos ajustaron su percepción del dominio de la técnica una vez que la aplicaron.

Las dificultades en la enseñanza de la técnica de partición en clases de equivalencia deben considerarse por lo tanto como una amenaza a la validez interna de los experimentos que persiguen evaluar la efectividad de esa técnica.

Chen y otros refieren un curso de testing en el que se presentan seis técnicas: Domain Testing, Coverage Testing, Combination Testing, Adaptive Random Testing, Cluster Test Selection, Simlarity-based Test Selection. Cada uno de los métodos de testing se presenta por separado y de forma ordenada tal que los primeros métodos preparen la introducción de los siguientes. El método elegido para el comienzo es Domain Testing porque es fácil de explicar y porque a los estudiantes les puede llevar un tiempo dominarlo. Como resulta difícil encontrar las clases de equivalencia, resulta natural pasar de Domain Testing a Random Testing y de este a Adaptive Random Testing. Domain Testing también hace pensar acerca de cómo utilizar el código fuente si no se cuenta con la especificación. A partir de esto se presenta Coverage Testing. Cuando los estudiantes identifican de manera adecuada clases de equivalencia suelen enfrentar el problema de la explosión combinatoria para múltiples variables de entrada, por lo que naturalmente se puede introducir Combination Testing [CHE11].

La técnica de Partición en Clases de Equivalencia (Domain Testing) se puede considerar como básica desde el punto de vista conceptual, por lo que las dificultades para lograr su dominio por parte de sujetos que no la conocen podrían también extenderse a otras técnicas más elaboradas.

Las dificultades en la enseñanza de técnicas de testing constituyen otro factor que podría limitar la validez interna de experimentos que requieran entrenamiento en el uso de esas técnicas y esto resulta especialmente relevante para experimentos que involucren a estudiantes.

4.2.3 Sesgos cognitivos

En el campo de la psicología cognitiva referida a juicios y toma de decisiones se ha prestado mucha atención a las heurísticas y a los sesgos cognitivos. Las heurísticas pueden considerarse como “reglas del dedo pulgar”. Estas a veces resultan apropiadas y a veces no. Cuando no lo son y dificultan los juicios, se les considera como “sesgos cognitivos”. Uno de estos sesgos cognitivos es el sesgo positivo (también conocido como sesgo confirmatorio o de confirmación) que consiste en la tendencia a poner a prueba una hipótesis referida a cierto fenómeno con datos diseñados para confirmar la hipótesis, más que con datos que pudieran refutarla. En muchas situaciones, el sesgo positivo puede resultar en que la evaluación de una hipótesis se lleve a cabo de forma no efectiva o no eficiente. Hay estudios que muestran que este fenómeno puede resultar muy fuerte, al punto de que la estrategia de instruir a sujetos en el uso de técnicas de selección de datos que podrían refutar la hipótesis no siempre logra reducir el sesgo. En estudios sobre sesgo positivo se han detectado tres factores que pueden afectarlo: nivel de conocimiento y experiencia previa de los suje-

tos, nivel de especificación de la regla a ser evaluada en lo atinente a lo completa de su representación por parte del sujeto y la presencia de errores [LEV94].

Al probar un programa se deben escribir casos de prueba para las condiciones válidas y esperadas y también para entradas inválidas y no esperadas y se debe examinar si un programa hace lo que se supone que debe hacer así como también evaluar si hace lo que se supone que no debe hacer [MYE04]. El sesgo positivo en testing corresponde a poner el énfasis en las entradas válidas y esperadas y en lo que se supone que el programa debe hacer en detrimento de las entradas inválidas y no esperadas y lo que se supone que no debe hacer. Otra manifestación del sesgo positivo consiste en poner el énfasis en entradas "simples" o con baja probabilidad de detectar defectos, como por ejemplo no considerar situaciones de borde.

Leventhal y otros investigaron la presencia de sesgo positivo en el testing de software, y la relación de este con el nivel de formación y experiencia en desarrollo y testing de software de los sujetos, el nivel de detalle de las especificaciones y la presencia de errores en los programas. En el estudio usaron un patrón de referencia generado usando la técnica de selección de casos de prueba por partición en clases de equivalencia y análisis de condiciones de borde con las dos clases principales: a) datos válidos y b) datos inválidos. No consideran dentro del sesgo positivo el no tomar en cuenta situaciones de borde. Como resultado de estos estudios, concluyeron que en testing de software se da el sesgo positivo, en la medida que encontraron una marcada tendencia a generar casos de prueba referidos a datos válidos. Este sesgo se reduce con mayor formación y experiencia de los sujetos y con mejores especificaciones. También notaron que la detección de errores en los programas por parte de los sujetos, en algunos sujetos reforzó el sesgo positivo y en otros lo redujo [LEV94].

El contar o no con información acerca de los errores detectados en un experimento podría tener efectos diversos en la aplicación de una técnica de selección de casos de prueba por parte de un sujeto en un experimento.

Calikli y otros realizaron un estudio para investigar la relación entre el sesgo confirmatorio, medido con técnicas de psicología cognitiva, el nivel de experiencia y formación, y la calidad del código generado. Concluyeron que no existe una relación significativa entre la experiencia en desarrollo o testing de software y habilidades para evaluar hipótesis y encontraron evidencia que sugiere que el sesgo confirmatorio disminuye con el uso continuo del razonamiento abstracto y del pensamiento crítico. Encontraron a su vez una alta correlación entre el nivel del sesgo confirmatorio de los sujetos y la cantidad de defectos del código producido por ellos, lo que podría estar asociado al papel de las pruebas unitarias en el desarrollo de software. Pero, dado el reducido número de sujetos, sus resultados no son significativos desde el punto de vista estadístico [CAL10, C AL12].

Causevic y otros realizaron un experimento piloto para comparar la calidad de los casos de prueba seleccionados en el marco de un proceso en el que estos se seleccionan una vez que el software ya está construido respecto al enfoque de Test Driven Development (TDD) en que los casos de prueba se seleccionan de forma previa a la construcción del software. Encontraron que los casos de prueba de TDD tienen mayor número de casos que prueban con entradas válidas que inválidas [CAU12], lo que es un indicio de que el sesgo positivo podría estar afectado por el contexto en el que se identifican los casos de prueba.

Los casos de prueba que representan condiciones de entrada inválidas o no esperadas suelen tener una mayor tasa de detección de errores que las que corresponden a entradas válidas o esperadas [MYE04],

por lo que el impacto del sesgo positivo sobre la detección de defectos podría ser aún mayor que sobre la identificación de casos de prueba.

4.2.4 Revisión de los resultados de las pruebas

Los resultados de cada prueba de software deben ser revisados con sumo cuidado, ya que con frecuencia quienes prueban software no llegan a observar fallas puestas en evidencia por los resultados de las pruebas [MYE04, BAS87].

En las dos replicaciones del experimento de Kamsties y Lott los experimentadores controlaron de forma independiente a los sujetos si los casos de prueba seleccionados por los sujetos permitían revelar fallas y encontraron que muchos sujetos no pudieron observar fallas que sus casos de prueba revelaban. En la primera replicación el porcentaje de fallas observadas por los sujetos respecto a las reveladas por los casos de prueba tuvo un promedio del 80%, con un mínimo de 40% y un máximo de 100% y en la segunda replicación estos valores fueron respectivamente de 76%, 25% y 100% [KAM95a]. En el estudio de Basili y Selby no fueron reportados un promedio del 19% de los defectos revelables a partir de los casos de prueba seleccionados por los sujetos, con una desviación estándar del 17,3% [BAS87].

La dificultad de los sujetos para observar fallas revelables podría constituir una amenaza a la validez de construcción de experimentos en los que no se haga la distinción entre lo observado por los sujetos y los defectos que podrían ser revelados por los casos de prueba.

4.2.5 Estudiantes como sujetos y motivación

Berander presenta un estudio empírico del desempeño de estudiantes en el marco de proyectos y llega a la conclusión de que los estudiantes en este marco tienen un desempeño más parecido al de los profesionales que en el marco de una clase [BER04].

Uno se puede preguntar entonces si los profesionales fuera del marco de un proyecto, por ejemplo en el marco de un ejercicio de laboratorio, no tendrán un desempeño más parecido al de estudiantes en el marco de un curso.

Host y otros afirman que el hecho de que múltiples experimentos controlados en ingeniería de software tengan resultados contradictorios está indicando que la comunidad de investigación no ha llegado a identificar las variables explicativas relevantes de forma satisfactoria, no se ha capturado el contexto de forma adecuada. Propone un marco para clasificar el contexto de los experimentos que considera los incentivos (más bien el contexto de motivación) y la experiencia de los sujetos. Respecto a la motivación, a partir de los resultados previamente mencionados de Berander, propone una clasificación de la motivación en los estudios empíricos que establezca si se da en el marco de un proyecto o no [HOS05].

4.2.6 Otros factores del contexto

Juristo et al. presentaron un estudio de un conjunto de ocho replicaciones de un experimento, en el que distinguen la técnica y su uso por los sujetos, en todos los casos estudiantes [JUR11, JUR12]. Las replicaciones no fueron idénticas, sino que diversos factores de contexto fueron variando. Al presentar los resultados y a partir de su análisis junto con el del contexto, formulan las proposiciones siguientes:

a) La falta de experiencia en programación disminuye la efectividad de cobertura de decisión.

b) El conocimiento de la técnica podría afectar su efectividad.

c) Las técnicas podrían ser menos efectivas si los sujetos no están motivados.

d) Las técnicas son más efectivas si se trabaja en parejas.

e) El trabajo bajo presión no parece influir en la efectividad de cobertura de decisión.

f) El cansancio no parece influir en la efectividad de la técnica.

g) Tener información previa del programa no parece influir en la efectividad de la técnica.

Entre los hallazgos de ese estudio está que las técnicas de partición en clases de equivalencia y branch testing son más efectivas cuando diferentes sujetos aplican la misma técnica que cuando el mismo sujeto aplica diferentes técnicas, lo que es un indicio de que las características de cada sujeto podrían tener más incidencia en la identificación de defectos que la aplicación de las técnicas de selección de casos de prueba.

4.2.7 Incidencia de la experiencia en la identificación de defectos

Estudios empíricos muestran que la cantidad de años de experiencia profesional está entre los factores que más afectan la efectividad en la identificación de defectos [BAS87].

Beer y Ramler presentaron el estudio de tres casos de proyectos exitosos en tres dominios de aplicación diferentes y en los que identifican que el testing basado en la experiencia jugó un papel destacado. En los tres proyectos la experiencia previa en testing del dominio o en versiones anteriores del mismo proyecto tuvo un papel destacado en el desarrollo de casos de prueba. Existió asimismo una preocupación por el desarrollo en los equipos de test del conocimiento del dominio de aplicación así como del conocimiento y experiencia referidos a cada proyecto.

En uno de los proyectos, se constató que entre el 10 y el 20% de los casos de prueba fueron generados a partir de la experiencia y estos casos permitieron detectar el 50% del total de defectos detectados [BEE08]. Esto indica que en promedio cada caso de prueba generado a partir de la experiencia tuvo una efectividad en cuanto a cantidad de defectos detectados como la de 4 casos de prueba de otro tipo.

Armour afirma: “Encontré que los buenos testers tienen “olfato” para hacer testing. Experimentan una especie de intuición que les dice qué y cómo probar. [...] Buenos testers son capaces de emplear un proceso de razonamiento intuitivo que no es fácil de explicar o codificar” [ARM05].

Kaner afirma que un tester experiente que conoce el producto y pasó por uno o dos ciclos de liberación es capaz de probar con mucha mayor efectividad -incluso sin instrucción alguna- comparado a un tester inexperiente que recibe un conjunto de instrucciones escritas acerca de como probar el producto [KAN02].

Resulta atinada la recomendación de Kitchenham y otros respecto a que en caso de que los sujetos que participen en un experimento sean estudiantes, se excluya a aquellos que cuenten con experiencia previa en la industria de forma de asegurar la homogeneidad de los sujetos [KIT02]. En caso de no hacerlo se podría afectar la validez interna del estudio.

4.2.8 Impacto en los testers de información de cobertura y su impacto dependiendo de técnica y experiencia

Berner y otros analizan el impacto del uso de información de cobertura de las pruebas en un proyecto en particular. La herramienta fue introducida con el objetivo de detectar áreas adicionales a probar pero sin fijar una meta a cumplir respecto a cobertura. Antes de la introducción de la herramienta de análisis de cobertura, esta fue creciendo lentamente hasta llegar a alrededor del 70% de las instrucciones. La interpretación que hacen es que los testers tenían confianza en la calidad de las pruebas realizadas y no se les pasaba por la mente que tuvieran que realizar pruebas adicionales. A partir de la introducción de la herramienta de análisis de cobertura, esta saltó rápidamente a más del 80%.

El impacto de la introducción de la herramienta fue claramente distinto dependiendo de la experiencia del personal. En el caso del personal senior, la cobertura saltó rápidamente porque identificaron áreas no probadas (manejo de excepciones, sincronización y robustez). Una vez identificados estos tests adicionales, los desarrolladores los incorporaron naturalmente en los nuevos tests. En el caso del personal junior el impacto de la introducción de la herramienta fue mucho mayor. En este caso el nivel de cobertura previo a la introducción de la herramienta era mucho menor y a partir de la introducción de la herramienta el nivel de cobertura se incrementó significativamente, llegando casi a niveles similares a los del personal senior. A diferencia de lo sucedido con los desarrolladores senior, muchos de los tests agregados correspondían a casos particulares de la funcionalidad central del producto. La introducción de la herramienta tuvo en el caso del personal junior un impacto muy importante en acercar la calidad de las pruebas a las del personal senior [BERN07].

Lawrence y otros presentan un experimento que evalúa el impacto de la visualización de la información de cobertura sobre el desempeño de los testers. El criterio de aceptación utilizado fue cobertura de bloques (sentencias y bifurcaciones). De este experimento se concluye que la información de cobertura no generó impacto alguno en cuanto a encarar mejores estrategias de testing, ni plantearon más casos de prueba ni encontraron más defectos que el grupo de control. Sin embargo la visualización de la información de cobertura llevó a los testers a sobreestimar la efectividad de sus pruebas tomando como referencia al grupo de control y redujo la variabilidad en la cantidad de casos de prueba generados al cambiar el estándar contra el cual evaluar la efectividad de sus pruebas [LAW05].

Rothermel y otros presentan un experimento análogo al anterior pero aplicando como criterio de adecuación define-use (d-u). Con este método más exigente como referencia, los testers generaron mejores casos de prueba en cuanto al cumplimiento del criterio d-u y no sobreestimaron la efectividad de sus pruebas, lo que sí hizo el grupo de control [ROT00].

Estos dos experimentos con resultados aparentemente contradictorios muestran que la información de cobertura tiene efectos sobre el conjunto de casos de prueba que generan los testers. Una posible explicación de los resultados es que con criterios de aceptación poco exigentes la información de cobertura tiende a infundir una confianza injustificada en los testers, mientras que si la referencia es un criterio de cobertura muy exigente, sucede exactamente lo contrario.

4.3 Factores que inciden en la detección de defectos por parte de sujetos

La figura 1 brinda una visión unificada de los factores que inciden en la detección de defectos -representados por rectángulos, y de sus relaciones -representadas por flechas con una referencia numérica. El diagrama se puede considerar como un esquema del proceso de aplicación de una técnica de selección de casos de prueba. En el diagrama hay dos rectángulos con bordes remarcados: Técnica de Selección de Casos de Prueba y Defectos Detectados. El primero marca el comienzo del proceso y el segundo marca el final. El proceso para obtener los Defectos Detectados involucra las actividades: Aplicación de la Técnica y Revisión de Resultados que aparecen sombreados en el diagrama. Aplicación de la Técnica genera dos resultados: CCP y Defectos Revelados. Se entiende por "defectos revelados" aquellos que pueden ser observados a partir de la salida de los casos de prueba. A continuación el proceso pasa por la Revisión de Resultados que tiene como salida los "Defectos detectados" que son aquellos efectivamente detectados por los sujetos. La aplicación de la técnica depende de las características de los programas, de los sujetos y del contexto de aplicación de la técnica.

Entre las características de los programas están su Tamaño, Complejidad, Tecnología, Dominio de aplicación, la Relación con la Especificación y las Características de los Defectos (Cantidad, Distribución, Tipos). Entre las características de los sujetos están su Experiencia (Profesional, en el Lenguaje, en el Dominio de Aplicación), Habilidad para Generar Casos que Revelen Defectos y Sesgo Positivo.

En la Aplicación de la Técnica inciden las Características del Programa, las Características de los Sujetos y el Contexto de Aplicación de la Técnica. Entre los factores que forman parte de este contexto están: si se trata de un Proyecto o Curso, los Incentivos para los sujetos, la Efectividad del Entrenamiento y la Información en el proceso de testing referida a la Cobertura Lograda y a los Defectos Detectados.

La Revisión de los Resultados tiene como salida la Cantidad de Defectos Detectados, la que va a estar influenciada por el Contexto de Aplicación de la Técnica, las Características de los Sujetos y la Probabilidad de Detectar una Falla. Esta última depende de las Características de las Fallas, las que están asociadas a las Características de los Defectos. La Información en el Proceso (parte del Contexto de Aplicación de la Técnica) incide de forma distinta dependiendo de la Técnica de Selección de Casos de Prueba y de la Experiencia de los sujetos. El grado de Relación con la Especificación del programa tiene distinta incidencia según cual sea la Técnica de Selección de Casos de Prueba que se considere, en particular es distinta para técnicas de Caja Negra y de Caja Blanca.

4.4 La cantidad de defectos detectados como indicador de la efectividad de la técnica

El diagrama muestra con claridad que existen distintos factores que pueden incidir en la forma en que los sujetos aplican una técnica de selección de casos de prueba para obtener un CCP y en la detección de los defectos revelados por cada CCP. Por esta razón podría resultar conveniente distinguir en los experimentos destinados a evaluar la efectividad de técnicas de selección de casos de prueba la cantidad de defectos "revelados" por los CCP de la cantidad de defectos "detectados" por los sujetos. Asimismo, si el objetivo del experimento es sacar conclusiones respecto a la efectividad de las distintas técnicas de selección de casos de prueba, correspondería evaluar si la aplicación de la técnica y la revisión de los resultados resultaron o no adecuadas.

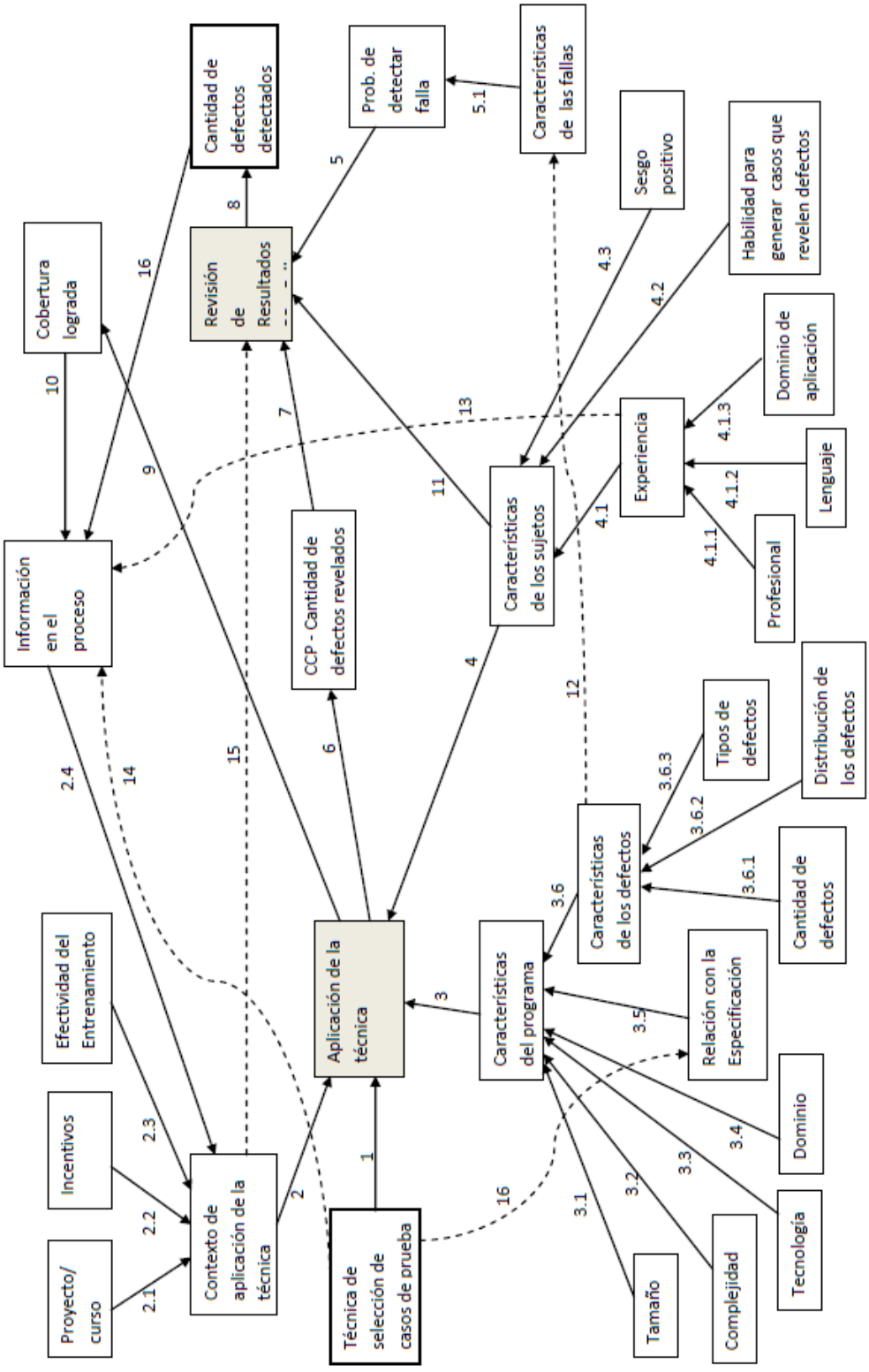

Figura 1: Factores que inciden en la detección de defectos

4.5 Evaluación de la satisfacción de la técnica en experimentos

Al evaluar una técnica de selección de casos de prueba mediante un experimento del tipo descrito al final del apartado 3.1 en el que sujetos aplican distintas técnicas de selección de casos de prueba para distintos programas, se debe asegurar que al momento de terminar el experimento, la técnica haya sido aplicada de forma completa y apropiada. Si hay un criterio de cobertura, se debe asegurar el 100% de cobertura para que los resultados de costo y efectividad resulten útiles [BRI04]. Esto seguramente ya lo tenían claro Basili y Selby en 1987, en que compararon mediante una serie de experimentos la efectividad de tres técnicas de verificación: lectura de código por "step-wise abstraction", testing estructural con cobertura de código de 100% y partición en clases de equivalencia con análisis de valores de borde. Para este último también midieron la cobertura de código. y para la técnica de cobertura de código, el experimento se daba por terminado de forma automática cuando el sujeto llegaba a una cobertura del 100% o muy próxima a este valor.

Dadas las dificultades que presenta el entrenamiento en las técnicas de partición en clases de equivalencia, la ausencia de evaluación del grado de satisfacción de la técnica en los casos de técnicas de caja negra constituye una amenaza a la validez de construcción de los experimentos.

Se puede considerar que en los experimentos en los que no se controla el grado de satisfacción de una técnica de selección de casos de prueba en lugar de evaluar la efectividad de la técnica se está evaluando la efectividad de su aplicación por sujetos con ciertas capacidades y entrenamiento dados. Para los experimentos cuyo objetivo es evaluar la efectividad de la técnica, no controlar el grado de satisfacción de la técnica constituye una amenaza a la validez externa.

Artículo	Control de satisfacción de la técnica
Basili y Selby [BAS87]	Por construcción controla para la técnica estructural (cobertura de sentencias), no para la funcional (partición en clases de equivalencia y análisis de valores de borde). Para la técnica funcional controla cobertura de sentencias.
Kamsties, E., and Lott, C. M [KAM95, KAM95a]	En lectura como el de Basili. En funcional no hay control de cobertura funcional, en estructural trata de lograr 100% y después acceden a información de cobertura para completar. Tanto para las pruebas con técnica funcional como estructural hay evaluación de la cobertura estructural ((bifurcación, condición múltiple, bucle y operador relacional)
Myers, G. J. [MYE78]	Compara efectividad de prueba caja negra, caja blanca e inspección. Para ningún tipo se especificó técnica en particular. El criterio para parar era que los que probaran quedaran conformes.
Wood, M. et al [WOO97]	Similar al de Basili Selby, replica experimento de Lott. Partición de equivalencia con análisis de borde, *branch coverage* y *code Reading by step-wise refinement*, Hay control de cobertura estructural para técnica estructural. No hay control de cobertura para técnica funcional.

Cuadro 1: Control de satisfacción de la técnica en experimentos

En los experimentos referidos por Juristo y otros en la revisión de 25 años de experimentos [JUR04] hay 4 casos en los que sujetos aplican técnicas de selección de casos de prueba. En tres de ellos (el de Basili y Selby, el de Kamsties y Lott y en el de Woods, Roper y Brooks), y solo para cobertura de pruebas de caja blanca, hay una evaluación del grado de satisfacción de la técnica. Para las técnicas de selección de casos de prueba de tipo caja negra, en ninguno de los casos hay una evaluación del grado de satisfacción de la técnica.

En los experimentos de Basili y Selby, para las pruebas de caja negra, si bien no hay una evaluación del grado de satisfacción de la técnica, sí hay, como un indicador de la calidad de los casos de prueba identificados por los sujetos, una evaluación del cubrimiento de sentencias, el que llegó en promedio al 97%, el mismo valor que el obtenido en las pruebas de caja blanca [BAS87]. En los experimentos de Kamsties y Lott, para las pruebas funcionales (caja negra) también hay una evaluación de la cobertura estructural (bifurcación, condición múltiple, bucle y operador relacional) e informan que les ayudó para evaluar cuánto del programa fue ejercitado en las pruebas, pero no agregan más detalles [KAM95a]. En el cuadro 1 se detallan las referencias y el control de satisfacción de la técnica en cada caso.

4.6 Satisfacción de técnicas funcionales

A diferencia de lo que sucede con las técnicas de selección estructurales, en que la propia definición de cada técnica aplicada al código de un programa implica la definición de un mecanismo para evaluar el grado de satisfacción de la misma, para las técnicas funcionales la definición de la técnica está basada en reglas heurísticas referidas a la especificación del programa, no a su código, y resulta difícil determinar si esas reglas son satisfechas [RIC89].

En tres de los experimentos mencionados en la sección previa, la técnica funcional de selección de casos de prueba aplicada consistió en partición en clases de equivalencia con análisis de valores de borde. En el artículo de Basili y Selby la única referencia que se propone para la técnica es la de Myers [MYE04] pero en su primera edición de 1979. En el artículo de Kamsties las referencias para esa técnica son el libro de Myers y el artículo de Howden [How80] y en el artículo de Wood las referencias son el libro de Pressman [PRE94] y el de Roper [ROP93]. La presentación de Pressman está basada en la de Myers. Juristo et al. en los artículos [JUR11, JUR12] indican que la técnica funcional de selección evaluada fue Partición en Clases de Equivalencia (PCE) y la referencia que ponen para PCE es también el libro de Myers. Corresponde notar que en los artículos de Juristo et al. no se menciona el Análisis de Valores de Borde.

La identificación de las clases de equivalencia en la técnica de partición en clases de equivalencia supone seguir los pasos siguientes :

a) Identificar las condiciones de entrada.
b) En caso de que para una condición de entrada se pueda suponer que el programa no maneja los elementos de una clase de manera idéntica, subdividir la condición en condiciones de entrada más precisas [MYE04].

El paso (b) puede dar lugar a dos interpretaciones distintas:

a) se parte de la hipótesis de que el programa se comporta de acuerdo a lo especificado y es en esa hipótesis que se identifican elementos dentro de la clase que el programa podría no manejar de forma idéntica o
b) se considera que el programa pudiera comportarse o no de acuerdo a lo especificado y es en esta otra hipótesis que se identifican los elementos dentro de la clase que el programa podría no manejar de forma idéntica.

Con la primera interpretación, las clases de equivalencia se van a derivar directamente de la especificación. La segunda interpretación deja el campo abierto a la conjetura de errores.

La conjetura de errores es una técnica que se basa en las habilidades y experiencia previa de quien está probando en las tecnologías en uso y en el dominio de aplicación. A menudo se notó que algunas personas parecen tener de forma natural una especial habilidad para el testing de software y sin aplicar ninguna metodología en particular tienen una rara capacidad para olfatear defectos. Una explicación es que estas personas están practicando de forma inconsciente la técnica de conjetura de defectos a partir de su intuición y experiencia [MYE04].

Esta técnica, de gran importancia en su aplicación práctica, podría estar contaminando la aplicación de cualquiera de las otras técnicas funcionales de selección de casos de prueba.

En el caso de la técnica de partición en clases de equivalencia el tema es un poco más complejo debido a que habría que precisar el alcance dado a la técnica por parte de quienes llevaron a cabo cada experimento y en especial durante el entrenamiento a los sujetos.

Para el caso de las técnicas funcionales, evaluar el grado de satisfacción de la técnica involucra verificar:

a) que se generaron casos de prueba de acuerdo a la técnica funcional en particular y
b) que no se generaron casos que correspondan a otra técnica, por ejemplo originados en la conjetura de errores.

Capítulo 5: Criterio de aceptación aplicable a Caja Negra

La mayoría de los criterios de aceptación hacen referencia al código del programa por lo que no resultan del todo adecuados para técnicas de selección de Caja Negra y no resultan aplicables en aquellos casos en los que el código no está disponible.

En 1983 Weyuker propuso la idea de que, dados una especificación S, un programa P a probar y un CCP T para P, si a partir de T fuera posible generar de forma automática S y P, podríamos tener confianza de que T resulta adecuado para P y S [WEY83]. En el análisis de la propuesta queda claro que este resulta poco factible de ser aplicable en la práctica por lo que propone dos alternativas posibles para que el enfoque resulte factible requiriendo que tan solo se infiera el programa por un lado o solo la especificación, por otro [WEY83]. La segunda alternativa resulta aplicable a enfoques de caja negra. A partir de estas ideas se generaron distintos desarrollos tanto teóricos como prácticos relacionando pruebas de software con inferencia inductiva.

En las distintas secciones del capítulo se trabaja a partir de estos desarrollos para terminar definiendo un criterio de aceptación aplicable a CCP generados utilizando una técnica de Caja Negra.

El único otro enfoque del que tenemos conocimiento relacionado con criterios de aceptación aplicables a enfoques de Caja Negra que no requieran una especificación formal está orientado a la prueba de tipos especiales de componentes que presentan requerimientos no triviales respecto al ordenamiento en que sus métodos o procedimientos debieran ser invocados. Este criterio está basado en una abstracción de la semántica asociada al comportamiento esperado [CZE13].

5.1 Relación entre pruebas de software e inferencia inductiva

El término “inferencia inductiva” (a veces también mencionado como “programación inductiva”) refiere al proceso de establecer la hipótesis de una regla general a partir de ejemplos. En la figura 2 se presenta un esquema del proceso correspondiente.

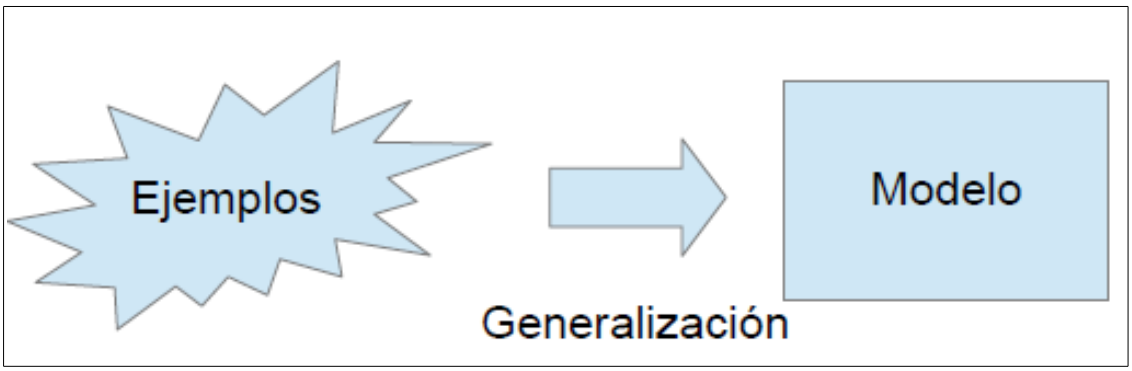

Figura 2: Esquema del proceso de Inferencia Inductiva

Ejemplo 1

Dada la secuencia:

011, 00000011, 00111, 00001, 0011

se podría conjeturar que los elementos de la secuencia siguen la regla "cualquier número de 0s seguido de cualquier número de 1s". Si la fuente de elementos de la secuencia sigue generando valores, los nuevos elementos podrían o no cumplir con esa regla. La inducción es una forma de razonamiento que permite pasar de lo particular a lo general [ANG83].

Ejemplo 2

Supongamos que tenemos secuencias de ternas para las variables x, y, z con los valores (2,3,5), (4,7,11), (5,1,6). Para esas ternas, los dos modelos siguientes resultarían adecuados.

Modelo 1:
z=x+y

Modelo 2:
si x=2, y=3 entonces z=5
si x=4, y=7 entonces z=11
si x=5, y=1 entonces z=6
si no z=1

Sin embargo el modelo 1 podemos considerarlo "más simple" que el modelo 2. Cabe también notar que el valor asignado a z en el el modelo 2 carece de justificación. En el proceso de inducción es posible generar diversos modelos y se han propuesto diversos criterios para elegir entre ellos. Uno particularmente importante es la "simplicidad" del mismo [ANG83].

La inferencia inductiva de un programa puede ser considerada como la inversa del proceso de probar un programa que consiste en tomar muestras del comportamiento de un programa [ZHU92]. El proceso de testing parte de una especificación y de un programa y busca parejas entrada/salida que caractericen cada aspecto tanto de los cálculos previstos como reales. La inferencia inductiva parte de parejas de entrada/salida y deriva el programa más simple consistente con el comportamiento previsto [WEY83].

En este paralelismo aparece con nitidez la necesidad de prestar atención a las condiciones de borde. Para poder inferir un programa es necesario identificar en qué lugar comienza y termina cada tipo de cálculo. En el caso de testing, las condiciones de borde deben ser incluidas ya que son especialmente propensas a contener errores [WEY83].

Por otra parte, el proceso de testing puede ser visto también como un proceso de inferencia inductiva (ver Figura 3) en el que quien prueba un programa trata de deducir las propiedades del mismo para todo el espacio de entrada a partir de la observación del comportamiento en un conjunto finito de casos [ZHU92, ZHU96A].

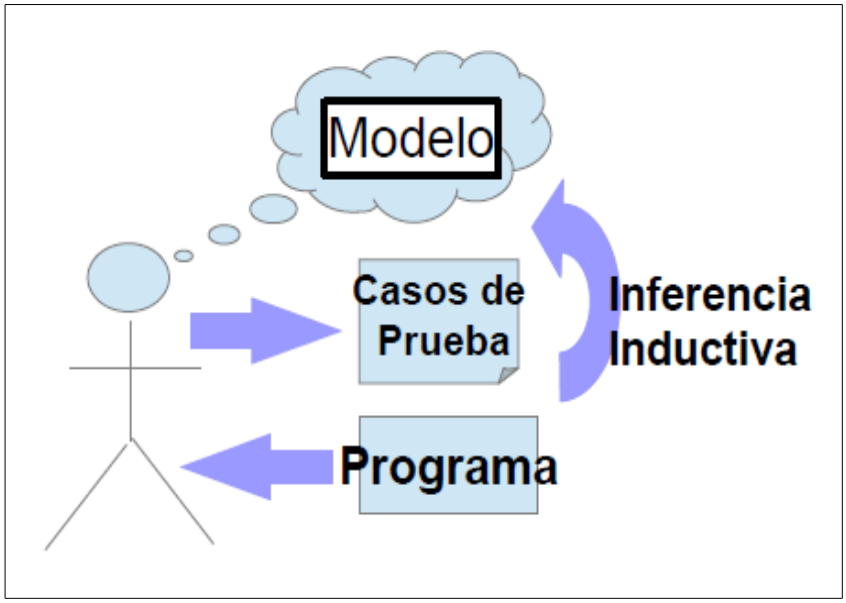

Figura 3: Proceso de testing visto como inferencia inductiva

Utilizando este enfoque, Zhu mostró que la convergencia del proceso de inferencia inductiva implícito en el proceso de testing es una condición para que el programa adecuadamente puesto a prueba sea correcto. Adicionalmente, también mostró que la convergencia depende de la complejidad del programa, por lo que cuanto más complejo sea el programa, mayor deberá ser la cantidad de casos de prueba a utilizar [ZHU96A]. Para hacer esa demostración define un dispositivo conceptual M capaz de realizar inferencia inductiva de un programa P a partir de un conjunto de casos de prueba T y define un criterio de aceptación que denomina de Inferencia Inductiva en M según el cual T satisface el criterio si M permite inferir inductivamente P a partir de T.

Impone a M dos características:

a) Para cualquier secuencia de instancias de entrada, M debe inferir un programa P' diferente de P solo si la nueva instancia es incompatible con P. A esta propiedad de M la denomina “conservativa”.

b) M debe generar el programa más simple consistente con la entrada y a esta propiedad la denomina “hipótesis más simple” [ZHU96A].

Estas propiedades frecuentemente están presentes en sistemas de software que permiten realizar inferencia inductiva de un programa a partir de un conjunto de parejas de entrada/salida [ZHU96A].

Este criterio puede ser aplicado como técnica para generar casos de prueba. Dado el programa P y un conjunto parcial de casos de prueba Tp, a partir de estos se genera el programa P’ consistente con P para esos datos de entrada. Comparando P y P’ se identifica un caso de prueba que distinga el comportamiento de P del de P’ y se agrega al conjunto de casos de prueba. Esto se repite hasta que no sea posible distinguir el comportamiento de P y P’ [BER96].

Si el programa es muy complejo puede ser muy grande la cantidad de casos de prueba necesarios para generar un programa que tenga el mismo comportamiento que el programa a probar. En razón de ello Walkinshaw propuso una técnica de selección de casos de prueba menos exigente relajando el requeri-

miento de que el comportamiento de P y P' sea idéntico [WAL10]. Esta propuesta está basada en la aplicación del marco PAC (Probably Aproximately Correct) definido en teoría de aprendizaje automático. Este marco plantea que el aprendizaje a partir de un conjunto de ejemplos puede ser aproximado (Aproximately Correct) y con determinada probabilidad (Probably) [VAL84].

Fraser y Walkinshaw proponen un mecanismo complejo para generar un modelo de un programa P. Este consta de varios pasos:

a) Generar de forma automática casos de datos de entrada E con enfoque de caja blanca para P.

b) Probar P con los datos de entrada E con lo que se obtienen las salidas S para esos datos de entrada.

c) Construir un CCP T con datos de entrada y salida conformados por los elementos correspondientes de E y S.

d) Generar a partir de T mediante inferencia inductiva un programa P', modelo del programa P.

e) Poner a prueba P' con un nuevo CCP T' generado de forma independiente

f) Evaluar el modelo P' de acuerdo al PAC de Valiant [FRA12].

5.2 Criterio de aceptación basado en inferencia inductiva

El criterio de aceptación de Inferencia Inductiva propuesto por Zhu lo fue a los efectos de desarrollar la teoría. Solo hace referencia al dispositivo de inferencia inductiva M, al programa P y al conjunto de casos de prueba T y no toma en cuenta la especificación del programa. Por otro lado resulta en la práctica muy fuerte la exigencia de que se genere exactamente el programa P. Si omitimos la referencia al dispositivo de inferencia inductiva y relajamos la exigencia de que el dispositivo genere el mismo programa P, podemos reformular el criterio como sigue.

Criterio 1: Dado un programa P y un conjunto de casos de prueba T, diremos que el conjunto de casos de prueba es adecuado si a partir de ese conjunto de casos de prueba se puede generar mediante inferencia inductiva un programa P' que sea equivalente a P, esto es que genere los mismos resultados para todos los datos de entrada posibles.

Este criterio de aceptación solo toma en cuenta el comportamiento del programa y no su código, por lo que podría aplicarse a casos de prueba generados utilizando un enfoque de caja negra,blanca o mixto.

En la práctica, asegurar la equivalencia entre los programas P y P' puede resultar una tarea sumamente compleja. Para evitar este problema, se puede relajar el criterio en este aspecto siguiendo las ideas de Walkinshaw..

Criterio 1r: Dado un programa P y un conjunto de casos de prueba T, diremos que el conjunto de casos de prueba es adecuado si a partir de ese conjunto de casos de prueba se puede generar mediante inferencia inductiva un programa P' que sea aproximadamente equivalente a P con error ε y nivel de confianza δ, esto es que con ese nivel de confianza genere los mismos resultados para todos los datos de entrada posibles con probabilidad 1- ε.

La generación de un conjunto de casos de prueba con enfoque de caja negra se hace a partir de la especificación y no del código. El criterio 1 está basado en el comportamiento del código, no en la especifica-

ción. Esto no plantea inconvenientes si el comportamiento del código es consistente con la especificación, pero no hay certeza de que efectivamente lo sea. Para resolver esta limitación proponemos el Criterio 2.

Criterio 2: Dada una especificación S y un conjunto de casos de prueba T, diremos que el conjunto de casos de prueba es adecuado si es posible generar un programa P' mediante inferencia inductiva del conjunto de casos de prueba T tal que P' sea correcto respecto a S.

Este criterio se puede aplicar por más que no esté construido el programa correspondiente a la especificación S. Aplicado a enfoques de caja negra exige que se tomen en cuenta las condiciones de borde ya que la inferencia inductiva requiere contar con la información de dónde comienza y termina cada tipo de cálculo. Por esta razón puede tomarse como criterio de suficiencia para la técnica de partición en clases de equivalencia con condiciones de borde. Para la técnica de partición en clases de equivalencia sin considerar condiciones de borde el criterio resulta demasiado exigente.

La técnica de partición en clases de equivalencia tiene importancia desde el punto de vista teórico y conceptual pero por sí sola no resulta útil en la práctica ya que las condiciones de borde juegan un rol fundamental en la detección de defectos [WEY83].

Para tomar en cuenta las características de P, se puede formular un criterio más exigente.

Criterio 3: Dado un programa P, una especificación S del programa y un conjunto de casos de prueba T, diremos que el conjunto de casos de prueba es suficiente si es posible generar un programa P' mediante inferencia inductiva del conjunto de casos de prueba T tal que P' sea correcto respecto a S y P' sea equivalente a P, esto es que genere los mismos resultados para todos los datos de entrada posibles.

El criterio 3 solo puede ser satisfecho en el caso de que el programa P sea correcto en el sentido de que cumpla con la especificación S. Este criterio corresponde a la propuesta original de Weyuker [WEY83) ligeramente relajada ya que en lugar de exigir generar S requiere generar un programa correcto respecto a S y en lugar de generar P basta con generar un programa equivalente a P .

La comparación entre los criterios 1 y 2 permite arrojar luz sobre las diferencias entre los enfoques de caja blanca y caja negra para la generación de casos de prueba. El criterio 1 toma en cuenta las características específicas de la implementación del programa, no así el criterio 2. Si P contiene construcciones especiales que no se derivan de la especificación S, encontrar eventuales defectos asociados a esas construcciones podría resultar más difícil si los casos de prueba se generan con un enfoque de caja negra que si se generan mediante un enfoque de caja blanca, lo que ilustra el impacto de la relación 3.6 (características de los defectos- características del programa) del diagrama de la Figura 1.

5.3 Criterio IIE

De todos los criterios presentados en la sección anterior, el más adecuado para aplicar a la evaluación de conjuntos de casos de prueba generados utilizando técnicas de caja negra es el criterio 2, al que vamos a denominar Inferencia Inductiva de la Especificación (IIE).

Criterio IIE: Dada una especificación S y un conjunto de casos de prueba T, diremos que el conjunto de casos de prueba es adecuado si es posible generar un programa P' mediante inferencia inductiva del conjunto de casos de prueba T tal que P' sea correcto respecto a S.

Este criterio corresponde a una de las alternativas consideradas por Weyuker para volver factible su enfoque de inferir tanto el programa como la especificación [WEY83], ligeramente relajada ya que no se requiere inferir la especificación sino que basta con inferir un programa correcto respecto a la especificación.

5.4 Herramientas para inferencia inductiva

Existen actualmente muchas herramientas que permiten inducir un programa a partir de un conjunto de casos de prueba, pero varias de estas herramientas solo son capaces de inducir programas que cumplen con ciertas restricciones. Kitzelman hace una revisión de los distintos enfoques y tipos de herramientas [KITZ10] y los clasifica en tres categorías:

a) Enfoque analítico funcional
b) Programación Lógica Inductiva
c) Enfoques basados en generación funcional y prueba.

El enfoque analítico funcional está basado en que bajo ciertas restricciones relacionadas con las primitivas a considerar, esquema del programa y elección de casos de entrada-salida, resulta posible generar un programa recursivo en LISP sin realizar búsquedas en el espacio de los programas. Esto permite generar un primer candidato de programa y a partir de él realizar búsquedas en el espacio de programas. Los sistemas IGOR1 e IGOR2 pertenecen a esta categoría.

El enfoque de Programación Lógica Inductiva constituye una rama del aprendizaje automático. Los conceptos se aprenden a partir de ejemplos y contra-ejemplos y utiliza lógica de primer orden para expresar las hipótesis, ejemplos y el conocimiento necesario. Los sistemas FOIL, GOLEM, PROGOL, CRUSTACEAN, CLAM, TIM, MRI y SMART son ejemplos de sistemas basados en este enfoque y son capaces de aprender conceptos no-recursivos, pero tienen problemas para aprender programas recursivos. Algunos de estos sistemas permiten incorporar conocimiento adicional del dominio de aplicación, además de los ejemplos.

El enfoque de generación funcional y prueba se caracteriza porque los ejemplos de entrada-salida no se utilizan en la construcción de los programas, sino que se utilizan para evaluar los programas generados. La generación de los programas se hace por ejemplo mediante Programación Genética(PG) [KOZ92]. Los sistemas basados en PG mantienen *poblaciones* de soluciones candidatas y generan nuevas mediante métodos estocásticos tales como *reproducción*, *mutación*, *recombinación (cross-over)* y *selección* para maximizar la *aptitud* (fitness). Las búsquedas evolutivas pueden resultar útiles cuando el espacio del problema es demasiado amplio para realizar una búsqueda exhaustiva y cuando a la vez se sabe poco o nada acerca de las características de la aptitud por lo que no se pueden construir heurísticas apropiadas. Ejemplos de este enfoque son los sistemas POLYGP y ADATE.

Este último ha sido considerado por Kitzelman como el más poderoso sistema de programación inductiva, especialmente diseñado para evolucionar programas recursivos, para lo que aplica sofisticados operadores de transformación, estrategias de búsqueda y funciones de evaluación [KITZ10].

5.5 Sistema ADATE

ADATE es un sistema evolucionario que mantiene una población de programas (reino en términos de ADATE) y realiza una búsqueda guiada por una función de aptitud.

La colección de programas que genera y mantiene ADATE comienza con un único individuo y luego se expande con el progreso de la evolución.

El sistema ADATE ejecuta sin fin, a menos que el usuario termine el proceso que consiste en los pasos siguientes:

1. Insertar un individuo en el reino.
2. Elegir un individuo del reino para su transformación y definir un límite de costo (esfuerzo de la transformación).
3. Generar un nuevo individuo mediante una transformación compuesta.
4. Insertar el nuevo individuo en el reino si va a formar parte de una nueva familia o calza en una familia ya existente.
5. Repetir los pasos 3 y 4 un número de veces de acuerdo al límite de costo establecido para la expansión.
6. Repetir los pasos 2 a 5 indefinidamente.

Los individuos que ADATE mantiene en el reino se ubican en cuadrículas de acuerdo a sus complejidades sintáctica y de tiempo. La complejidad de tiempo puede ser vista como el tiempo que el individuo utiliza para ejecutar los ejemplos de entrenamiento y la complejidad sintáctica como su tamaño en bits.

Los individuos en ADATE se clasifican de acuerdo a una taxonomía análoga a la utilizada en biología, compuesta por: Reino, Filo, Clase, Orden, Familia, Género, Especie.

Un *individuo* es la clasificación más baja y representa un programa en ADATE.

Un *individuo base* es el más pequeño encontrado hasta el momento dados los tamaños sintáctico y de tiempo.

Un *individuo output* es un individuo que tiene el mismo valor de evaluación de la función de aptitud pero es semánticamente diferente del individuo base y de todos los otros individuos output de la familia.

Un *individuo incrustado* (embedding individual) es un individuo con funciones auxiliares con argumentos extra respecto a las funciones auxiliares del individuo base.

Una *especie* es representada por un único individuo a partir del cual se genera mediante transformaciones compuestas.

ADATE lleva a cabo transformaciones de los individuos para obtener nuevos individuos. Cualquier individuo puede ser rastreado hacia atrás a través de una serie de transformaciones hasta el programa inicial. ADATE tiene seis formas básicas de transformación, denominadas transformaciones atómicas:

1. R (Reemplazo): Cambia parte del individuo con nuevas expresiones.
2. REQ (Reemplazo que asegura que el individuo no es peor).
3. ABSTR (Abstracción): Toma una expresión en el individuo y la pone en una función en un bloque let...in y reemplaza la expresión con una llamada a esa función.

4. CASE-DIST (Case Distribution): Introduce una expresión “case” en un llamado a una función y mueve el llamado a la función a cada uno de los bloques de “case”. También puede cambiar el alcance de las definiciones de funciones.
5. EMB (Embedding): Cambia el tipo de argumento de las funciones.
6. Crossover: Esta transformación considera un número de transformaciones REQ como alelos (alternativas genéticas) y los recombina utilizando un algoritmo genético que opera sobre genomas (información genética) de largo fijo.

Cuando un individuo es elegido para su expansión, se le aplica una transformación compuesta por una secuencia de transformaciones atómicas. El programa resultante es evaluado para su incorporación al reino. El número de transformaciones compuestas que se aplican a un programa se determina por el límite de costo establecido que corresponde al número de individuos a generar.

Los programas se representan en ADATE-ML, un subconjunto de Standard ML [VAT06]. Standard ML es un lenguaje de programación que presenta las siguientes características:

- Fundamentalmente funcional
- Fuertemente y estáticamente tipado
- Extiende la noción básica de tipos con polimorfismo paramétrico
- Fácil manejo de excepciones
- Sistema de modularización basado en estructuras (que contienen el código), definición aparte del tipo de las estructuras y estructuras parametrizadas
- El compilador infiere los tipos por lo que estos en general no deben ser especificados en cada caso por el programador [PUC01].

ADATE provee un conjunto de tipos de datos ya creados para los que no es necesario definir los tipos de datos y funciones asociadas. ADATE permite inferir programas a partir de casos de prueba, pero no es capaz de inferir nuevos tipos de datos por lo que estos en general deben ser suministrados junto con los casos de prueba [VAT06].

Una especificación en ADATE contiene:

1. Las definiciones de tipos de datos que describen la entrada y la salida.
2. Primitivas y otras funciones auxiliares a ser usadas en la síntesis de los programas.
3. Valores de entrada de ejemplo
4. La función de evaluación de la salida -normalmente parejas de valores de entrada-salida para los valores de entrada de ejemplo [OLS98, VAT06].

Una especificación describe los requerimientos de la función a sintetizar por el sistema, establece lo que los programas debieran hacer, no cómo. Consiste en un archivo de texto que contiene una primera parte escrita en ADATE-ML, separada por el texto “%%” de una segunda parte escrita en Standard ML. La primera parte debe estar disponible para ser usada por la función inferida y la segunda se usa en el proceso de inferencia.

La especificación tiene la estructura siguiente:

```
Código ADATE-ML
%%
Código Standard ML
```

La parte con código ADATE-ML se compone de tres elementos:

1. Tipos de datos y código que puede ser usado por la función inferida.
2. La definición de la función a inferir de nombre *f,* para la que usualmente se especifica que levanta una excepción de las predefinidas en ADATE para ser usadas por *f.*
3. La función *main* encargada de tomar la entrada para *f*.

La parte con código Standard ML puede contener cualquier construcción Standard ML, pero ADATE requiere que se implementen algunas estructuras y funciones particulares:

1. Entradas: una lista ML con los valores de entrada.
2. Validación de entrada: una lista ML con la validación de la entrada (opcional).
3. Tipos Abstractos de Datos: Una lista de los tipos de valores que pueden ser inspeccionados pero no construidos.
4. Funciones a utilizar: Conjunto de funciones necesarias para le inferencia.
5. Funciones de rechazo: Lista de funciones que controlan código redundante.
6. Transformación de restauración: Función que permite alterar individuos para acomodar una nueva función *f.*
7. Grado: Estructura ML usada para asignar un grado a los individuos para los cálculos de evaluación.
8. Función de evaluación de salida: Función para evaluar los individuos.
9. Máximo de salida por género: Número de individuos que pueden existir como salida en un género.
10. Límites de tiempo máximo y de base: Tiempo límite de ejecución de los individuos.

La mayor parte de estas están pensadas para un uso avanzado de ADATE para la síntesis de programas complejos de forma incremental y para mejorar la eficiencia en la búsqueda.

Para la generación de programas simples, los puntos relevantes son: 1. Entradas, 4. Funciones a utilizar y 8. Función de evaluación de salida.

La especificación delimita la porción del espacio de soluciones que es accesible para ADATE durante la inducción. Como ADATE es un sistema relativamente joven, los efectos de las combinaciones de los parámetros en la especificación no se han estudiado completamente y no son comprendidos del todo. Sin embargo, Olsson [OLS98] recomienda que las especificaciones se mantengan simples dado que información superflua tiende a incrementar el esfuerzo de inferencia. Por otro lado es importante que la información suministrada sea adecuada para describir el problema porque podría resultar imposible, o por lo menos más difícil encontrar una solución válida y confiable. En la práctica suele no ser trivial escribir especificaciones que sean simples, no ambiguas, concisas y precisas. Se debiera ver a ADATE como un niño que no sabe nada acerca de su entorno y va a experimentarlo durante el entrenamiento y la especificación debiera hacer que el aprendizaje gradual y evolucionario resulte posible.

El conjunto de datos de entrada de entrenamiento es la parte más importante de la especificación. La falta de cobertura de todos los casos relevantes de entrada puede llevar a que los programas inferidos brin-

den resultados incorrectos en casos no presentes en el conjunto de entrenamiento. Entradas simples son importantes porque ADATE puede encontrar programas para estos de forma más rápida que para entradas más grandes y por lo tanto acelerar y guiar el comienzo de la evolución. La falta de instancias simples puede llevar a una convergencia prematura o a una inferencia ineficiente. El tamaño del conjunto de entrenamiento debiera ser lo suficientemente grande para cubrir todos los casos y tener una progresión en el tamaño de la entrada para facilitar un aprendizaje eficiente.

ADATE puede ejecutar solo en Linux corriendo en CPUs x86. ADATE consiste de código ML que se compila para cada problema individual utilizando el compilador Mlton[VAT06, MLT13].

El sistema ADATE utiliza un algoritmo para seleccionar el individuo más apto que toma en cuenta el tamaño (en bits) y el tiempo de ejecución de los casos de entrada [VAT06], por lo que la propiedad de "hipótesis más simple" se cumple de una manera especial. No resulta del todo claro que cumpla con la propiedad de ser "conservativo" en términos de Zhu.

5.6 Generación de un programa desde de casos de prueba con ADATE

A continuación se describen los pasos que debe seguir un usuario del sistema ADATE para generar un programa a partir de casos de prueba.

El sistema ADATE proporciona los archivos *main1.sml* y *main2.sml*. El usuario crea la especificación en un archivo *spec.sml* (en lugar de spec, podría tener otro nombre). El usuario concatena los archivos *main1.sml, spec.sml, main2.sml* para formar el archivo *main.sml* con el programa fuente necesario para el problema concreto a atacar. El usuario compila *main.sml* utilizando el compilador Mlton [VAT06]. A continuación el usuario corre el ejecutable generado por esta compilación.

Para evaluar el enfoque probamos uno de los ejemplos disponibles en la página Web de ADATE [OLS13] que consiste en la generación de un programa que ordena una lista de números enteros que recibe como entrada. En anexo A se incluye la especificación correspondiente. A continuación describimos el proceso llevado a cabo.

1. Ejecutamos el comando

    makespec SORT.spec

Este comando generó el archivo *SORT.spec.sml*.

2. Concatenamos *main1.sml, SORT.spec.sml, main2.sml* para formar el archivo *main.sml*.
3. Compilamos el archivo main.sml con el compilador Mlton que generó el ejecutable main.
4. Ejecutamos main.

La ejecución de main generó varios archivos entre los cuales SORTxxxx.log y SORTxxxx.trace.

Estos archivos contienen información acerca de los programas que ADATE va generando.

Para cada programa que genera ADATE evalúa los casos de entrada-salida y en función de esa evaluación categoriza al programa como solución candidata o no.

Para identificar el programa más adecuado generado hasta cierto momento el usuario debe buscar en el archivo "trace" la secuencia "pe1g0s1" comenzando desde el final. Para cada programa aparece la infor-

mación de la cantidad de parejas entrada-salida satisfechas. Antes de esa secuencia aparece la identificación del programa. Buscando por esa identificación en el archivo "log" se ubica el código del programa generado. Tanto en el archivo "trace" como en "log" aparece indicada la cantidad de casos de prueba de la entrada satisfechos por el programa generado. ADATE genera múltiples programas, pero los que a partir de reglas heurísticas considera como candidatos a ser solución, los identifica con la secuencia "pe1g0s1".

La ejecución de la generación debe seguir por lo menos hasta que se genere un programa que satisface todas las parejas de entrada-salida.

5.7 Aplicación de IIE con ADATE

Dado un conjunto de casos de prueba derivado de una especificación S es posible aplicar el criterio IIE para aceptar o no el conjunto de casos de prueba. Para aplicar el criterio se puede utilizar ADATE como motor de inferencia pero hay que tomar en cuenta que ADATE no genera un único programa a partir de los casos de prueba sino que genera muchos programas y no tiene un mecanismo de parada, sino que sigue ejecutando sin fin. Es posible obtener en todo momento información acerca del "mejor" programa generado hasta entonces.

A partir de la definición de IIE un CCP satisface el criterio IIE utilizando ADATE si ADATE genera un programa candidato que es correcto con respecto a la especificación. Se debe definir un criterio adicional para determinar en qué condiciones se considera que el criterio IIE no se satisface por lo que a continuación se ajusta la definición del criterio de manera de tomar en cuenta algunas peculiaridades de ADATE.

Definición: *Criterio IIE para ADATE.* Dados una especificación S y un CCP T, se dice que el CCP T es adecuado para S si ADATE genera un programa P' a partir del CCP T tal que P' sea correcto respecto a la especificación S. En caso de que esto no se cumpla, si ADATE genera un programa candidato P" que satisface todos los casos del CCP T y P" no es correcto y luego de cierto número E de evaluaciones adicionales ADATE no encuentra un programa candidato mejor, se dice que el CCP T no satisface el criterio IIE para ADATE.

El valor de E podría depender de la complejidad del programa. Para este caso se fijó el valor de E de forma relativamente arbitraria en veinte millones.

5.8 Ejemplos de aplicación del criterio IIE

Consideremos un programa que recibe como entrada una lista de valores enteros y devuelve una lista con los valores de entrada ordenados de menor a mayor.

Para ese programa se definen dos conjuntos de casos de prueba expresados como parejas entrada-salida esperada. Cada lista aparece ente corchetes, primero la lista de entrada y a continuación la lista con la salida esperada correspondiente. Los casos están separados por el símbolo "/".

Los casos son:

a) [] , [] / [1,2,3] , [1,2,3] / [3,2,1] , [1,2,3]

b) [] , [] / [1,2,4] , [1,2,4] / [1,3,2] , [1,2,3] / [2,1,2] , [1,2,2] / [2,1,-3] , [-3,1,2]

Para evaluar la correctitud de los programas generados utilizamos el CCP **CORR** compuesto por los 17 casos siguientes:

[], [] / [~1], [~1] / [3,1], [1,3] / [1,2,3], [1,2,3] / [1,3,2], 1,2,3] / [2,1,3], [1,2,3] / [2,3,1], [1,2,3] / [3,1,2], [1,2,3] / [3,2,1], [1,2,3] / [1,3,3], [1,3,3] / [3,1,3], [1,3,3] / [3,3,1], [1,3,3] / [1,1,3], [1,3,3] / [1,3,1], [1,1,3] / [3,1,1], [1,1,3] / [3,3,3], [3,3,3] / [7,5,6,5,4,1]. [1,4,5,5,6,7]

Este conjunto fue generado para incluir las siguientes condiciones para la lista de entrada:

a) vacía
b) con un elemento
c) con dos elementos, desordenada
d) con tres elementos con uno, dos y tres valores distintos, todas las permutaciones con repetición posibles
e) con un número grande (seis) de elementos, desordenados.

A partir del CCP **(a)** ADATE generó múltiples programas que satisfacen todo los casos de prueba, pero ninguno de esos programas es capaz de ordenar cualquier lista de enteros. El mejor de ellos, entre otros defectos, ordena mal las listas [1,3.2] y [2,3,1]. ADATE siguió generando programas y no logró encontrar uno mejor que este luego de más de 44 millones de evaluaciones adicionales. Por lo tanto el CCP **(a)** no cumple con el criterio IIE.

A partir del CCP **(b)** ADATE generó también múltiples programas y generó un programa luego de poco más de 17.000 evaluaciones que ordenó de forma adecuada los 17 casos y por inducción completa demostramos que el programa generado ordena de forma correcta listas de cualquier largo, por lo que el CCP **(b)** cumple el criterio IIE.

Capítulo 6: Grado de satisfacción de IIE

Por la forma en la que se definió el criterio IIE, este es del tipo satisface/no satisface. Los criterios de aceptación de tipo caja blanca ofrecen la posibilidad de evaluar un grado de aceptación. Por ejemplo dado el criterio de "cobertura de sentencias" se tiene como resultado un porcentaje de cobertura. Dados tres conjuntos de casos de prueba para un programa P, se pueden tener resultados del tipo: 100%, 96%, 58%. Esto tiene la ventaja de que para aquellos casos en los que no se logra una cobertura del 100% se tiene una idea acerca de cuán lejos o cerca se está de lograr la cobertura deseada. Es deseable poder tener también para IIE información acerca del "grado de cobertura" para los CCP que no cumplen el criterio.

El grado de cumplimiento del criterio para un CCP dado se puede medir utilizando como indicadores el grado de diferencia del programa inferido respecto a la especificación o por otro lado considerar el grado de diferencia entre el CCP y un CCP que cumpla el criterio.

La diferencia entre el programa inferido y la especificación puede considerarse desde el punto de vista del código del programa o de su comportamiento. El primero de estos enfoques parece poco promisorio dado que el código del programa inferido podría tener muy poco que ver con un programa que corresponda a la especificación. Por otro lado pueden existir muchos programas distintos que corresponden a una misma especificación.

Respecto al comportamiento, se puede someter al programa a un conjunto de datos de entrada elegido de forma adecuada. Este enfoque es específico para cada programa a evaluar y no hay un procedimiento general que permita construir uno que resulte adecuado para cada programa.

Otra posibilidad consiste en generar datos de entrada de forma aleatoria con cierta distribución de probabilidad. Si el programa generado es correcto, para el 100% de los casos el programa inferido brindará una salida correcta. Se puede considerar la proporción de casos con resultado correcto como el grado de cumplimiento del criterio IIE. Un programa que para todo dato de entrada genere un resultado incorrecto tendrá un cumplimiento de 0%.

La diferencia entre el CCP y uno que cumpla con el criterio la podemos caracterizar considerando algún atributo de casos de prueba adicionales que adicionados al CCP original den como resultado un CCP que sí cumpla con el criterio. Para obtener un grado de cumplimiento se puede considerar la relación entre el atributo de esa diferencia y el atributo de un CCP mínimo.

Entre los atributos que se pueden tomar en cuenta están:

a) la cantidad de casos de prueba
b) la complejidad de Kolmogorov de la cadena de caracteres que corresponde a un CCP
c) el largo de la cadena de caracteres del CCP
d) la cantidad de campos de entrada-salida en el CCP.

El grado de cumplimiento para el conjunto vacío debiera ser 0. La medición del grado debiera ser monotónica en el sentido de que si un CCP está incluido en otro, el grado de su cobertura debiera ser menor o igual a la del otro.

$$\text{Si } CCPA \subseteq CCPB, \text{ entonces } grado(CCPA) <= grado(CCPB)$$

Todas las alternativas consideradas cumplen con estas propiedades.

En la sección siguiente se analiza la evaluación del grado de cumplimiento de IIE mediante entradas aleatorias y en las tres siguientes se analizan las correspondientes a atributos de los casos de prueba adicionales respecto a un CCP de referencia.

6.1 Evaluación del grado de cumplimiento de IIE mediante entradas aleatorias

Definimos como Error del programa inferido a la probabilidad de que el programa inferido tenga un resultado incorrecto respecto a la especificación S, dada una determinada distribución de probabilidad de los valores de entrada. El grado de cumplimiento de IIE será el complemento de ese error.

En esta definición se hace referencia a una cierta distribución de probabilidad de los valores de entrada, la que se requiere para determinar la probabilidad de que el programa inferido falle. La elección de la distribución va a tener impacto sobre el error del programa inferido y por lo tanto también sobre el grado de cumplimiento del criterio. Por ejemplo, si un programa que recibe un número como entrada falla cuando la entrada es par y genera un resultado correcto cuando esta es impar, si los datos de entrada se generan de tal forma que la probabilidad de que sea par es igual a la de que sea impar y es 0,5 , la probabilidad de que el resultado sea incorrecto será 0,5. Si por el contrario se generan de forma tal que la probabilidad de que sea par es 0,1 y la probabilidad de que sea impar 0,9 , la probabilidad de que el resultado sea incorrecto será 0,1.

Apliquemos este enfoque para evaluar el grado de cumplimiento de IIE de los CCP **(a)** y **(b)** del programa que ordena una lista de valores enteros presentados en la sección 5.8. Existen dos factores que afectan los datos de entrada: el largo de la lista y los valores que la integran. Las probabilidades que asignemos a los casos “Lista vacía” y “Lista con un elemento” van a incidir en los resultados y presentan características especiales, por lo que en una primera instancia no los tomamos en cuenta. Se generaron 1101 listas. Para su largo se partió de listas de largo 2 y su largo final se determinó mediante un sorteo para cada largo con probabilidad 0,2 de no agregar más elementos. El conjunto resultante consta de listas de largo entre 2 y 26 con la siguiente cantidad de casos para cada uno: 210, 181, 134, 120, 91, 71, 71, 36, 40, 33, 24, 19, 21, 13, 10, 9, 4, 3, 1, 4, 2, 0, 1, 0, 3.

A posteriori se sortearon valores para cada una de estas listas con una distribución uniforme entre valores enteros 1 a 10. De esta forma es seguro que van a aparecer valores repetidos. A continuación se agregaron los casos “lista vacía” y lista con un elemento, con lo que la muestra quedó formada por 1103 listas. Denominamos **RND** a este CCP. El tamaño de la muestra resulta adecuado para estimar una proporción en una población infinita con proporción desconocida, con un error del 4% y un nivel de confianza del 99%. Cada uno de los programas generados se ejecutó con este CCP. La evaluación de los resultados se realizó utilizando una planilla electrónica. Los resultados se presentan en el Cuadro 2.

CCP	Resultados Correctos	Resultados Incorrectos	% Correctos
a	453	650	41,07
b	1103	0	100

Cuadro 2: Grado como porcentaje de resultados correctos

Este enfoque es fácil de implementar si se dispone de un oráculo que permita evaluar los resultados. Presenta la limitación de que los valores que se obtienen dependen de la elección de la distribución de probabilidad de los valores de entrada. La estructura de los datos de entrada puede, como en este caso, requerir que se defina una distribución de probabilidad en más de una dimensión.

6.2 Evaluación del grado de cumplimiento de IIE por la cantidad de casos adicionales necesarios

Una forma de evaluar el grado de cumplimiento del criterio consiste en estimar la cantidad mínima de casos de prueba adicionales necesarios para que ADATE genere un programa correcto y considerar como grado de cumplimiento 1 menos la proporción entre casos adicionales y el número mínimo de casos. A continuación se analizan estas ideas y se explora la determinación del número mínimo de casos necesario para que un CCP del programa considerado cumpla con IIE.

Consideremos un caso de prueba C1 conformado por (C1e, C1s) en donde C1e es la entrada y C1s la salida esperada. Si la MII generó un programa P' a partir de un CCP T, el caso C1 aporta información nueva si el resultado esperado de C1 es distinto del resultado que brinda P' :

C1s <> P'(C1e)

Diremos en este caso que el C1 es “discordante” con el programa P' inferido.

Seguimos considerando los CCP **(a)** y **(b)** introducidos en la sección 5.8:

a) [] , [] / [1,2,3] , [1,2,3] / [3,2,1] , [1,2,3]

b) [] , [] / [1,2,4] , [1,2,4] / [1,3,2] , [1,2,3] / [2,1,2] , [1,2,2] / [2,1,-3] , [-3,1,2]

El CCP **(b)** permitió generar un programa correcto por lo que tomamos este CCP como referencia. Los casos 2 y 3 de ambos CCP resultan bastante similares, por lo que se optó por completar el CCP **(a)** con los casos 4 y 5 del CCP **(b)** para conformar el CCP **(c)**:

[],[] / [1,2,3],[1,2,3] / [3,2,1],[1,2,3] / [2,1,2],[1,2,2] / [2,1,-3],[-3,1,2]

Con este CCP ADATE no logró generar un programa correcto, por lo que se analizaron más en detalle las características de los casos de prueba involucrados. El CCP debe suministrar el máximo de información posible a la máquina de inferencia. Los casos 3 y 5 del CCP **(c)** aportan la misma información ya que [3,2,1] y [2,1,-3] están ambos ordenados en forma descendente.

Se definió entonces el CCP **(d)** compuesto por:

[],[] / [1,2,3],[1,2,3] / [3,2,1],[1,2,3] / [2,1,2],[1,2,2] / [1,3,2],[1,2,3].

Este CCP contiene la misma información respecto al ordenamiento que el CCP **(b)** y para este ADATE generó un programa correcto, equivalente al generado para el CCP **(b)**.

Los casos [3,2, 1], [1, 2, 3] / [2,1, -3], [-3, 1, 2] pertenecen a una misma clase de equivalencia en el sentido tradicional de que resulta esperable un comportamiento análogo del programa, pero con una gra-

nularidad muchos más fina de lo usual. Denominamos a estas clases como “clases con igual información relevante para la especificación”. En lo que sigue nos referimos a ellas como “clases con igual información”.

Sus elementos se caracterizan por suministrar exactamente la misma información a la herramienta de inferencia inductiva respecto al ordenamiento. Esto constituye una relación de equivalencia definida en el conjunto de datos de entrada, por lo que establece una partición en clases de equivalencia disjuntas en ese conjunto. Estas clases existen para cada tamaño de lista.

Corresponde notar que al identificar clases de equivalencia utilizando la técnica de partición en clases de equivalencia se obtienen clases que pueden solaparse, por lo que pueden existir casos de prueba que cubran más de una clase [MYE04], a diferencia de las clases ”con igual información”, en que un caso solo pertenece a una clase.

Parece existir una relación fuerte entre la partición en clases de equivalencia con igual información y el criterio IIE. Podemos conjeturar que el mínimo de casos de prueba necesarios para cumplir con el criterio IIE para un programa dado está compuesto por casos que corresponden a clases con igual información distintas ya que si tuviéramos dos casos que brindaran la misma información, resulta plausible que pudiéramos prescindir de uno de ellos.

Sin embargo, a partir del análisis del CCP **(b)** podemos afirmar que el criterio IIE no exige cubrir todas las clases con igual información para un tamaño de lista dado, y tampoco requiere cubrir todos los tamaños de listas. La relación entre estos elementos y el criterio IIE merece un análisis más profundo que escapa al alcance de este trabajo.

Para determinar el mínimo de casos necesario para cumplir el criterio IIE se consideraron CCP con tres casos no vacíos.

Se probó primero el CCP **Min1**: [] , [] / [1,2,3] , [1,2,3] / [2,3,1] , [1,2,3] / [3,2,1] , [1,2,3] con el que ADATE no logró generar un programa correcto luego de más de 40 millones de evaluaciones posteriores a la generación de un programa que satisfacía ese CCP. El mejor programa candidato generado por ADATE satisfacía casi la totalidad del CCP de evaluación, a excepción del caso [1,3,2],[1,2,3]. Se optó por sustituir el primer caso [1,2,3] / [1,2,3] por este caso discordante. Por lo que en segundo término se probó con el CCP **Min2**: [] , [] / [1,3,2] , [1,2,3]) / [2,3,1] , [1,2,3] / [3,2,1] , [1,2,3], con el que ADATE generó un programa correcto en un número muy reducido de evaluaciones.

A continuación se probó con el CCP **Min3** obtenido dejando de lado la lista vacía: [1,3,2],[1,2,3]/ [2,3,1],[1,2,3]/[3,2,1],[1,2,3].

Con este CCP ADATE también generó un programa correcto, pero requiriendo para ello 15.941.534 evaluaciones. Hay que tomar en cuenta que ADATE recibió además de los CCP la definición recursiva del tipo lista. Seguimos sin tener certeza respecto a si el número mínimo de casos necesario para cumplir el criterio IIE es 3 o es posible lograrlo con un número todavía menor de casos.

Esto muestra que:

a) La determinación del número mínimo de casos necesarios para cumplir con el criterio IIE resulta una tarea compleja y no parece haber un procedimiento general efectivo que nos permita determinar ese número.

b) No todas las clases de equivalencia con igual información brindan la misma cantidad de información, por lo que podemos conjeturar que los CCP que cumplen con el criterio IIE debieran estar conformados por casos pertenecientes a distintas clases con igual información que brindan la mayor cantidad de información respecto al tratamiento a efectuar a los datos de entrada.

c) La definición de los tipos de datos que requiere ADATE brinda información adicional a los CCP que resulta relevante en la generación de los programas.

La cantidad de casos mínima va a estar dada por el mínimo de casos que puedan suministrar la información necesaria acerca del comportamiento del programa. Surge la pregunta de si será posible brindar la misma información dada por los tres casos anteriores a través de dos casos de prueba. Para ello combinamos el primero y segundo caso en uno solo, con lo que generamos el CCP conformado por: [1,3,2,5,6,4],[1,2,3,4,5,6]/[3,2,1],[1,2,3]. Probamos primero la generación con este CCP al que se le agregó el caso de lista vacía: [],[] y que denominamos CCP **Min4**. Con este CCP ADATE generó un programa correcto luego de 499.995 evaluaciones. A continuación probamos con el CCP sin el agregado de lista vacía (CCP **Min5**) y generó un programa correcto luego de 7.222.343 evaluaciones.

A continuación, se consideró además de la lista vacía un único caso que integrara toda la información: [],[]/[1,3,2,5,6,4,9,8,7],[1,2,3,4,5,6,7,8,9]. Con este CCP que denominamos CCP **Min6**, ADATE no logró encontrar un programa correcto luego de más de 38.000.000 de evaluaciones.

A partir de los resultados anteriores, no parece adecuado medir el grado de cumplimiento por la cantidad de casos de prueba ya que la misma información puede ser brindada con distinto número de casos.

6.3 Evaluación del grado de cumplimiento de IIE considerando la Complejidad de Kolmogorov o el largo de la cadena

Dado que lo que queremos medir es la cantidad de información en una secuencia de caracteres, parece adecuado considerar la Complejidad de Kolmogorov (CK) de la secuencia que corresponde a la cantidad de información en bits mínima necesaria para especificarla. Si la secuencia es corta, su complejidad corresponde al largo de la secuencia. Si es más larga y puede ser comprimida como una secuencia y un programa que reconstruye la secuencia original, su complejidad corresponderá al largo del programa que permite expresar un modelo para la secuencia y su codificación en ese modelo [LI08].

La CK de la especificación S, la CK de los casos de prueba que cumplen el criterio IIE (CCP(IIE)) y la CK de la Máquina de Inferencia Inductiva (CK(MII)) están relacionados por la desigualdad:

CK(S) <= CK(MII) + CK (CCP(IIE))

La CK de una secuencia corta de caracteres coincide con el largo de la secuencia, por lo que para tamaños reducidos resulta preferible medir el grado directamente por el largo de la secuencia de caracteres que hay que agregar al CCP respecto al largo del CCP de largo mínimo en caracteres que cumple con IIE.

El CCP de largo mínimo de los que se evaluaron que cumple IIE consta de dos casos de prueba y tiene un largo de 44 caracteres. No podemos estar seguros de que este sea efectivamente el valor mínimo, pero sin lugar a dudas es un valor al menos próximo al mínimo.

Para determinar los casos de prueba adicionales necesarios se partió del CCP **RND. RND** se ordenó alfabéticamente de forma tal que aparecieran primero los casos de prueba con menor número de elementos en los datos de entrada. Denominamos **RNDO** a este CCP ordenado. Para el CCP **(a)**, se consideró el primer caso de prueba con resultado incorrecto ("discordante") en la ejecución de evaluación del programa generado, por lo que se se agregó al CCP el caso: [1,10,9],[1,9,10]. Se corrió ADATE para este nuevo conjunto. ADATE generó un programa que satisface todos los casos, pero que no es correcto.

Se agregó al CCP anterior el caso [2,7,1], [1,2,7] que fue el primero incorrecto hallado. Con este nuevo CCP el programa generado sí fue correcto. Los casos de prueba adicionales agregados, incluyendo los símbolos de separación tienen un largo total del 32 caracteres.

El grado de cumplimiento se calcula como (cantidad mínima de caracteres – cantidad de caracteres adicionales)/(cantidad mínima de caracteres). Si el numerador fuera negativo, se considera que el cumplimiento es cero. El grado de cumplimiento para el CCP **(a)** del criterio IIE es por lo tanto (43-32)/43= 0,26 (Ver Cuadro 3).

CCP	Casos Adicionales Necesarios	Largo Total de Casos Adicionales	Grado de Cumplimiento de IIE respecto a mínimo estimado (44)	Grado de Cumplimiento de IIE respecto a CCP de referenci a partir de lista aleatoria (74)
a	[1,10,9],[1,9,10]/[2,7,1],[1,2,7]	32	26	55
b		0	100	100

Cuadro 3: Grado de cumplimiento por casos adicionales

Esta manera de medir el grado de cumplimiento requiere determinar el largo mínimo de la cadena de caracteres necesario para cumplir IIE. La determinación de ese valor no parece ser sencilla y en general puede no ser factible hacerlo. En la aplicación del enfoque se evitó este problema eligiendo un CCP que cumple IIE y que si bien no se sabe si es mínimo, sí puede considerarse como suficientemente próximo al mismo. La elección de este largo tiene impacto sobre el grado de cumplimiento calculado.

Una forma de evitar la determinación del largo mínimo consiste en tomar como referencia el largo de un CCP generado mediante un procedimiento establecido en el que utilizamos el CCP **RNDO**. Se parte de un CCP vacío. A este CCP le corresponde un programa vacío que no hace nada. Este programa falla con el primer caso del CCP **RNDO**, por lo que deberemos agregar el caso [] , [] y luego el [1] , [1]. Dado este CCP el programa generado falla en el siguiente caso de **RNDO**: [1,1] , [1,1]. El siguiente falla en el caso [1 , 10] , [1 , 10]. El siguiente a su vez falla en el [10 , 1] , [1 , 10] y el siguiente en [1,10,9] ,[1,9,10] y por último en [10,1,2] , [1,2,10].

De esta forma se obtiene el CCP **REF** de referencia: [] , [] / [1] , [1] / [10,1] , [1,10] / [1,1] , [1,1] / [1,10,9] , [1,9,10] / [10,1,2] , [1,2,10] con un largo de 74 caracteres y conformado por 24 campos. En el Cuadro 4 se presentan los valores obtenidos con este valor de referencia. Además de evitar el problema de la

determinación del mínimo, el uso del mismo procedimiento para determinar el CCP de referencia y el agregado de casos resulta más consistente y asegura que el valor mínimo posible del grado es 0.

6.4 Evaluación del grado de cumplimiento de IIE considerando la cantidad de campos de entrada-salida en el CCP

En la sección anterior se evaluó el grado de cumplimiento tomando en cuenta los largos de las cadenas de caracteres. Si el rango de valores posible de un campo puede tener largos de cadena de caracteres muy distintos la evaluación estaría favoreciendo el uso de los largos más cortos posibles lo cual podría afectar de manera significativa los resultados. Para evitar este problema se puede tomar en cuenta la cantidad de campos de entrada-salida en lugar de la cantidad de caracteres. En el Cuadro 4 se rehacen los cálculos aplicando este criterio.

CCP	Casos Adicionales Necesarios	Cantidad Total de Campos Adicionales	Grado de Cumplimiento de IIE respecto a mínimo estimado (18)	Grado de Cumplimiento de IIE respecto a CCP REF (24)
a	[1,10,9],[1,9,10]/[2,7,1],[1,2,7]	12	33	50
b		0	100	100

Cuadro 4: Grado de cumplimiento por campos adicionales

6.5 Elección de dos formas de evaluar el grado de cumplimiento de IIE

A partir de los análisis previos se optó por evaluar el grado de cumplimiento del criterio de dos formas: por un lado en base al comportamiento del programa generado por la MII sometido a entradas aleatorias con cierta distribución de probabilidad y por otro en base a la cantidad de campos adicionales respecto a un CCP de referencia generado a partir de las mismas entradas aleatorias ordenadas de forma que aparezcan primero los casos con menor cantidad de campos de entrada-salida. De esta manera se evitan los problemas relacionados con el tamaño mínimo y se aprovecha la generación de valores aleatorios en ambas mediciones. Los resultados de ambas mediciones dependen de la elección de la distribución de probabilidad de los valores de entrada.

Capítulo 7: IIE en la evaluación de un experimento

El criterio IIE puede ser utilizado como un indicador de la consistencia con la que sujetos aplicaron una técnica de selección de casos de prueba en un experimento destinado a evaluar la efectividad de técnicas de selección de casos de prueba. Pusimos a prueba este enfoque aplicando el criterio a los conjuntos de casos de prueba generados por 11 sujetos utilizando la técnica "Partición en clases de equivalencia con análisis de borde" en un experimento que se llevó a cabo en Montevideo en el año 2009 [VALL09]. Como parte del experimento todos los sujetos recibieron entrenamiento en esa técnica de selección de casos de prueba y a continuación recibieron un programa para el que debían construir casos de prueba utilizando esa técnica.

Ese programa escrito en Java, recibe una lista de enteros y devuelve la lista de enteros ordenada de menor a mayor.

7.1 Aplicación de IIE con ADATE

En la aplicación del criterio IIE utilizamos el sistema ADATE para llevar a cabo la inferencia. Para ello generamos un archivo de especificación de ADATE con el conjunto de casos de prueba formulados por cada uno de los sujetos.

En muchos casos los conjuntos de casos de prueba incluían valores de entrada incorrectos (no enteros), incluso en algunos casos la mayor parte de los casos de prueba correspondían a tipos no válidos al punto que algunos sujetos pusieron foco sobre los tipos inválidos y plantearon muy pocos casos para tipos válidos. Standard ML es fuertemente tipado y el control de tipos se hace en el momento de la compilación, por lo que estos casos no fueron considerados en la generación. Estos casos corresponden en realidad a contra-ejemplos. Incorporarlos hubiera complicado significativamente la tarea sin aportar demasiado a cambio. Esto constituye una diferencia que puede ser significativa al momento de evaluar la aplicación de la técnica "Partición en Clases de Equivalencia con Análisis de Borde" utilizando el criterio IIE.

Con ADATE es posible aplicar el criterio IIE para las clases válidas, pero para las clases inválidas solo resulta práctico considerar aquellas con tipos válidos. Por esta razón los tipos inválidos no se consideran en este análisis. En la sección 7.2 se describe una manera posible de tomar en cuenta los tipos no válidos al aplicar el criterio IIE.

Corresponde notar que en la aplicación del experimento los sujetos tampoco pudieron tomar en cuenta los casos de tipos inválidos ya que las pruebas fueron implementadas con Junit y tampoco pudieron ejecutar los casos para tipos inválidos, por lo que estos casos no pudieron ser tomados en cuenta por los experimentadores.

Para detectar programas incorrectos utilizamos el CCP **CORR** presentado en la sección 5.8. Los programas que brindaron resultados correctos para estos datos de entrada diremos que "verificaron el test de correctitud".

Las generaciones se llevaron a cabo en un equipo con procesador AMD Phenom II X4 B95 a 3.00 GHz con sistema operativo Windows 7 con 2,75 GB de memoria utilizable, sobre el que se instaló Oracle Virtual Box, en donde se definió una máquina virtual con 1385 MB de memoria asignada, en la que se instaló la versión de Linux Crunchbang, ambiente en el que se ejecutó ADATE versión 0.50 usando el compilador Ml-ton 20041109-1.i386 que viene incluido en la distribución de esa versión de ADATE.

Los programas generados se probaron en el ambiente interactivo suministrado por Standard ML of New Jersey v110.74 corriendo en Windows.

Como ADATE ejecuta sin fin y se requiere forzar su terminación, el tiempo de ejecución en cada caso resultó diferente. En cada corrida se esperó a que ADATE generara un programa candidato que satisficiera todos los casos de prueba incluidos en la especificación de la generación.

En el Cuadro 5 se detalla el tiempo ADATE (Global Time) total al momento de parar cada ejecución y la cantidad total de evaluaciones realizadas en cada una. En el proceso de generación ADATE genera y eva-lúa una gran cantidad de programas, gran parte de los cuales tan solo sirven para el "cruzamiento genético" y no constituyen candidatos de solución. En razón de ello en lo que sigue vamos a denominar como "pro-grama generado" por ADATE solo a aquellos que ADATE asocia con la secuencia "pe1g0s1" (programas can-didatos de solución).

Su-je-to	Tiempo Global	Evaluacio-nes	Veri-fica prue ba	Genera-do en Tiempo	Generado tras Evalua-ciones	Tiempo desde ge-ración an-terior	Evaluaciones desde progra-ma anterior	Tiempo posterior	Evaluacio-nes poste-riores
1	3.758,20	45.315.822		88,27	928.211	87,15	918.276	3.669,93	44.387.611
2	4.254,44	36.033.305		36,87	272.450	15,82	140.138	4.217,57	35.760.855
3	0,00	47.757.174		2.952,24	27.439.177	1,51	4.023	2.630,63	20.317.997
4	2.811,37	30.426.038		206,71	2.153.499	206,02	2.144.515	2.604,66	28.272.539
5	3.508,47	23.569.826		248,94	1.026.333	248,9	1.025.723	3.259,53	22.543.493
6	13.934,39	95.358.756		79,44	492.019	79,41	192.990	13.785,62	94.314.532
7	2.640,25	23.895.062	S	1.859,29	18.948.349	54,67	52.756	758,04	4.912.774
8	13.034,45	57.845.624	S	17,55	39.664	6,05	4.570	13.016,90	57.805.960
9	5.470,38	58.059.752		1.529,65	17.412.111	396,98	2.764.303	3.940,73	40.647.641
10	207,98	1.782.191		5,49	50.536	5,46	45.504	202,49	1.731.655
11	7.497,65	68.647.441	S	7,32	36.326	4,85	17.638	7.490,33	68.611.115

Cuadro 5: Tiempos, evaluaciones y verificación de la prueba

En aquellos casos en los que un programa generado verificó la prueba de correctitud se indica en qué tiempo de ejecución fue generado y el número de evaluaciones realizadas para obtenerlo. En los casos en que ningún programa generado verificó la prueba de correctitud, se indica el tiempo de generación del úl-timo programa generado. En todos los casos se indica el tiempo total y el número de evaluaciones entre la generación considerada y la terminación de la ejecución

Sujeto	Parejas entrada-salida	Evaluación con casos de prueba	Largos en lista de entrada válidas
1	[],[]/[1,2,3],[1,2,3]/[3,2,1],[1,2,3]	Ordena mal [1,3,2], [2,3,1], entra en ciclo sin fin para [3,1,3], [3,3,1], [1,1,3], [3,1,1]	0,3
2	[],[]/[4,2,3,1,3],[1,2,3,3,4]/[1,2,3,3,4],[1,2,3,3,4]/ [2,3,1,4],[1,2,3,4]/[1,2,3,4],[1,2,3,4]	Ordena mal [1,3,2] y [1,3,1]	0,5,4,
3	(*)/[],[]/[1],[1]/[0,1,2,3,4,5],[0,1,2,3,4,5]/[5,4,5,2,1,4], [1,2,4,4,5,5]/	Ordena mal [1,3,2]	0,1,6
4	(*)/[],[]/[0],[0]/[-99999,999999],[-99999,999999]/ [4,2,3,2],[2,2,3,4]	Ordena mal [3,1], genera listas de largo 4 para las listas de entrada de largo 3 y en varios casos no ordenadas. Para la entrada de largo 6 genera lista de largo 4	0,1,2,4
5	[1,3,2,Max,Min],[Min,1,2,3,Max]/(hola)/(1,5,hola)/(maxletos+1)/(Min-1)/(Max+1)/(null)/([2],[3])/([],[])/ [i*3432],[i*3432]	Ordena mal [1,2,3],[1,3,2],[2,1,3],[3,1,2], [3,2,1],[1,3,3],[3,1,3], [1,1,3], [3,1,1], [7,5,6,5,4,1]	5,0
6	[Max,3,2,Min],[Min,2,3,Max]/([],[])/([no int]]/([no int2])/(*)/([2],[3])/([no int3])	Ordena mal [1,2,3], [1,3,2],[2,1,3], [2,3,1],[3,1,2],[1,3,3],[3,1,3],[1,1,3], [1,3,1],[7,5,6,5,4,1]	4,0
7	(*)/[],[]/[0.01],[0.01]/[-1],[-1]/[0,0],[0,0]/[1,3,2],[1,2,3]/ [1,2,4,5,2,2],[1,2,2,2,4,5]/[4,3,2,1,1],[1,1,2,3,4]/ [1,3,2,1,2,3],[1,1,2,2,3,3]/[1,1,2,2,3,3,],[1,1,2,2,3,3]	Verifica la prueba	0,1,3,6,5
8	[],[]/[1,-1],[-1,1]/[2,0],[0,2]/[0,-2],[-2,0]/[4,2],[2,4]/[-2,-4],[-4,-2]/[5,6],[5,6]/[-4,-3],[-4,-3]/[0,2],[0,2]/[-2,0],-[2,0]/[2,4,-3,-1],[-3,-1,2,4]/[3,1,5,0],[0,1,3,5]/[-4,0,-1,-5],[-5,-4,-1,0]/[5,1,3,2],[1,2,3,5]/[-5,-1,-4,-3],[-5,-4,-3,-1]/[1,2,3,4],[1,2,3,4]/[-4,-3,-2,-1],[-4,-3,-2,-1]/[0,2,4,6], [0,2,4,6]/[-3,-2,-1,0],[-3,-2,-1,0]/[2],[2]/[0],[0]/[-2],[-2]/ [-2,3,-1],[-2,-1,3]/[2,-1,3],[-1,2,3]/[-1,0,-2],[-2,-1,0]/ [0,3,2],[0,2,3]/[-2,-4,-1],[-4,-2,-1]/[4,1,3],[1,3,4]/[-4,-2,3],[-4,-2,3]/[-2,1,3],[-2,1,3]/[-4,-2,0],[-4,-2,0]/[0,2,4], [0,2,4]/[-3,-2,-1],[-3,-2,-1]/[1,3,5],[1,3,5]/(*)/ (4294967296)/(-4294967296)	Verifica la prueba	0,2,4,1,3
9	(*)/([b,c,a])/[],[]/[1],[1]/[4,-2,0,15,3,-10,10,-1,34,-50,5,-87,8],[-87,-50,-10,-2,-1,0,3,4,5,8,10,15,34]/[50,-2,0,1,3,10,-1,-50,-1,0,8,-3,50],[-50,-3,-2,-1,-1,0,0,1,3,8,10,50,50]/[4,4,4,4,4],[4,4,4,4,4]	ordena mal [1,3,2],[1,3,1]	0,1,13,5
10	[],[]/(*)/[1,-2,-1],[-1,-2,1]!!;([numero])	no soporta largos 1 y 2 de lista, ordena mal [1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2], [1,3,3],[3,1,3],[1,1,3],[1,3,1] y recorta [7,5,6,5,4,1] a largo 3	0,3
11	[],[]/[1,2,4],[1,2,4]/[1,3,2],[1,2,3]/[2,1,2],[1,2,2]/[2,1,-3],[-3,1,2]/(*)	Verifica la prueba	0,3

Cuadro 6: Casos de prueba y evaluación del mejor programa generado por ADATE

En el Cuadro 6 se presentan las parejas entrada-salida que planteó cada sujeto y los resultados obtenidos en las pruebas del mejor programa generado por ADATE a partir de esas parejas. Para todos ellos ADATE generó al menos un programa que satisfizo todos los casos de entrada correspondientes.

En 8 instancias (correspondientes a los sujetos 1, 2, 3, 4, 5 ,6, 9, 10) de las 11, el programa generado no fue correcto a pesar de que en cada una de ellas el programa generado satisface todas las parejas de entrada-salida. En tres de las otras instancias (sujetos 7, 8, 11) el programa generado verificó la prueba. Deberemos evaluar si los programas generados son correctos para determinar el cumplimiento del criterio IIE.

A continuación se presenta el código generado en estas tres instancias identificados como PG y el número de sujeto.

PG7

```
fun f Xs =
  case Xs of
    nill => Xs
  | cons( V1927, V1928 ) =>
  case f( V1928 ) of
    nill => Xs
  | cons( V3350, V3351 ) =>
  case ( V1927 < V3350 ) of
    false => cons( V3350, f( cons( V1927, V3351 ) ) )
  | true => cons( V1927, f( V1928 ) )
```

PG8

```
fun f Xs =
  case Xs of
    nill => Xs
  | cons( V1A03, V1A04 ) =>
  case f( V1A04 ) of
    nill => Xs
  | cons( VCF06, VCF07 ) =>
  case ( VCF06 < V1A03 ) of
    false => cons( V1A03, f( V1A04 ) )
  | true => f( cons( VCF06, f( cons( V1A03, f( VCF07 ) ) ) ) )
```

PG11

```
fun f Xs =
  case Xs of
    nill => Xs
  | cons( V42AB [V1927], V42AC [V1928]) =>
  case f( V42AC [V1928] ) of
    nill => Xs
  | cons( V42AE [V3350], V42AF [V3351]) =>
  case ( V42AB [V1927]< V42AE [V3350]) of
    false => cons( V42AE [V3350], f( cons( V42AB [V1927], V42AF [V3351]) ) )
  | true => cons( V42AB [V1927], f( V42AC [V1928] ) )
```

Al comparar estos tres programas resulta notorio que tienen una estructura muy similar. En particular, los programas P7 y P11 resultan equivalentes ya que se obtiene PG8 a partir de P11 sustituyendo los literales de PG11 de acuerdo a la correspondencia siguiente:

- V42AB [V1927]
- V42AC [V1928]

– V42AE [V3350]
– V42AF [V3351]

En anexo B se presenta PG11 anotado con los literales a sustituir entre corchetes [].

Para demostrar que estos tres programas son correctos, basta con demostrar que PG7 y PG8 son correctos. Standard ML es un lenguaje funcional, lo que facilita las demostraciones. En anexo C se presentan las demostraciones por inducción completa de que ambos programas ordenan de forma correcta listas de largo n.

La generación correspondiente al sujeto 8 mostró un comportamiento no previsto del sistema ADATE. Este sistema, además de llevar la cuenta de los casos de prueba de entrada verificados por el programa generado, realiza una evaluación del grado de confianza que merece la satisfacción de cada caso de prueba respecto a su validez general a partir de reglas heurísticas . El sujeto 8 propuso 34 casos y el primer programa generado que los satisfizo todos fue evaluado como que solo 18 de los casos de entrada resultaban de confianza. ADATE continuó generando programas y fue logrando aumentar el número de casos de confianza, pero a costa de obtener programas cada vez más complejos que verificaban el conjunto de casos de entrada. En razón de ello corresponde evaluar todos los programas que fueron generados y que satisfacen la totalidad del conjunto de casos de prueba de entrada y no tan solo el último ya que unos pueden resultar correctos y otros no.

En 8 de las 11 instancias (73%) no resultó posible generar con ADATE un programa consistente con la especificación, por lo que no fue posible cumplir el criterio de aceptación IIE. Surge la pregunta acerca de qué significa esto respecto a la calidad de los conjuntos de casos de prueba elegidos.

La instancia 10 contiene un caso de prueba en el que el resultado es inconsistente con la entrada ya que para la lista de entrada [-1,-2,1] la salida esperada establecida es la misma lista [-1,-2,1] , siendo [-2,-1,1] el resultado correcto. Para esta instancia sí podemos afirmar que el conjunto de casos de prueba no es adecuado.

Para el resto de las instancias (1, 2, 3,4, 5, 6, 9) no tenemos certeza de que ADATE no pudiera generar el programa correcto con más tiempo de ejecución. Vamos a analizar a continuación las características de los conjuntos de casos de prueba que permitieron generar un programa correcto y las de las instancias en que no.

Los primeros contienen casos que cubren las siguientes categorías:

a) Lista vacía []
b) Entrada con un elemento (excepto la instancia 11)
c) Entrada desordenada
d) Entrada ordenada
e) Entrada con valores repetidos
f) Entrada sin valores repetidos.

En el Cuadro 7 se indica para cada instancia, con excepción de la 10 que es atípica, la cantidad de casos correspondientes a cada una de estas categorías. Las instancias se presentan ordenadas de mayor a menor por la cantidad total de casos de prueba con valores desordenados y para un mismo valor de estos, por la cantidad total de casos.

Instancia	Vacía	Un elemento	Desordenada Sin Reps.	Desordenada Con Reps.	Total desordenados	Ordenada Sin Repetidos	Ordenada Con Repetidos	Total de casos	Largos de lista	Programa Generado Correcto
8	1	3	17		17	13		34	0,1,2,3,4	S
7	1	1	1	4	5		1	8	0,1,2,3,5,6	S
11	1		2	1	3	1		5	0,3	S
2	1		1	1	2	1	1	5	0,4,5	N
9	1	1	1	1	2		1	5	0,1,5,13	N
3	1	1		1	1	1		4	0,1,6	N
4	1	1		1	1	1		4	0,1,2,4	N
1	1		1		1	1		3	0,3	N
5	1		1		1			2	0,5	N
6	1		1		1			2	0,4	N

Cuadro 7: Tiempos, evaluaciones y verificación de la prueba

En este cuadro quedaron agrupadas en la parte superior las tres instancias para las que ADATE generó un programa correcto. Parecería que con menos de tres casos con valores desordenados, ADATE no es capaz de inferir un programa que ordene los valores de entrada. Sin embargo corresponde notar que sí lo hizo con los CCP **Min4** y **Min5**.

Las instancias 1, 3, 4, 5 y 6 tienen tan solo un caso cada una con valores de entrada desordenados. Esto se puede considerar correcto desde el punto de vista de partición en clases de equivalencia, pero no desde el punto de vista de partición en clases de equivalencia con análisis de borde. En esta técnica, para un clase se eligen normalmente varios casos representativos de la clase, alguno en cada borde y alguno intermedio. Pero no está claro cuáles serían los "bordes" de una lista desordenada. Una entrada de este tipo tiene varias dimensiones:

- el largo de la lista
- el "grado" de desorden
- el valor de cada elemento.

Los largos de lista 0 y 1 corresponden a clases definidas y distintas del resto. El largo 2 constituiría un borde, un largo de lista muy grande podría considerarse como representativo de otro borde y también habría que probar con un largo intermedio. Esto daría tres casos de prueba con valores desordenados.

Un análisis similar se puede hacer para el "grado" de desorden de una lista y para los valores de los elementos. Por ejemplo: todos ordenados salvo el último, o el primero. Con esta comprensión de la técnica, su aplicación solo habría sido adecuada en las instancias 8 y 7 y estaría en el límite para la 11.

Respecto a los valores de los elementos corresponde tomar en cuenta los valores máximos y mínimos posibles, lo que aparece considerado en las instancias 4, 5, y 6.

7.2 Clases inválidas correspondientes a tipos no válidos

Los casos de prueba con tipos no válidos se pueden considerar aparte con una entrada de tipo texto. En la especificación de los casos de prueba el resultado esperado podría tomar los valores: Válido o No Válido. En este análisis se debieran incluir todos los casos de prueba.

El criterio IIE se cumpliría para los tipos no válidos si a partir de la totalidad de los casos de prueba con ese tipo de resultado esperado la MII es capaz de generar un programa correcto de validación de los tipos de la entrada. La implementación de este enfoque queda por fuera del alcance de este trabajo.

7.3 Evaluación del grado de cumplimiento

Cada uno de los programas generados se ejecutó con el CCP **RNDO.** La evaluación de los resultados se realizó utilizando una planilla electrónica. Los resultados se presentan en el Cuadro 8, ordenados por grado de cumplimiento de mayor a menor.

Programa generado a partir de CCP Sujeto	Resultados Correctos	Resultados Incorrectos	% Correctos
7	1103	0	100
8	1103	0	100
11	1103	0	100
3	983	120	89,12
2	896	207	81,23
9	896	207	81,23
1	453	650	41,07
6	161	942	14,60
5	156	947	14,14
4	122	981	11,06
10	43	1060	3,72

Cuadro 8: Grado de cumplimiento con entrada aleatoria

El análisis del cuadro 7 muestra claramente que los CCP que cumplen el criterio IIE cubren más condiciones y presentan más casos que aquellos que no lo cumplen. Al analizar el cuadro 8 se detectan 5 grupos de resultados:

1) Los CCP 7, 8 y 11 que cumplen IIE

2) Los CCP 3, 2 y 9 con un porcentaje de resultados correctos por encima del 80%

3) El CCP 1 con un porcentaje del 41%

4) Los CCP 6, 5 y 4 con porcentajes de entre un 10 y 15

5) El CCP 10 con un porcentaje por debajo del 5

Resulta llamativo que el CCP 4 obtenga un grado IIE tan bajo con entradas aleatorias, muy por debajo del obtenido por el CCP 3 que tiene la misma cantidad de casos por categoría y por debajo incluso de los obtenidos por los CCP 1, 6 y 5, siendo que contiene más casos de prueba que cada uno de ellos.

Cada uno de estos CCP solo contiene un único caso con valores desordenados. Adicionalmente, cada uno contiene la lista vacía. Los CCP 1,3 y 4 contienen también un caso con datos ordenados y el CCP 4 uno

con un elemento. El mejor programa generado por ADATE para el CCP 4 está muy lejos de ordenar una lista de valores de entrada. El correspondiente al CCP 6 es mucho mejor a pesar de que se podría suponer que el CCP 6 contendría menos información. Ambos contienen la lista vacía y una lista desordenada de largo 4. El CCP 4 contiene además un caso con una lista de largo 1 y un caso de largo 2 con la lista ordenada.

Si consideramos las clases con "igual información", cualquier elemento de la clase se puede tomar como representante de la misma. Para que salte a la vista la información relacionada al ordenamiento, se puede tomar como representante de cada una de estas clases a la conformada por el valor mínimo 0 y los valores consecutivos subsiguientes. Por ejemplo, para la clase de la lista [7,10,3,10,5,2,7] se toma como representante de la clase a: [3,4,1,4,2,0,3]. La primera lista está formada por los valores distintos [2,3,5,7,10], a los que les hacemos corresponder los valores [0,1,2,3,4].

CCP de Sujeto	**Casos Adicionales Necesarios**	**Cantidad Total de Campos Adicionales**	**Grado de Cumplimiento de IIE respecto a CCP de referencia a partir de lista aleatoria (24) (en porcentaje)**
7		0	100
8		0	100
11		0	100
2	[1,10,9],[1,9,10]	6	75
3	[1,10,9],[1,9,10]	6	75
9	[1,10,9],[1,9,10]	6	75
1	[1,10,9],[1,9,10]/[2,7,1],[1,2,7]	12	50
4	[10,1],[1,10]/[1,10,9],[1,9,10]/ [10,5,1],[1,5,10]	16	33
6	[10,1], [1,10]/[1,10],[1,10]/[1,1], [1,1]/[1,10,9],[1,9,10]	18	25
5	[10,1],[1,10]/[1,10],[1,10]/ [1,10,9],[1,9,10]/[10,1,2],[1,2,10]	20	17
10	Por más casos que se agreguen seguirá siendo incorrecto	Infinito	0

Cuadro 9: Grado de cumplimiento por cantidad de casos adicionales

Si se normalizan los casos de prueba desordenados de los CCP 4 y 6 con el criterio anterior, se obtienen los casos:

4: [2,0,1,0], [0,0,1,2]

6: [3,2,1,0], [0,1,2,3]

Salta a la vista que en el CCP 4 los dos elementos del medio permanecen en la salida mientras que en el CCP 6 todos los elementos cambian de lugar.

Si se consideran los CCP 4 y 3, las entradas normalizadas de los casos de prueba desordenados quedan como sigue:

3: [3,2,3,1,0,2], [0,1,2,2,3,3]

4: [2,0,1,0], [0,0,1,2]

En el CCP 3 todos los valores cambian de lugar en la salida y este CCP contiene casos que cubren las mismas categorías que el CCP 4. Sin embargo sus grados por valores aleatorios y por campos adicionales son de los más altos que no cumplen IIE. Un nuevo indicio de que la cantidad de información brindada por listas de entrada desordenadas puede ser muy distinta.

En el cuadro 9 se presentan los cálculos del grado de cumplimiento en base a la cantidad de casos adicionales respecto al CCP **REF**.

7.4 Conclusiones sobre la evaluación del grado de cumplimiento de IIE

La evaluación del grado de cumplimiento a partir de la proporción de resultados correctos del programa generado para un conjunto de datos de casos de prueba aleatorio obtenido a partir de cierta distribución, independientemente de cuál sea la distribución elegida, va a brindar valores 100 para programas que cumplen IIE y valores 0 para programas que siempre arrojan resultados incorrectos. Los valores intermedios en general van a depender de la distribución elegida para los valores de entrada. La elección de esta distribución podría tener impacto significativo sobre los grados de cumplimiento que se observen. Una vez que se cuenta con el CCP generado de forma aleatoria y se ordena de forma adecuada, este puede ser usado para determinar un CCP de referencia. De las distintas maneras de medir el grado de cumplimiento de un CCP utilizando el CCP de referencia se podrían utilizar tanto la cantidad de caracteres como la cantidad de campos. La elección de uno u otro enfoque puede depender de las características de la entrada-salida del programa a probar.

En el cuadro 10 se presentan los grados obtenidos con CCP aleatorio para los CCP que no cumplen IIE, exceptuando el del sujeto 10 y a su lado los obtenidos a partir de los campos de entrada-salida adicionales para cumplir IIE respecto al CCP de referencia, ordenados de mayor a menor por este último. El ordenamiento de los valores resulta similar, excepto para el caso del CCP 4.

CCP de Sujeto	Grado por prueba con CCP aleatorio	Grado por campos adicionales necesarios respecto a CCP de referencia
3	89,1	75
2	81,2	75
9	81,2	75
1	40,96	50
4	10,9	33
6	14,44	25
5	13,99	17

Cuadro 10: Grado de cumplimiento por cantidad de casos adicionales

La primera de estas formas de medir el grado evalúa la bondad del programa generado para la MII. La segunda en cambio, toma en cuenta la cantidad de información adicional necesaria para obtener un CCP que cumpla IIE. Los CCP 5 y 6 solo contienen 2 casos de prueba con entradas válidas, en ambos conformados por lista vacía y lista desordenada de largos 5 y 4 respectivamente. El CCP 4 está conformado por 4 casos de prueba: lista vacía, largo 1, largo 2 ordenada, largo 4 desordenada con repetidos. Este CCP sin duda brinda mucho más información que los de 5 y 6, por lo que resulta natural que la cantidad de información

adicional necesaria sea menor que para estos últimos. Si se dispone de un oráculo, resulta más fácil de evaluar el grado de cumplimiento con el primer enfoque, pero el segundo mide la cantidad de información adicional necesaria para cumplir con IIE, por lo que resulta más adecuado para evaluar el grado de cumplimiento del criterio. A partir de las mediciones hechas podemos afirmar que el CCP 3 resulta algo mejor que el 2 y 9 y estos tres son de los mejores que no cumplen IIE, el CCP 1 está en un nivel intermedio y los CCP 4, 6 y 5 son los que están más lejos de cumplir IIE.

7.5 Evaluación de la aplicación de la técnica

A partir de los resultados obtenidos, resulta claro que los conjuntos de casos de prueba propuestos por los distintos sujetos presentan una gran dispersión en cuanto a la cantidad de casos de prueba, las clases válidas consideradas y el cumplimiento del criterio IIE. Un fenómeno análogo ya fue observado por Briand et al. en la aplicación de la técnica Category Partition [BRI07, BRIA07].

La dispersión en los grados de cumplimiento del criterio IIE no nos permite afirmar que la técnica de partición en clases de equivalencia con análisis de borde fue aplicada de forma inadecuada ya que el criterio IIE no está directamente asociado a esta técnica.

La aplicación inadecuada de la técnica puede dar origen a CCP pobres por un lado o excesivos por otro, situaciones que denominaremos respectivamente de defecto o de exceso. Cualquiera de estas condiciones puede tener impacto sobre la evaluación de la técnica en un experimento.

Corresponde por lo tanto hacer dos evaluaciones:

a) evaluar si los CCP que presentan un grado más bajo de cumplimiento del criterio IIE están asociados a una aplicación inadecuada de la técnica y

b) evaluar si alguno de los CCP que sí cumplen el criterio IIE de forma estricta incluyen casos de prueba “adicionales” respecto a lo que la técnica establece.

7.6 CCP que no cumplen IIE

Para los CCP que no cumplen IIE de forma estricta, podría darse una de las situaciones siguientes con respecto a la relación entre los grados de cumplimiento del criterio IIE observados y la calidad en la aplicación de la técnica:

a) en los casos en que el grado de cumplimiento de IIE es bajo la técnica se aplicó de forma inadecuada o

b) la dispersión en los grados de cumplimiento de IIE no guarda relación alguna con la calidad en la aplicación de la técnica.

Vamos a analizar la calidad en la aplicación de la técnica en cada uno de los tres grupos obtenidos a parir de la evaluación del grado de cumplimiento de IIE.: (3,2,9),(1), (4,5,6).

El grupo intermedio tiene el CCP 1 como único integrante. El CCP está compuesto por 3 casos de prueba: lista vacía, lista ordenada de 3 elementos sin repetidos, lista desordenada de 3 elementos sin repetidos. Llama la atención que no se haya incluido el largo de lista 1 como un caso de borde ni la existencia de repetidos como una clase de equivalencia especial. Por cada clase hay tan solo un caso de prueba que lo cubre. El análisis de borde exigiría incorporar más de uno, en especial para listas de largo mayor a uno.

El grupo peor evaluado está compuesto por los CCP 4, 5 y 6. Para los dos últimos resulta claro que la técnica no fue aplicada de forma adecuada ya que tan solo se identificaron las clases lista vacía y lista no vacía desordenada y no hay ningún caso correspondiente a una condición de borde. Respecto al CCP 4, las clases (no disjuntas) identificadas serían: lista vacía, lista con un elemento, lista con dos elementos, lista con varios elementos, lista ordenada, lista desordenada. Algunos de los casos de prueba cubren más de una de estas clases. Lo que no parece haber es un análisis de condiciones de borde de cada una de las clases.

Para los dos grupos peor evaluados resulta claro que la técnica no fue aplicada de forma adecuada.

El grupo mejor evaluado está compuesto por los CCP 3, 2 y 9. A partir de cada uno de ellos, al agregar el caso adicional [1,10,9],[1,9,10] se logró cumplir el criterio IIE.

Instancia	Vacía	Un elemento	Desordenada Sin Reps.	Desordenada Con Reps.	Total desordenados	Ordenada Sin Repetidos	Ordenada Con Repetidos	Total de casos	Largos de lista	Programa Generado Correcto
11	1		2	1	3	1		5	0,3	S
2	1		1	1	2	1	1	5	0,4,5	N
9	1	1	1	1	2		1	5	0,1,5,13	N
3	1	1		1	1	1		4	0,1,6	N

Cuadro 11: Características de los CCP 2, 3, 9 y 11

En el cuadro 11 se presenta la cantidad de casos correspondiente a cada clase para estos CCP junto con el CCP 11 de forma de permitir compararlos. Del análisis del CCP 3 se infiere que las clases de equivalencia identificadas serían: lista vacía, lista con un elemento, lista desordenada, lista ordenada. No se toma en cuenta la existencia de valores repetidos. Solo incluye un caso de prueba con datos de entrada desordenados por lo que esta clase no está representada de acuerdo a las exigencias del análisis de borde.

Para el CCP 9 las clases identificadas serían: lista vacía, lista con un elemento, lista desordenada, lista ordenada, lista con repetidos, lista sin repetidos. Llama la atención que el representante de la clase "lista ordenada con repetidos" es una lista de 5 veces el valor 4. No hay ningún caso con una lista ordenada con valores distintos, por lo que o bien la elección del caso no sería adecuada o fue otra la clase identificada que responde a este caso. Los casos correspondientes a las clases desordenadas indican que no se realizó análisis de borde para estas clases. También resulta llamativo que los dos casos de prueba desordenados tienen cada uno 13 elementos. Parece que la elección hubiera estado guiada por la regla heurística según la cual a mayor largo y complejidad del caso, mayor probabilidad de detectar defectos. Independientemente de que esta regla sea o no válida, es claro que es ajena a la técnica "partición en clases de equivalencia con análisis de borde". Los resultados obtenidos con el CCP utilizado para la prueba de generación, en que varios programas inferidos ordenaron el caso de largo 6 de forma correcta y fallaron con casos de largo 3 y en especial los resultados con el programa inferido a partir del propio CCP 9 que solo falló en los casos (1,3,1) y (1,3,2) permiten poner en duda la validez de esa regla.

Para el CCP 2 las clases identificadas serían: lista vacía, lista desordenada, lista ordenada, lista con repetidos y lista sin repetidos. Al igual que para el CCP 9, los casos correspondientes a las clases desordenadas indican que no se realizó análisis de borde de estas clases.

En el cuadro 12 se presenta el detalle de los CCP correspondientes.

Los tipos de clases consideradas para el CCP 11 que cumple con IIE resulta muy parecido al correspondiente al CCP 2 que no lo cumple. La diferencia parece estar en la cantidad de casos de prueba desordenados y en la cantidad de información adicional. En el CCP 2 los casos de prueba desordenados tienen el valor máximo uno al principio y otro al final. No hay un caso con el valor máximo en el medio. El programa generado falla en la prueba de generación justamente en casos en que el valor máximo está en el medio. El CCP 11 contiene casos con máximo y mínimo al final, en el medio y al principio.

Sujeto	Parejas entrada-salida	Evaluación con casos de prueba	Largos en lista de entrada válidas
2	[],[]/[4,2,3,1,3],[1,2,3,3,4]/[1,2,3,3,4],[1,2,3,3,4]/ [2,3,1,4],[1,2,3,4]/[1,2,3,4],[1,2,3,4]	Ordena mal [1,3,2] y [1,3,1]	0,5,4,
3	(*)/[],[]/[1],[1]/[0,1,2,3,4,5],[0,1,2,3,4,5]/ [5,4,5,2,1,4],[1,2,4,4,5,5]/	Ordena mal [1,3,2]	0,1,6
9	(*)/([b,c,a])/[],[]/[1],[1]/[4,-2,0,15,3,-10,10,-1,34,-50,5,-87,8],[-87,-50,-10,-2,-1,0,3,4,5,8,10,15,34]/ [50,-2,0,1,3,10,-1,-50,-1,0,8,-3,50],[-50,-3,-2,-1,-1,0,0,1,3,8,10,50,50]/[4,4,4,4,4],[4,4,4,4,4]	ordena mal [1,3,2],[1,3,1]	0,1,13,5
11	[],[]/[1,2,4],[1,2,4]/[1,3,2],[1,2,3]/[2,1,2],[1,2,2]/ [2,1,-3],[-3,1,2]/(*)	Verifica la prueba	0,3

Cuadro 12: Detalle de los CCP 2, 3, 9 y 11 y fallas con CCP de control

Podemos afirmar que para los CCP que no cumplieron IIE, la técnica no fue aplicada de forma adecuada y el menor grado de cumplimiento se corresponde con una peor aplicación.

7.7 CCP que sí cumplen IIE

Para los CCP que sí cumplen IIE de forma estricta, podría darse una de las situaciones siguientes:

a) para todos estos CCP la técnica fue aplicada de forma adecuada (sin defectos) y sin excesos o
b) en alguno de estos CCP la técnica fue aplicada con defectos y/o excesos (habría que agregar algunos casos y/o suprimir otros).

En el Cuadro 13 se presentan los datos para los CCP 7, 8 y 11 y en el cuadro 14 se repiten los análisis de los CCP 8 7 y 11 para las clases identificadas.

También al interior de este grupo existe una gran dispersión en la cantidad de casos de prueba y largos de listas considerados.

Sujeto	Parejas entrada-salida	Evaluación	Largos entrada
7	(*)/[],[]/[0.01],[0.01]/[-1],[-1]/[0,0],[0,0]/[1,3,2],[1,2,3]/[1,2,4,5,2,2],[1,2,2,2,4,5]/[4,3,2,1,1],[1,1,2,3,4]/[1,3,2,1,2,3],[1,1,2,2,3,3]/[1,1,2,2,3,3,],[1,1,2,2,3,3]	Verifica la prueba	0,1,3,6,5
8	[],[]/[1,-1],[-1,1]/[2,0],[0,2]/[0,-2],[-2,0]/[4,2],[2,4]/[-2,-4],[-4,-2]/[5,6],[5,6]/[-4,-3],[-4,-3]/[0,2],[0,2]/[-2,0],-[2,0]/[2,4,-3,-1],[-3,-1,2,4]/[3,1,5,0],[0,1,3,5]/[-4,0,-1,-5],[-5,-4,-1,0]/[5,1,3,2],[1,2,3,5]/[-5,-1,-4,-3],[-5,-4,-3,-1]/[1,2,3,4],[1,2,3,4]/[-4,-3,-2,-1],[-4,-3,-2,-1]/[0,2,4,6],[0,2,4,6]/[-3,-2,-1,0],[-3,-2,-1,0]/[2],[2]/[0],[0]/[-2],[-2]/[-2,3,-1],[-2,-1,3]/[2,-1,3],[-1,2,3]/[-1,0,-2],[-2,-1,0]/[0,3,2],[0,2,3]/[-2,-4,-1],[-4,-2,-1]/[4,1,3],[1,3,4]/[-4,-2,3],[-4,-2,3]/[-2,1,3],[-2,1,3]/[-4,-2,0],[-4,-2,0]/[0,2,4],[0,2,4]/[-3,-2,-1],[-3,-2,-1]/[1,3,5],[1,3,5]/(*)/(4294967296)/(-4294967296)	Verifica la prueba	0,2,4,1,3
11	[],[]/[1,2,4],[1,2,4]/[1,3,2],[1,2,3]/[2,1,2],[1,2,2]/[2,1,-3],[-3,1,2]/(*)	Verifica la prueba	0,3

Cuadro 13: Detalle de los CCP 7, 8 y 11

Instancia	Vacía	Un elemento	Desordenada Sin Reps.	Desordenada Con Reps.	Total desordenados	Ordenada Sin Repetidos	Ordenada Con Repetidos	Total de casos	Largos de lista	Programa Generado Correcto
8	1	3	17		17	12		33	0,1,2,3,4	S
7	1	1	1	4	5		1	8	0,1,2,3,5,6	S
11	1		2	1	3	1		5	0,3	S

Cuadro 14: Análisis de los CCP 7, 8 y 11 para las clases identificadas

Resalta el CCP 8 compuesto por 33 casos y con largos de lista 0,1,2,3,4. Las clases (no disjuntas) identificadas parecen ser para lista: vacía, con un elemento, con dos elementos, con valores positivos, con valores negativos, ordenadas, desordenadas. Cada una de estas clases está representada en muchos casos, pero esos muchos casos no parecen responder a un análisis de borde de la clase. Se nota una preocupación por repetir el mismo patrón con valores positivos, negativos y cero. Los casos se pueden agrupar en función de la cantidad de elementos de la lista y del ordenamiento relativo en:

- lista vacía (1 caso)
- patrón [1] (3 casos)
- patrón [1,2] (4 casos)
- patrón [2,1] (5 casos)
- patrón [1,2,3] (6 casos)
- patrón [1,3,2] (2 casos)
- patrón [2,1,3] (1 caso)
- patrón [2,3,1] (1 caso)
- patrón [3,1,2] (1 caso)
- patrón [1,2,3,4] (4 casos)

- patrón [1,4,2,3] (1 caso)
- patrón [2,4,,3,1] (1 caso)
- patrón [3,2,4,1] (1 caso)
- patrón [3,4,1,2] (1 caso)
- patrón [4,1,3,2] (1 caso).

En este análisis llama la atención la cantidad de casos correspondientes a los patrones [1], [1,2], [2,1], [1,2,3] y [1,2,3,4]. Solo para el primero de estos patrones aparecen valores muy grandes y muy pequeños. Para el resto resulta difícil encontrar justificación para la cantidad de casos en base a la técnica Partición en Clases de Equivalencia con Análisis de Borde. Por otra parte, se nota una preocupación por considerar de forma sistemática las distintas configuraciones posibles de ordenamiento relativo en listas de 3 y 4 elementos, lo que sí puede asociarse a la técnica.

Del CCP 7 se infiere que las clases identificadas son: lista vacía, lista con un elemento, lista con tres elementos, lista con más elementos, lista ordenada, lista desordenada, lista con repetidos, lista sin repetidos. En los casos del prueba el valor máximo figura al comienzo, en el medio y al final. El valor mínimo al principio y en el medio, en ningún caso al final.

El CCP 11 ya fue analizado en la sección anterior.

En los tres CCP que cumplen el criterio se nota una preocupación por situaciones límite en el orden relativo de los elementos de la lista. El CCP 11 no denota preocupación por distintos largos de lista. En ninguno de los tres aparece preocupación por los valores mínimos y máximos de cada elemento, lo que sí está considerado en CCP que no cumplen el criterio.

7.8 Conclusiones de la aplicación del criterio IIE

Del análisis de los CCP queda claro que estos presentan una gran dispersión en sus características y que estas se reflejan en el grado de cumplimiento del criterio. No es posible establecer una correspondencia estricta entre el cumplimiento del criterio y la satisfacción de la técnica. Un grado bajo de cumplimiento aparece asociado a una aplicación de la técnica inadecuada. El cumplimiento del criterio no necesariamente implica una aplicación de la técnica adecuada. En un caso de cumplimiento del criterio se identificó un número de casos importante que no son atribuibles a la aplicación de la técnica en estudio. Este tipo de situaciones podría afectar la validez interna de un experimento que tenga por objetivo evaluar la efectividad de la técnica.

7.9 Comparación de resultados de IIE y Cobertura de Sentencias

En el experimento de Basili para evaluar la efectividad de técnicas de selección de casos de prueba, se midió la cobertura de sentencias para los casos de prueba generados mediante técnicas de caja negra.

Para evaluar en qué medida este criterio de aceptación puede ser un indicador válido de si una técnica de caja negra fue aplicada de forma adecuada o no, procedimos a aplicar este criterio a los CCP generados por los sujetos en el experimento. Para ello procedimos a ejecutar el programa Java a probar como parte del experimento en un ambiente Eclipse Java EE IDE for Web Developers, Versión Mars.1 Release (4.5.1). En la medición de cobertura de sentencias se utilizó el Plug-In EclEmma Java Code Coverage Version

2.3.2.201409141915. En el Cuadro 15 se presentan los valores obtenidos con cobertura de sentencias junto con los ya presentados para el criterio IIE.

En 9 de los 11 CCP la cobertura de sentencias toma el valor máximo de 100, incluso para los CCP 5 y 6 para los que la aplicación de la técnica fue particularmente pobre. Para este CCP la cobertura de sentencias no resultó un indicador adecuado de la calidad en la aplicación de la técnica de Partición en Clases de Equivalencia con Análisis de Borde.

CCP de Sujeto	Cobertura de Sentencias	Grado IIE con entrada aleatoria	Grado IIE por Cantidad de Casos Adicionales
7	100	100	100
8	100	100	100
11	100	100	100
3	62,7	89,1	71
2	100	81,2	71
9	100	81,2	71
1	100	40,96	43
6	100	14,44	14
5	100	13,99	5
4	62,7	10,9	24
10	100	3,7	0

Cuadro 15: Grados de Cobertura de Sentencias y de IIE

7.10 Factores de contexto

Resulta interesante contrastar la calidad obtenida en la aplicación de la técnica con los factores de contexto de esa aplicación resumidos en la figura 1. En el cuadro 16 se detallan esos factores.

De su revisión se destaca que:

- El contexto de aplicación fue un curso, sin incentivos para los estudiantes
- El programa a probar era pequeño, simple con tecnología y en un dominio de aplicación conocido
- No se cuenta con información respecto a la eventual experiencia profesional de alguno de los estudiantes
- No fue evaluada la habilidad para generar casos de prueba que revelen defectos
- No fue evaluado el sesgo positivo, pero los resultados obtenidos mostraron un marcado sesgo negativo, se presume que originado por el entrenamiento
- Los sujetos contaron con información de los defectos detectados, no así de la cobertura lograda.

La dispersión de los resultados y el fuerte sesgo negativo de algunos sujetos parecen estar originados en la dispersión en la efectividad del entrenamiento. Es posible suponer que la introducción de incentivos asociados a la calidad en la aplicación de la técnica e información de cobertura con un criterio exigente (incluyendo IIE) podrían haber mejorado la calidad en la aplicación de la técnica.

Factor	En el experimento
Contexto de aplicación	Curso
Incentivos	No se dieron. La calidad en la aplicación de la técnica no tuvo incidencia en la calificación
Efectividad del entrenamiento	No fue evaluada
Características del programa	
Tamaño	Pequeño
Complejidad	Simple
Tecnología	Java, conocida para los participantes
Dominio	Abstracto, ordenación de listas
Relación con la especificación	No evaluada
Características de los defectos	No evaluadas
Características de los sujetos	Estudiantes
Experiencia	
Profesional	No hay información respecto a si alguno tenía experiencia profesional
Lenguaje	Con experiencia en el lenguaje
Dominio de aplicación	Con experiencia en el dominio
Habilidad para generar casos que revelen defectos	No evaluada
Sesgo positivo	Los resultados indican para varios sujetos un marcado sesgo negativo, se presume que originado por el entrenamiento
Información en el proceso	
Cantidad de defectos detectados	Sí
Cobertura lograda	No
Uso de la técnica para identificar casos / criterio de aceptación	Para identificar casos ya que no hay criterio de aceptación para la técnica

Cuadro 16: Factores de contexto en el experimento

Capítulo 8: Análisis y evaluación de IIE

En este capítulo se analiza y evalúa el criterio IIE a partir de los resultados obtenidos, se aborda el estudio de la noción relacionada de clases con "igual información" y se plantean posibles usos y aplicaciones del criterio.

8.1 Grado de IIE y la detección de defectos

Una forma de evaluar los CCP generados con enfoque de caja negra para un especificación S consiste en aplicarlos para detectar defectos en diversos programas incorrectos respecto a esa especificación. Como resultado de la aplicación del criterio IIE hemos obtenido un conjunto de programas incorrectos y una serie de CCP que cumplen el criterio. Vamos a evaluar si estos CCP permiten detectar que esos programas son incorrectos.

Tenemos tres CCP obtenidos en los experimentos que cumplen IIE (CCP 7, 8 y 11) y cuatro CCP creados en el proceso de búsqueda del CCP con mínimo de casos (Min2, Min3, Min4, Min5). Se dispone de ocho programas incorrectos, los generados por ADATE para los CCP 1, 2, 3, 4, 5, 6, 9, 10 que no cumplen IIE. En el Cuadro 17 para cada uno de estos CCP en la columna correspondiente a cada uno de esos programas un 1 significa que el CCP genera al menos una falla en ese programa y un 0 significa que no genera ninguna falla.

CCP	Casos	Falla Prog 1	Falla Prog 2	Falla Prog 3	Falla Prog 4	Falla Prog 5	Falla Prog 6	Falla Prog 9	Falla Prog 10	Programas que fallan
7	8	1	1	1	1	1	1	1	1	8
8	34	1	1	1	1	1	1	1	1	8
11	5	1	1	1	1	1	1	1	1	8
Min2	4	1	1	1	1	1	1	1	1	8
Min3	3	1	1	1	1	1	1	1	1	8
Min4	3	1	1	1	1	1	1	1	1	8
Min5	2	1	1	1	1	1	1	1	1	8
1	3	0	0	0	1	1	1	0	1	4
2	5	1	0	0	1	1	1	0	1	5
3	4	1	0	0	1	1	1	0	1	5
4	4	0	0	0	0	1	1	0	1	3
5	2	1	0	0	1	0	1	0	1	4
6	2	0	0	0	1	1	0	0	1	3
9	5	1	0	0	1	1	1	0	1	5
10	2	0	0	0	1	1	1	0	1	4

Cuadro 17: Análisis de los CCP 7, 8 y 11 para las clases identificadas

Los CCP que cumplen IIE detectan fallas en todos los programas incorrectos generados por ADATE. Lo mismo sucede con los CCP generados en la búsqueda de CCP con mínimo de casos. En tanto que los CCP a

partir de los que ADATE generó programas incorrectos solo permiten detectar defectos en a lo sumo 5 de los 8 programas.

8.2 Efectividad teórica para detectar defectos

El resultado anterior va a ser analizado desde un punto de vista teórico. Sea M una MII que posee la propiedad de hipótesis más simple.

Teorema 1. Dado un CCP T para una especificación S y un programa P' generado por M mediante inferencia inductiva a partir de T. Entonces T revela al menos una falla en todo programa P'' generado por M a partir de T tal que satisface P'' es más simple que P'.

Prueba: Si P'' no falla en ningún caso de T entonces P'' es más simple que P' y satisface T en contradicción con la propiedad de hipótesis más simple de M con entrada T.

El hecho de que una MII genere un programa incorrecto a partir de un CCP puede ser interpretado como falta de información en el CCP respecto a la especificación. Es por esto que el programa más simple corresponde a otra especificación. Si se toma como medida de complejidad de una especificación la complejidad del programa más simple que la satisface, entonces esa otra especificación es una más simple. Esto establece una relación entre la cantidad de información proporcionada por un CCP respecto a la especificación y la complejidad de la especificación.

La relación entre CCP "M infiere a partir del CCP X un programa con la misma complejidad que el programa inferido a partir del CCP Y" es una relación de equivalencia en el conjunto de CCP para la especificación S, por lo que establece una partición de este conjunto en clases de equivalencia. Cualquier CCP de una de esas clases revela al menos un defecto en todo programa inferido a partir de CCP que corresponden a programas de menor complejidad.

Vale la pena notar que programas con la misma complejidad pueden brindar salidas distintas cuando son alimentados con la misma entrada.

Teorema 2. Dado un CCP T que satisface IIE para la especificación S con M, esto es, existe un programa P' generado por M a partir de T que es correcto respecto a S. Entonces todo programa P'' generado por M a partir de CCP T' consistente con S se cumple que P' no es más simple que P''.

Prueba. P' es correcto respecto a S. Entonces P' satisface T' consistente con S. P' es más simple que P'' contradice la propiedad de hipótesis más simple de M con entrada T'.

El teorema 2 establece que para una especificación S la complejidad de los programas inferidos por M es como máximo la complejidad de un programa correcto.

Teorema 3. Dado un CCP T que satisface IIE para la especificación S con M, entonces existe un programa P' generado por M que es correcto respecto a S. Entonces T revela al menos una falla en todo programa generado por M a partir de CCP T' que no satisface IIE y es más simple que P'.

Prueba: Por aplicación directa del teorema 1.

El teorema 3 proporciona una generalización de los resultados observados al aplicar el criterio IIE.

Corresponde notar que una MII puede ser utilizada para generar programas incorrectos que pueden resultar útiles en la evaluación de CCP.

8.3 Clases de equivalencia con igual información en los casos de prueba aleatorios

Para ilustrar el concepto de Clases con Igual Información, se procede a analizar la cantidad de clases de ese tipo presentes en el conjunto de casos de prueba aleatorios generados para conformar el CCP RNDO. El conjunto está compuesto de 1103 casos. Cada clase de este tipo está integrada por casos de prueba que suministran exactamente la misma información a la MII. Para el programa de ordenación esto corresponde a listas del mismo largo con elementos con el mismo ordenamiento relativo.

Se transformaron los elementos de este conjunto de manera análogo a lo hecho en 7.3. Cada caso se sustituye por otro caso en el que el valor mínimo original se sustituye por el valor 0 y el resto por los valores consecutivos subsiguientes. Con esta transformación, a partir de los 1103 datos de entrada aleatorios, se obtuvieron 633 listas ejemplo distintas, poco más del 57% de los casos. Esto muestra que para un programa que ordena listas de enteros, valores de entrada aleatorios suministran información distinta en una elevada proporción de los casos.

8.4 Clases de equivalencia con igual información en la prueba de correctitud

El CCP **CORR** utilizado en la prueba de correctitud contiene casos con listas de largo 0, 1, 2, 3 y 6. En este CCP cada caso de prueba corresponde a una clase distinta de las clases con igual información y están cubiertas todas las clases de equivalencia con igual información para los largos de lista 0,1 y 3. La cantidad de clases distintas para listas de largo 0 es 1, para las de largo 1 es 1, para las de largo 2 es 3 y para las de largo 3 es 13.

8.5 Evaluación automática del grado de IIE con entradas aleatorias

Si se dispone de un oráculo O para la especificación S tal que para cualquier valor de entrada devuelve el resultado esperado de acuerdo a S, el grado de IIE con entradas aleatorias puede ser evaluado de forma automática de acuerdo al procedimiento siguiente.

a) Generar entradas aleatorias i de acuerdo a S

b) Completar las entradas i con las salidas esperadas utilizando O para formar casos de prueba de la forma [i, O(i)] para obtener el CCP T.

c) La MII M genera un programa P’ a partir de T.

d) Ejecutar P’ con nuevas entradas aleatorias i’.

e) Evaluar las salidas de P’ para las entradas i’ con el oráculo O.

Este procedimiento permite que el criterio IIE, a pesar de ser de caja negra, pueda ser evaluada experimentalmente sin la participación de sujetos que implementen la técnica. Para ello basta con aplicar un

procedimiento análogo al utilizado por Frankl y Weiss en su experimento destinado a comparar la efectividad de “branch testing” y “data flow testing”.

8.6 Ejecución de programas generados por una MII con casos con “igual información”

Si una MII M posee la propiedad de hipótesis más simple, la siguiente conjetura resulta plausible.

Conjetura 1: Dado un programa P’ generado por una MII M a partir de un CCP T consistente con la especificación S y dados dos casos de prueba diferentes C_1 y C_2 pertenecientes a la misma clases con igual información relevante para S, las trayectorias de ejecución de ambos casos en P’ serán idénticas.

Justificación: Si la trayectoria de C_1 en P’ fuera diferente de la trayectoria en P’ de C_2 y dado que ambos casos de prueba debieran ser sometidos al mismo tratamiento, sería posible obtener un programa P’’ más simple que P’ haciendo que la trayectoria sea la misma, en contradicción con la hipótesis más simple.

Si esta conjetura resultara cierta, sería posible relacionar el criterio IIE con criterios de aceptación de caja blanca sobre los programas inferidos por la MII M.

Capítulo 9: Conclusiones y trabajo futuro

En este capítulo se presentan las conclusiones, los aportes del trabajo, sus limitaciones y los trabajos planteados a futuro.

9.1 Conclusiones

La selección de casos de prueba tiene importancia en la efectividad de la verificación de software. Desde hace más de 30 años se han llevado a cabo experimentos para evaluar la efectividad de distintas técnicas. Los experimentos llevados a cabo hasta el momento no han permitido extraer conclusiones claras respecto a la efectividad de las distintas técnicas. Las técnicas de selección de caja negra exigen la participación de sujetos que apliquen esa técnica. Esos experimentos suelen responder al esquema general:

a) selección de sujetos
b) entrenamiento a los sujetos en el uso de las distintas técnicas
c) los sujetos aplican las técnicas para detectar defectos en un conjunto de programas
d) la cantidad de los defectos detectados utilizando determinada técnica es utilizada para evaluar la efectividad relativa de cada técnica.

Se analizaron múltiples factores de contexto que pueden incidir en los resultados que obtienen sujetos que aplican una técnica de selección de casos de prueba. Se elaboró un esquema que muestra las relaciones entre esos factores y su incidencia en la cantidad de defectos detectados. Se remarca la distinción entre defectos “revelados” por un CCP y los “detectados” por los sujetos. Este análisis muestra la conveniencia de incluir en los experimentos en que participen sujetos que aplican técnicas de selección de casos de prueba que tengan por objetivo evaluar la efectividad de esas técnicas, la evaluación de la calidad de la aplicación de la técnica por parte de los sujetos.

Se definió el criterio de aceptación IIE que depende de la especificación y no del código del programa, por lo que resulta a priori adecuado para técnicas de selección de caja negra.

Se aplicó el criterio IIE para evaluar la calidad de los CCP generados por sujetos en un experimento para evaluar la efectividad de técnicas de selección de casos de prueba de caja negra. La aplicación de este criterio mostró una gran dispersión en los grados de cumplimiento del criterio y esta misma dispersión se correspondió con el análisis cualitativo de esos CCP. En la aplicación a estos CCP del criterio de aceptación Cobertura de Sentencias se obtuvieron resultados con mucho menor dispersión y casi sin relación con el análisis cualitativo. Este resultado resulta alentador respecto a que el criterio puede resultar útil en la evaluación de la calidad de la aplicación de técnicas de selección de casos de prueba de caja negra.

El análisis detallado de los CCP generados en el experimento reafirmó la conveniencia de evaluar la calidad en la aplicación de las técnicas de selección de casos de prueba en los experimentos.

9.2 Aportes del trabajo

Se elaboró un diagrama que muestra las relaciones entre factores que inciden en la detección de defectos. Este diagrama puede ser utilizado para facilitar el control de estos factores en la realización de expe-

rimentos destinados a evaluar la efectividad de técnicas de selección de casos de prueba y para analizar los factores que inciden en la detección de defectos en la industria.

Se definió el criterio de aceptación IIE que solo depende de la especificación. Se analizaron distintas alternativas para evaluar el grado de cumplimiento del criterio y se eligieron dos como las más adecuadas. Se mostró como puede ser aplicado IIE utilizando el sistema ADATE y que su aplicación resulta técnicamente factible con las condiciones actuales de la tecnología, por lo menos en un ámbito académico.

Se introdujo la noción de "clase de equivalencia con igual información relevante para la especificación" relacionada con la información que suministra cada caso de prueba a la MII. Esta noción merece un análisis más profundo.

Se aplicó el criterio IIE a CCP generados en un experimento destinado a evaluar la efectividad de técnicas de selección de casos de prueba de caja negra. La aplicación mostró una gran dispersión en los resultados. Este fenómeno ya había sido observado en la aplicación de otras técnicas de caja negra. El análisis de las mediciones mostró que los CCP con mayor grado IIE al considerar sus características (cantidad y características de los casos) resultaron mejores que los que poseen menor grado.

Se mostró que los CCP que cumplen IIE permiten generar al menos una falla en todos los programas incorrectos inferidos por ADATE como MII. Este resultado se generalizó desde el punto de vista teórico. Se mostró que un CCP T consistente con la especificación S y para el que una MII M generó un programa P' a partir de T permite revelar al menos una falla en todo programa P'' más simple que P'.

Se mostró que la cantidad de información relevante para la especificación en un CCP tiene impacto en la detección de defectos. Dada una MII M que posee la propiedad de hipótesis más simple, un CCP T para el que M genera el programa P' detecta defectos en todo programa más simple que P'. Si para un CCP T' M genera un programa P'' más simple que P' se debe a que T' contiene menos información relevante para la especificación que T, por lo que M puede inferir un programa que corresponde a una especificación más simple.

El criterio IIE podría ser aplicado en distintos contextos y con diferentes objetivos. Podría ser aplicado en la enseñanza de técnicas de selección de casos de prueba como información de "cobertura". Podría ser aplicado también como técnica de selección de casos de prueba, ya sea utilizando una MII o tomando este criterio como referencia. Podría ser utilizado en la evaluación de CCP en experimentos o en trabajo de campo.

La técnica de selección de casos de prueba asociada al criterio de aceptación IIE, si bien es de caja negra, podría ser evaluada experimentalmente sin requerir de la participación de sujetos que la apliquen. Esto posibilitaría evitar gran parte de las dificultades de la experimentación con participación de sujetos.

A su vez, una MII como ADATE podría ser utilizada en la generación de programas defectuosos para experimentos y en la obtención de especificaciones de requerimientos ejecutables a partir de un CCP o ejemplos de funcionamiento esperado de un programa.

9.3 Limitaciones

La principal limitación del criterio IIE deriva de la dependencia de una MII y de sus capacidades:

a) El valor de E, su determinación, impacto y relación con las características de cada especificación o programa deben ser evaluadas.

b) No está claro si ADATE posee la propiedad conservativa requerida por Zhu para una MII [ZHU96]. Por otra parte tampoco está claro el impacto de que no tenga esa propiedad.

c) El criterio IIE presenta un conjunto de dificultades para su aplicación en un ambiente real. La principal está relacionada con el tiempo de ejecución requerido por una MII como ADATE para generar un programa. Algunas de las generaciones realizadas para el programa de ordenación que es muy simple ejecutaron por más de 24 horas.

d) No está claro si este enfoque puede escalar a programas más complejos o si solo resulta factible con programas muy simples. ADATE dispone de un mecanismo mediante el cual es posible suministrar componentes a ser usados en la generación, por lo que una generación gradual por componentes es concebible, pero no se ha evaluado.

e) El sistema ADATE presenta hoy en día serias limitaciones en lo atinente a su interfaz con el usuario.

f) Casos de prueba con tipos inválidos no fueron incluidos en la aplicación del criterio IIE.

9.4 Trabajo futuro

Los resultados obtenidos y las limitaciones marcan líneas de trabajo futuro:

a) Evaluar la aplicación de IIE en programas más complejos.

b) Evaluar la aplicación de IIE en la enseñanza de técnicas de selección de casos de prueba.

c) Evaluar la aplicación de IIE como criterio de selección de casos de prueba.

d) Evaluar la efectividad de IIE como criterio de selección de casos de prueba en experimentos sin participación de sujetos que apliquen la técnica.

e) Aplicar IIE para evaluar diversos CCP, ya sea en experimentos, cursos o en trabajo de campo.

f) Aplicar IIE considerando entradas con tipos válidos y no válidos.

g) Explorar la noción de “clase de equivalencia con igual información relevante para la especificación”.

h) Explorar la generación de programas defectuosos mediante una MII y su eventual utilidad.

i) Evaluar el uso de una MII para generar una especificación ejecutable de un programa a partir de CCP.

Bibliografía

[ANG83] D.Angluin, C.H.Smith, Inductive Inference: Theory and Methods, ACM Comput. Surveys, 15(3), pp. 237-269, 1983

[ARM05] P.Armour, The Unconscious Art of Software Testing. CACM, vol 48, iss.1, January 2005, pp. 15-18

[BAS86] V.Basili, R.Selby, D.Hutchens, Experimentation in Software Engineering, IEEE Transactions on Software Engineering, Vol SE-12,no 7, July 1986, IEEE, 1986

[BAS87] V.Basili, R.Selby, Comparing the Effectiveness of Software Testing Strategies, IEEE Transactions on Software Engineering, Vol.13, issue 12, pp. 1278-1296, Dec. 1987

[BAS96] V.Basili, The Role of Experimentation in Software Engineering: Past, Current, and Future, 18th International Conference on Software Engineering ICSE'96, pp. 442-449, IEEE CS, 1996

[BAS02] F. Shull, V. Basili, J. Carver, J.C. Maldonado, G.H. Travassos, S. Fabbri, ,Replicating software engineering experiments: Addressing the tacit knowledge problem, ISESE '02 Proceedings of the 2002 International Symposium on Empirical Software Engineering, pp. 7-16

[BEE08] A.Beer, R.Ramler, The Role of Experience in Software Testing Practice, Software Engineering and Advanced Applications, 2008, SEAA'08, 34th Euromicro Conference, pp. 258-265

[BEI90] B.Beizer, Software Testing Techniques (2nd. ed.), Van Nostrand Reinhold Co., New York, 1990

[BER03] A.Bertolino, Software Testing Research and Practice, ASM'03 Proceedings of the abstract state machines 10th international conference on Advances in theory and practice, pp. 1-21, Springer-Verlag Berlin, Heidelberg, 2003

[BER04] P.Berander, Using Students as Subjects in Requirements Prioritization, Proceedings of the 2004 International Symposium on Empirical Software Engineering, IEEE CS, 2004

[BER07] A. Bertolino, Software Testing Research: Achievements, Challenges, Dreams, Proceedings FOSE'07 Future of Software Engineering, pp 85-103, IEEE CS, Washington DC, 2007

[BERN07] S.Berner, R.Weber, R.Keller, Enhancing Software Testing by Judicious Use of Code Coverage Information, Proceedings of the 29th International Conference on Software Engineering ICSE'07, pp. 612-620, IEEE Computer Society, Washington DC, 2007

[BER96] F.Berdagamo, D.Gunetti, Testing by Means of Inductive Program Learning, ACM Transactions on Software Engineering and Methodology – TOSEM, Vol. 5 Issue 2, April 1996, pp. 119-145, ACM, New York, 1996

[BRI04] L.C.Briand, M.Di Penta, Y.Labiche, Assessing and Improving State-Based Class Testing: A Series of Experiments, IEEE Transactions on Software Engineering, Vol 30, NO.11, November 2004

[BRI07] L.C Briand, A Critical Analysis of Empirical Research in Software Testing, ESEM'07 Proceedings of the First International Symposium on Empirical Software Engineering and Measurement, pp.1-8, IEEE Computer Society, Washington, 2007

[BRI07A] L.C. Briand, Y. Labiche, Z. Bawar, Using Machine Learning to Refine Black-Box Test Specifications and Test Suites, Technical Report TR SCE-07-05, Carleton University

[CAL10] G.Calikli, A.Bener, B.Arslan, An Analysis of the Effects of Company Culture, Education and Experience on Confirmation Bias Levels of Software Developers and Testers, ICSE'10, Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering – Volume 2,pp. 187-190,ACM New York, 2010

[CAL12] G.Calikli, A.Bener, Influence of confirmation biases of developers on software quality: an empirical study, Software Quality Journal, July 2012, Springer US, 2012

[CAU12] A.Causevic, D.Sundmark, S. Punnekkat, Test Case Quality in Test Driven Development:
A Study Design and a Pilot Experiment, International Conference on Evaluation & Assessment in Software Engineering (EASE 2012), España, 2012

[CHE11] Z.Chen, J.Zhang, B.Luo, Teaching software testing methods based on diversity principles, Proceedings of the 24th Conference on Software Engineering Education & Training CSEET '11, pp. 391-395, IEEE CS

[CLA89] L.Clarke et al., A Formal Evaluation of Data Flow Path Selection Criteria, IEEE Transactions on Software Engineering, Vol 15, No. 11, November 1989.

[CZE13] H. Czemerinski, H., Braberman, V., Uchitel, S.: Behaviour Abstraction Coverage as Black-Box Adequacy Criteria, Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on, pp. 222-231, 2013

[DIJ70] E.W.Dijkstra, Notes on Structured Programming, T.H.-Report 70-WSK-03, disponible en http://www.cs.utexas.edu/~EWD/ewd02xx/EWD249.PDF

[FORG02] I.Forgacs, A.Bertolino, Preventing Untestedness in Data-Flow Based Testing, Software Testing, Verification and Reliability, Volume 12, issue 1, pp. 29-58, John Wiley & Sons, March 2002

[FRA93] G.Frankl, E.Weyuker, Provable Improvements on Branch Testing, IEEE Transactions on Software Engineering, Vol 19, no 10, October 1993

[FRA12] G. Fraser, N. Walkinshaw, Behaviourally Adequate Software Testing. Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on, pp. 300-309, 2012

[GHE03] C.Ghezzi, M.Jazayeri, D.Mandrioli, Fundamentals of Software Engineering, Second Edition, Pearson Prentice-Hall, 2003

[HAM89] R. Hamlet, Theoretical Comparison of Testing Methods, ACM SIGSOFT Software Engineering Notes, Volume 14 Issue 8, Dec. 1989

[HAM90] D.Hamlet, R.Taylor, Partition testing does not inspire confidence, IEEE Trans. Softw. Eng., 16(12):1402-1411, 1990

[HAR00] M.J.Harrold, Testing: a roadmap, ICSE '00 Proceedings of the Conference on The Future of Software Engineering, Pages 61 – 72, ACM, New York, 2000

[HOF07] A.Höfer, W. F. Tichy, Status of Empirical Research in Software Engineering, International Conference on Empirical Software Engineering Issues, Critical Assessment and Future Directions 2006, pp.10-19, Springer-Verlag Berlin, Heidelberg, 2007

[HOS05] M.Höst, C.Wohlin, T.Thelin, Experimental Context Classification: Incentives and Experience of Subjects, Proceedings of the 27th International Conference on Software Engineering ICSE'05, pp. 470-478, IEEE CS

[HOW80] W. E. Howden. Functional program testing. IEEE Transactions on Software Engineering, SE-6:162–169, March 1980.

[HUM89] W. S. Humphrey, Managing the Software Process, Addison-Wesley, Boston, USA, 1989

[ISO25] ISO/IEC 25000:2014 - Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- Guide to SQuaRE

[JED07] A.Jedlitschka, L.C.Briand, The Role of Controlled Experiments – Working Group Results, International Conference on Empirical Software Engineering Issues, Critical Assessment and Future Directions 2006, pp.58-62, Springer-Verlag Berlin, Heidelberg, 2007

[JON11] C.Jones, O.Bonsignor, The Economics of Software Quality, Pearson Education, Boston, 2011

[JUR01] N.Juristo, A.M.Moreno, Basics of Software Engineering Experimentation, Kluwer Academic Publishers, Boston, 2001

[JUR04] N.Juristo, A.M.Moreno, S.Vegas, Reviewing 25 Years of Testing Technique Experiments, Empirical Software Engineering, 9, 7-44, Kluwer 2004

[JUR04b] N.Juristo, A.M.Moreno, S.Vegas, Towards Building a Solid Body of Knowledge in Testing Techniques, ACM SIGSOFT, September 2004, Vol 29, 5

[JUR11] N.Juristo, S.Vegas, M.Solari, S.Abrahao, I.Ramos, Estudio de la Efectividad de Tres Técnicas de Evaluación de Código: Resultados de una Serie de Experimentos, Actas de las XVI Jornadas de Ingeniería del Software y Bases de Datos, Universidad de la Coruña, 2011

[JUR12] N.Juristo, S.Vegas, M.Solari, S.Abrahao, I.Ramos, Comparing the Effectiveness of Equivalence Partitioning, Branch Testing and Code Reading by Stepwise Abstraction Applied by Subjects, IEEE Fifth International Conference on Software Testing, Verification and Validation, pp. 330-339

[KAM95] E. Kamsties, C.M. Lott, 1995. An empirical evaluation of three defect-detection techniques. Proceedings of the Fifth European Software Engineering Conference. Sitges, Spain.

[KAM95a] E. Kamsties, C.M. Lott, 1995. An empirical evaluation of three defect-detection techniques, Technical Report ISERN 95-02, Dept. Computer Science, University of Kaiserslautern, May 1995

[KAN99] C.Kaner, j.Falk, H.Q.Nguyen, Testing Computer Software, second ed., John Wiley & Sons, 1999

[KAN02] C.Kaner, J.Bach. B.Petichord, Lessons Learned in Software Testing: A Context-Driven Approach, Wiley & Sons,2002

[KAN07] C.Kaner, S.Padmanabhan, Practice and Transfer of Learning in the Teaching of Software Testing, Proceedings of the 20th Conference on Software Engineering Education & Training CSEET '07, pp. 157-166, IEEE Computer Society, Washington DC, 2007

[KEN07] D.Kennedy, A. Hyland, N.Ryan, Writing and using learning outcomes: a practical guide, University College Cork, 2007

[KIT02] B.A.Kitchenham, S.L.Pfleeger, L.M.Pckard, P.W.Jones, D.C.Hoaglin, K.El Emam, J.Rosenberg, Preliminary Guidelines for Empirical Research in Software Engineering, IEEE Transactions on Software Engineering, Vol.28, NO. 8, August 2002

[KIT04] B.Kitchenham, T.Dyba, M.Jorgensen, Evidence-Based Software Engineering, Proceedings of the 26th International Conference on Software Engineering ICSE'04, pp. 273-281, IEEE Computer Society, Washington DC, 2004

[KITZ10] E.Kitzelmann, Inductive Programming: A Survey of Program Synthesis Techniques, Approaches and Applications of Inductive Programming, Lecture Notes in Computer Science Volume 5812, pp 50-73, Springer-Verlag, Berlin, 2010

[KOZ92] J.R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems), MIT Press, 1992

[LAW05] J.Lawrence, et al., How well do professional developers test with code coverage visualizations? An empirical study, IEEE Symposium on Visual Languages and Human-Centric Computing 2005, pp.53-60

[LEV94] L.M.Leventhal, B.E.Teasley, D.S.Rohlman, Analyses of factors related to positive test bias in software testing, International Journal of Human-Computer Studies, Volume 41, Issue 5, November 1994, pp. 717-749

[LI08] Li, Ming; Vitányi, Paul: An Introduction to Kolmogorov Complexity, Third Edition, ISBN: 978-0-387-33998-6, Springer 2008

[MLT13] MLton compiler, disponible en http://mlton.org/, accedido 5/5/17

[MYE78] G.J. Myers,. 1978. A controlled experiment in program testing and code walkthroughs/inspections. Communications of the ACM. 21(9): 760–768.

[MYE04] G.J.Myers, The Art of Software Testing, Second Edition, John Wiley & Sons, Inc., 2004

[PRE94] Roger S. Pressman. Software Engineering - A Practitioners Approach. McGraw-Hill, 1994

[PUC01] R.Pucella, Notes on Programming Standard ML of New Jersey (version 110.0.6), disponible en http://www.cs.cornell.edu/riccardo/prog-smlnj/notes-011001.pdf, accedido 5/5/17

[OLS98] J.R.Olsson, The Art of Writing Specifications for the ADATE Automatic Programming System, Genetic Programming 1998: Proceedings of the Third International Conference, pp 278-283, Morgan Kaufmann, october 1999

[OLS13] J.R.Olsson, ADATE – Automatic Design of Algorithms Through Evolution, disponible en http://www-ia.hiof.no/~rolando/ , accedido 5/5/17

[RIC89] D. Richardson, O. O'Malley, C. Tittle, Approaches to Specification-Based Testing, ACM SIGSOFT Software Engineering Notes, Volume 14, Issue 8, Dec. 1989, pp. 86-96

[ROP93] Marc Roper. Software Testing. McGraw-Hill, 1993.

[ROT00] K.Rothermel et al., WYSIWYT testing in the spreadsheet paradigm: an empirical evaluation, Proceedings of the 22nd International Conference on Software Engineering ICSE'00, pp. 230-239, ACM, New York, 2000

[RUN06] P.Runeson et al., What Do We Know about Defect Detection Methods?, IEEE Software, Volume 23 Issue 3, May 2006, pp. 82-90, IEEE CS, Los Alamitos, CA, 2006

[SJO05] D.Sjoberg et al., A Survey of Controlled Experiments in Software Engineering, IEEE Transactions on Software Engineering, Vol 31, no 9, September 2005, pp. 733-753, IEEE CS 2005

[SJO07] D.Sjoberg, T.Dyba, M.Jorgensen, The Future of Empirical Methods in Software Engineering Research, Future of Software Engineering FOSE'07,pp. 358-378, IEEE CS, 2007

[SWE04] Guide to the Software Engineering Body of Knowledge, 2004 Edition – SWEBOK, disponible en http://www.computer.org/portal/web/swebok/home

[TIC97] W.F.Tichy, Should Computer Scientists Experiment More?, IEEE Computer, Volume 31, Issue 5, May 1998, pp 32-40, IEEE CS, 1998

[VAL84] L.Valiant, A Theory of the Learnable, Communications of the ACM 27(11), pp. 1134-1142, 1984

[VALL09] D, Vallespir, J. Herbert, Effectiveness and Cost of Verification Techniques: Preliminary Conclusions on Five Techniques. Computer Science (ENC), 2009 Mexican International Conference on, pp. 264 – 271, 2009

[VAT06] G.Vattekar, ADATE User Manual, Østfold University College, March 2006, disponible en http://www-ia.hiof.no/~rolando/ , accedido 5/5/17

[WAL10] N.Walkinshaw, The Practical Assessment of Test Sets with Inductive Inference Techniques, TAIC PART'10 – 5th International Academic and Industrial Conference on Testing – Practical and Research Techniques, pp 165-172, Springer-Verlag, Berlin, 2010

[WEY83] E.J.Weyuker, Assessing test data adequacy through program inference, ACM Transactions on Programming Languages and Systems, 5(4), 641-655

[WHI00] J.Whittaker, What Is Software Testing? And Why Is It So Hard?, IEEE Software, Volume 17, Issue 1, Jan.2000, pp 70-79, IEEE CS, 2000

[WOH03] Wohlin, C., Höst, M., & Henningsson, K. (2003). Empirical research methods in software engineering. Empirical Methods and Studies in Software Engineering, 7-23.

[WOO97] M. Wood, M. Roper, A. Brooks, J. Miller, 1997. Comparing and combining software defect detection techniques: A replicated empirical study. Proceedings of the 6th European Software Engineering Conference. Zurich, Switzerland.

[WQR15] World Quality Report 2015-16, Seventh Edition, Capgemini, HP, Sogeti, disponible en https://www.capgemini.com/thought-leadership/world-quality-report-2015-16, último acceso 25/2/16

[ZHU92] H.Zhu, P.Hall, J.May, Inductive Inference and Software Testing, Software Testing, Verification and Reliability, Volume 2, Issue 2, pp 69-81, July 1992, Wiley 1992

[ZHU96] H.Zhu, A Formal Analysis of the Subsume Relation Between Software Test Adequacy Criteria, IEEE Transactions on Software Engineering, Vol 22, no 4, April 1996

[ZHU96A] H.Zhu, A Formal Interpretation of Software Testing as Inductive Inference, Software Testing, Verification and Reliability, Volume 6, Issue 1, pp 3-31, March 1996, Wiley 1996

[ZHU97] H. Zhu, P. Hall, J. May, Software Unit Test Coverage and Adequacy, ACM Computing Surveys, Vol. 29, No. 4, December 1997

Anexo A: Especificación de SORT

Especificación de SORT

```
datatype list = nill | cons of int * list
fun f( Xs : list ) : list = raise D1
fun main( Xs : list ) : list = f Xs

%%
~
fun to_list [] = nill
  | to_list( X :: Xs ) = cons( X, to_list Xs )

fun from_list nill = []
  | from_list( cons( X, Xs ) ) = X :: from_list Xs

val Input_output_pairs = [
  ( [], [] ),
  ( [7], [7] ),
  ( [5,6], [5,6] ),
  ( [1,0], [0,1] ),
  ( [0,1,2], [0,1,2] ),
  ( [0,2,1], [0,1,2] ),
  ( [1,0,2], [0,1,2] ),
  ( [1,2,0], [0,1,2] ),
  ( [2,0,1], [0,1,2] ),
  ( [2,1,0], [0,1,2] ),
  ( [1,0,2,3], [0,1,2,3] ),
  ( [2,1,0,4,3], [0,1,2,3,4] ),
  ( [2,5,1,0,4,3], [0,1,2,3,4,5] ),
  ( [5,2,1,4,3,0], [0,1,2,3,4,5] ),
  ( [9,2,5,1,0,7,8,3,4,6], [0,1,2,3,4,5,6,7,8,9] )
  ]

val Validation_inouts = [
  ( [2,1,3,0,5,4], [0,1,2,3,4,5] ),
  ( [7,2,1,3,8,0,5,4,9,6], [0,1,2,3,4,5,6,7,8,9] )
  ]

val Inputs = map( to_list o #1, Input_output_pairs )
val Validation_inputs = map( to_list o #1, Validation_inouts )

val Outputs = Vector.fromList( map( to_list o #2,
  Input_output_pairs @ Validation_inouts ) )

val Abstract_types = []

val Funs_to_use = [ "false", "true", "<", "nill", "cons" ]

val Reject_funs = []
fun restore_transform D = D


structure Grade : GRADE =
struct

type grade = unit
val zero = ()
val op+ = fn(_,_) => ()
val comparisons = [ fn _ => EQUAL ]
val toString = fn _ => ""
val fromString = fn _ => SOME()

val pack = fn _ => ""
val unpack = fn _ =>()
```

```
val post_process = fn _ => ()

val toRealOpt = NONE

end

fun output_eval_fun( I : int, _ : list, Y : list ) =
  if Vector.sub( Outputs, I ) = Y then
    { numCorrect = 1, numWrong = 0, grade = () }
  else
    { numCorrect = 0, numWrong = 1, grade = () }

val Max_output_genus_card = 2

val Max_time_limit = 262144
val Time_limit_base = 2.0
```

Anexo B: PG11 anotado con literales de PG7

```
PG11 anotado con literales de PG7
fun f Xs =
  case Xs of
    nill => Xs
  | cons( V42AB [V1927], V42AC [V1928]) =>
  case f( V42AC [V1928] ) of
    nill => Xs
  | cons( V42AE [V3350], V42AF [V3351]) =>
  case ( V42AB [V1927]< V42AE [V3350]) of
    false => cons( V42AE [V3350], f( cons( V42AB [V1927], V42AF [V3351]) ) )
  | true => cons( V42AB [V1927], f( V42AC [V1928] ) )
```

Anexo C: PG7 y PG8 correctos

I. Demostración de que PG7 ordena correctamente listas de largo n

PG7

```
fun f Xs =
case Xs of
nill => Xs
| cons( V1927, V1928 ) =>
case f( V1928 ) of
nill => Xs
| cons( V3350, V3351 ) =>
case ( V1927 < V3350 ) of
false => cons( V3350, f( cons( V1927, V3351 ) ) )
| true => cons( V1927, f( V1928 ) )
```

1. Largo de lista 0

```
case Xs of
nill => Xs
```

para lista vacía devuelve lista vacía => cumple

2. Largo de lista n cumple => largo de lista n+1 cumple

Xs de largo n+1 > 0

```
|cons( V1927, V1928 )
case f( V1928 ) of
nill => Xs
```

para lista de un elemento devuelve la lista => cumple

```
| cons( V3350, V3351 ) =>
```

V1928 es de largo n por lo que f(V1928) está ordenada => V3350 <= todo elemento de V1928

```
case ( V1927 < V3350 ) of
false => cons( V3350, f( cons( V1927, V3351 ) ) )
```

V3350 <= V1927 y V3350 <= todo elemento de V3351 y
cons(V1927, V3351) es de largo n por lo que f(cons(V1927, V3351)) está ordenada
=> cumple

```
case ( V1927 < V3350 ) of
| true => cons( V1927, f( V1928 ) )
```

V1927 < V3350 y V3350 <= todo elemento de V1928 y V1928 es de largo n por lo que f(V1928) está ordenada
=> cumple

Esto cubre todos los casos, entonces f ordena listas de largo n+1

3. Por inducción completa f ordena listas de largo n siendo n natural

II. Demostración de que PG8 ordena correctamente listas de largo n

PG8

```
fun f Xs =
case Xs of
nill => Xs
| cons( V1A03, V1A04 ) =>
case f( V1A04 ) of
nill => Xs
| cons( VCF06, VCF07 ) =>
case ( VCF06 < V1A03 ) of
false => cons( V1A03, f( V1A04 ) )
| true => f( cons( VCF06, f( cons( V1A03, f( VCF07 ) ) ) ) )
```

1. Largo de lista 0

```
case Xs of
nill => Xs
```

para lista vacía devuelve lista vacía => cumple

2. Largo de lista n cumple => largo de lista n+1 cumple

Xs de largo n+1 > 0

```
| cons( V1A03, V1A04 ) =>
case f( V1A04 ) of
nill => Xs
```

para lista de un elemento devuelve la lista => cumple

```
| cons( VCF06, VCF07 ) =>
```

V1A04 es de largo n por lo que f(V1A04) está ordenada => VCF06 <= todo elemento de V1A04

```
case ( VCF06 < V1A03 ) of
false => cons( V1A03, f( V1A04 ) )
```

V1A03 <= VCF06 y V1A03 <= todo elemento de V1A04 y
V1A04 es de largo n por lo que f(V1A04) está ordenada
=> cumple

```
case ( VCF06 < V1A03 ) of
| true => f( cons( VCF06, f( cons( V1A03, f( VCF07 ) ) ) ) )
```

VCF06 < V1A03 y VCF06 <= todo elemento de V1A04 y
cons(V1A03,f(VCF07)) es de largo n por lo que f(cons(V1A03,f(VCF07))) está ordenada
=> cumple

Esto cubre todos los casos, entonces f ordena listas de largo n+1

3. Por inducción completa f ordena listas de largo n siendo n natural