Tesis de licenciatura

Testing de calidad en datos

Departamento de Computación

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Tesista

Ernesto Kiszkurno

Director

Daniel Yankelevich

Agosto del 2000

Resumen

En el presente trabajo se define un modelo de testing para el análisis de la *calidad de datos*. Este modelo se logra a partir de una dualización del modelo de testing de software vigente en nuestros días. El modelo propuesto es acompañado por nuevas definiciones para viejos conceptos relacionados con testing.

Utilizando el modelo propuesto, y a modo de validación, se presentan dos técnicas de derivación de casos de prueba sobre los datos. Para la primera, y debido a su origen estructural, se elabora también un modelo de cubrimiento para el objeto de test. Esto se acompaña con la definición de criterios de cubrimiento sobre el modelo y un análisis de las implicaciones existentes entre ellos. La segunda técnica, de origen funcional, se explica directamente en base a ejemplos.

La correctitud, la utilidad y la aplicabilidad de las técnicas y el modelo definidos, se evalúan utilizando un experimento y un caso de estudio.

Indice

Capítulo 1

Introducción

1. Testing de programas

El objetivo del testing de programas es disminuir el riesgo de ocurrencia de errores en él. Si bien no es la única actividad destinada al aseguramiento de calidad dentro del desarrollo de sistemas, es una de las más importantes. La idea es establecer un nivel de calidad aceptable en el software y verificarlo de alguna forma.

Desde sus inicios, fueron apareciendo distintas técnicas y metodologías destinadas a atacar este problema. En un principio dichas técnicas constituían esfuerzos aislados. Podría decirse que constituían soluciones particulares al problema. Abarcaban un dominio específico y las posibilidades de re uso eran bajas.

No obstante esto, a partir del trabajo de investigación y el transcurso del tiempo, la actividad ha evolucionado. Hoy en día disponemos de un modelo de testing de programas establecido. El mismo permitió, entre otras cosas, consolidar las técnicas antes mencionadas, encontrar nuevas variantes y re usar la experiencia adquirida a lo largo del tiempo. Las nociones de caso de test y dato de test, de resultado esperado u objeto de test son pilares de este modelo ya que permitieron manejar un lenguaje común y uniforme dentro de la actividad. Además, conceptos tales como, el tipo de testing (funcional o estructural), el nivel de test (unidad, componente, sistema o integración) o el grado de cubrimiento de un test, se han visto beneficiados por este modelo y han evolucionado con él.

En pocas palabras, la maduración y establecimiento de un modelo de testing de programas hizo posible sistematizar y ordenar el estudio de la calidad del software.

2. Calidad de datos y Testing de datos

El problema de la baja calidad en la información es un tópico que ha adquirido una gran importancia en los últimos años. Si bien el tema ha sido abordado desde diferentes perspectivas, el área de los llamados Sistemas de Información ha sido la más dedicada, pudiéndose encontrar diversos trabajos al respecto. Generalmente estos están basados en la idea de analizar la calidad a partir de la noción de dimensión o atributo. Además en ellos el énfasis esta puesto en el proceso de generación de la información.

En concordancia con estos esfuerzos, y ya desde el punto de vista de la Ingeniería de Software, creemos que es posible analizar la calidad de la información a partir de la instanciación de técnicas originarias del desarrollo de sistemas en los datos. En particular, es posible analizar el nivel de calidad de los datos almacenados en sistemas de software a partir de practicar actividades de testing sobre los datos.

Creemos que el rol secundario otorgado a los datos durante las etapas de desarrollo de los sistemas contribuye a aumentar los problemas de calidad que luego existirán en ellos. Esta realidad se contrapone con un hecho innegable actualmente: Los datos conforman la pieza fundamental de interacción entre los distintos sistemas de software en una organización y constituyen un patrimonio importante dentro de la misma [BOBR98].

En este contexto es posible definir el objetivo del Testing de Datos. El mismo es establecer técnicas, metodologías y herramientas que permitan determinar el nivel de calidad de los datos almacenados en los sistemas informáticos en el ámbito de una organización en particular. Esto debe hacerse además, a través de todo el ciclo de vida de dichos sistemas.

Un buen punto de partida para esta tarea es re usar la experiencia del testing de programas puesto que la nueva actividad tiene una gran similitud con él. Esto puede verse en diversos aspectos, como por ejemplo, la terminología empleada, los conceptos básicos o el modelo. No obstante, será necesario establecer algunas nociones paralelas que tengan en cuenta el nuevo objeto de estudio: los datos.

3. Objetivo de este trabajo

La definición de un modelo de trabajo definido y común, dentro de una determinada actividad, permite encarar la tarea de investigación en forma más precisa y productiva. Posibilita a los investigadores intercambiar conocimiento, re usar las experiencias anteriores y continuar el trabajo comenzado por otros corroborándolo o refutándolo en beneficio del objetivo común.

Teniendo esto en mente y siendo tan largo el camino recorrido en el Testing de programas (la cantidad de conocimiento acumulado en dicha actividad es muy importante), nuestro objetivo en el presente trabajo es definir un modelo de testing para los datos. Este modelo deberá servir como framework para el establecimiento de técnicas de análisis de calidad sobre ellos. Adicionalmente al modelo, buscaremos establecer una noción básica de conceptos tales como nivel de testing o cubrimiento.

Creemos que es posible aprovechar el estado del arte del testing de programas en beneficio de la nueva actividad. A partir de los elementos más comunes del modelo de testing de programas, pretendemos establecer un modelo dual para el testing de datos. Este modelo deberá mantener un vocabulario común que permita aprovechar lo más posible las técnicas definidas para software, en los datos.

Además se validará el modelo propuesto en base a dos experiencias prácticas (un experimento y un caso de estudio). Esto permitirá evaluar la correctitud y la utilidad del mismo.

4. El presente trabajo

En el capítulo 2 presentamos el modelo de testing de datos propuesto. El mismo se formula a partir del modelo de testing de programas que sirve como base para el paralelo. Por este motivo, se presenta una breve reseña del modelo de software y luego se realiza la descripción del nuevo modelo.

En el capítulo 3 se determinan los lenguajes de escritura adecuados para los dos elementos fundamentales del modelo definido (los casos de prueba y sus queries asociadas) y la nueva noción de instanciación existente entre ellos. Para esto, en primer lugar se presenta una breve descripción de los niveles de expresividad posibles a la hora de enunciar propiedades sobre datos. Luego, se enumeran las características relacionadas con expresividad deseables en cada uno de los lenguajes seleccionados. A partir de esto se presenta la elección realizada y la noción de instanciación que la acompaña.

En el capítulo 4 se describen las nociones de nivel y tipo de testing las cuales resultan fundamentales para situar las técnicas enunciadas en los capítulos siguientes. Primero se presentan los distintos niveles de testing de datos practicables sobre un conjunto de datos utilizando gran parte del vocabulario ya existente en la actividad de testing de programas. Luego se separan las nociones de testing estructural y funcional. A partir de esta distinción, ya existente en la actividad para el software, es posible definir dos técnicas de derivación de casos de prueba sobre los datos.

Los capítulos 5 y 6 presentan un modelo de cubrimiento en primer lugar, y los criterios asociados al mismo en segundo. Dicha presentación se elabora a partir del modelo de datos que se pretende probar, puesto que desde él se construye un grafo (llamado de datos) sobre el cual es posible luego definir distintos criterios de cubrimiento. Algunos de ellos se basan solamente en la estructura del grafo mientras que otros requieren de ciertas anotaciones complementarias sobre los campos o atributos del modelo. Por otro lado, y para clarificar la técnica, se presenta un experimento que sirve de ejemplo durante la explicación de las distintas tareas asociadas al uso tanto del modelo de cubrimiento como de los criterios (el experimento se utiliza para la construcción del grafo, para la derivación de casos de prueba y de las queries). Se presentan además los resultados obtenidos luego de la ejecución de dichas queries sobre los datos para permitir evaluar la efectividad del conjunto de casos.

El capítulo 7 trata sobre criterios de testing funcional. Aquí se presenta una técnica que permite derivar casos de prueba a partir de criterios funcionales la cual utiliza fuertemente el concepto de dimensión de la calidad. Además de explicarse la forma de uso de dicha técnica, se discute en este capítulo su grado de aplicabilidad sobre las dimensiones de la calidad y ciertos tipos de datos.

El capítulo 8 presenta un segundo experimento sobre el cual se aplican las nociones de testing estructural definidas en los capítulos 5 y 6. El objetivo aquí es utilizar las nociones de cubrimiento definidas sobre un caso de estudio del cual no se tiene conocimiento funcional. Aquí se describen todos los pasos seguidos para ejecutar la prueba y los resultados obtenidos. Como último punto presentamos algunas conclusiones obtenidas a partir de este caso y de su comparación con el primer experimento de cubrimiento.

El capítulo 9 describe las conclusiones generales del trabajo y las distintas alternativas de trabajo futuro descubiertas durante su elaboración.

5. Contexto

Este trabajo se realiza en el marco del proyecto Artes: Arquitectura y Testing de Sistemas de Software.

El principal objetivo del proyecto es obtener fundamentos para algunas prácticas consensuadas en la profesión. También se espera derivar prácticas de conceptos (teóricos) clave de la ingeniería del software.

El proyecto se centra en las áreas de arquitectura y diseño (asociadas a la descripción de sistemas), y en las de validación y verificación de software. No se hace hincapié en prácticas relacionadas con aspectos más distantes de la ciencia de la computación (comerciales, humanos, etc.), aunque probablemente se puedan incluir puntos relevantes como ser el análisis de riesgos, la planificación y la captura de requerimientos (consideradas prácticas claves en todas las recomendaciones de ingeniería del software).

Se trabaja en la integración de las actividades de descripción y validación de sistemas. Esta integración se logra a partir de diferentes mecanismos y estrategias, respondiendo a preguntas tales como: ¿cómo usar la información obtenida durante una de estas actividades en la otra? ¿cuál es el marco común en el que estas actividades se desarrollan? ¿cómo se modifica la actividad de testing de acuerdo al tipo de arquitectura en el que se trabaja?

Sobre calidad se pretenden aplicar las técnicas tradicionales utilizadas en calidad de software para el trabajo sobre datos. La definición de métricas para la calidad de los datos o la replicación de modelos (como el de testing) serán temas de investigación. El objetivo perseguido es brindar un marco homogéneo de trabajo en dicha área.

Capítulo 2

Modelo para el testing de datos

1. Introducción

La actividad de testing consiste en observar el comportamiento de un objeto ante determinados estímulos. El objetivo pues es verificar que ambos comportamientos (el observado y el esperado) se corresponden.

Como en cualquier actividad científica esta observación se hace sobre un modelo definido que cumple ciertos supuestos. Estos supuestos son los que luego permitirán al observador sacar conclusiones sobre lo observado. El modelo de testing entonces constituye el marco de referencia en el cual se desarrolla la actividad testing.

En la actualidad existen diversas técnicas aplicables para el testing de programas. Ellas asumen un modelo propio con determinados supuestos de trabajo sobre los cuales la técnica es aplicable. Por ejemplo, las técnicas de cubrimiento para el testing estructural presuponen la existencia de un grafo de control para el programa. Si bien existen diversos modelos para el testing de programas, es posible encontrar cierto elementos comunes a todos ellos. Estos elementos definen el vocabulario básico para la actividad y resultan indispensables para el desarrollo de la actividad.

En este capítulo describiremos un modelo básico asociado a la actividad de testing, pero planteada ahora sobre los datos, y para ello utilizaremos como punto de partida el modelo existente para el software.

Si bien existe cierta dualidad entre ambos modelos, los términos conocidos para el software deben ser replanteados para el nuevo objeto a testear: los datos.

2. Reseña del modelo de software

La cantidad de conceptos, términos y definiciones manejadas generalmente en el testing de programas puede llegar a ser grande. Sin embargo, la definición de un modelo básico puede hacerse mediante pocos elementos.

A continuación se mostraremos en forma esquemática dicho modelo.

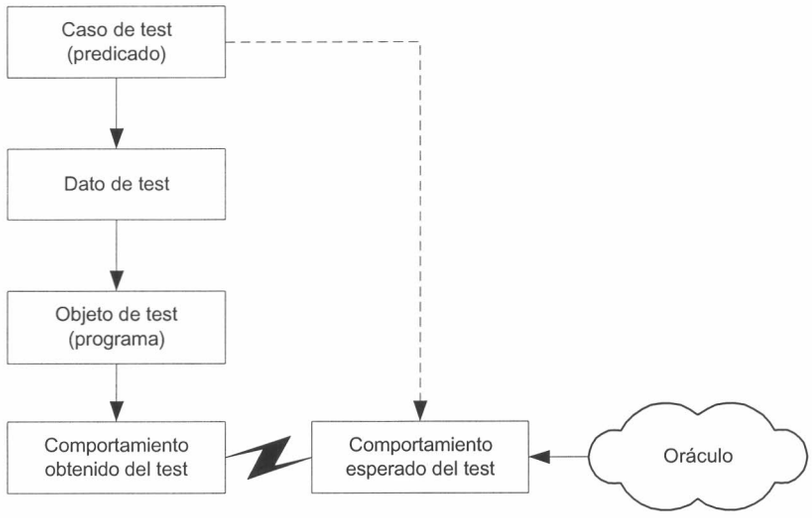


Ilustración 1. Modelo de testing de programas

En este modelo pueden verse los siguientes elementos:

- **Objeto de test.** Es el programa, función o módulo a testear, dependiendo del tipo de testing que sé este realizando.
- **Caso de test.** Generalmente es un predicado expresado en términos del dominio del objeto a testear. En el caso de una función por ejemplo, el predicado será escrito en base las entradas que la alimentan.
- **Dato/s de test.** Los *datos de test*, para un caso dado, están constituidos por un conjunto de datos que "*cumplen*" con el caso en cuestión. La relación entre caso y dato es de *instanciación*. Si c es el caso de test y d es su conjunto de datos, se dice que existe una función f_c tal que $f_c(d)$ = verdadero.
- **Comportamiento esperado.** Es una predicción sobre la salida que deberá tener el objeto testeado. Esta salida es definida sobre la base de un cierto *oráculo*.
- **Comportamiento obtenido.** Es la salida obtenida a partir de la ejecución del test. Esto es, la ejecución del programa en base a los datos de test.

Además de repasar las definiciones de cada uno de los elementos, es importante también, describir la forma en que estos interactúan durante el testing. La mecánica usual en este modelo es:

1. Se definen los casos de prueba y el comportamiento esperado para cada uno de ellos.
2. Se encuentran los datos asociados a cada caso de test (las instancias para cada caso).
3. Se ejecuta el objeto de test utilizando los datos definidos.

4. Se compara el comportamiento obtenido en el objeto con el comportamiento esperado para el mismo mediante alguna noción de falla.
5. Si los comportamientos no coinciden (esto se deduce mediante la función de éxito), nos encontramos frente a una falla en el objeto a testear. Si coinciden, el caso se considera exitoso.

Hasta aquí una breve descripción del modelo para el testing de programas[1], ahora pasaremos a la definición del modelo de testing para los datos.

3. Modelo Dual

El modelo para el testing de datos puede construirse a partir del de software. Esto se debe a que la actividad de testing en sí misma no cambia demasiado aún siendo otro el objeto observado. En este sentido, la noción de caso de test ha permanecido intacta. Para ver esto, basta pensar al caso de test como una hipótesis a observar sobre el objeto de pruebas, es decir una propiedad sobre los datos. Lo mismo sucede tanto para el concepto de resultado esperado como para el de resultado obtenido (ellos constituyen la comprobación empírica de la hipótesis).

La diferencia fundamental entre ambos es el nexo que une al caso de test y al objeto de test. Y esto es porque al cambiar el objeto a observar cambia, la herramienta de observación debe cambiar también.

A continuación presentamos un esquema dual que muestra lo dicho.

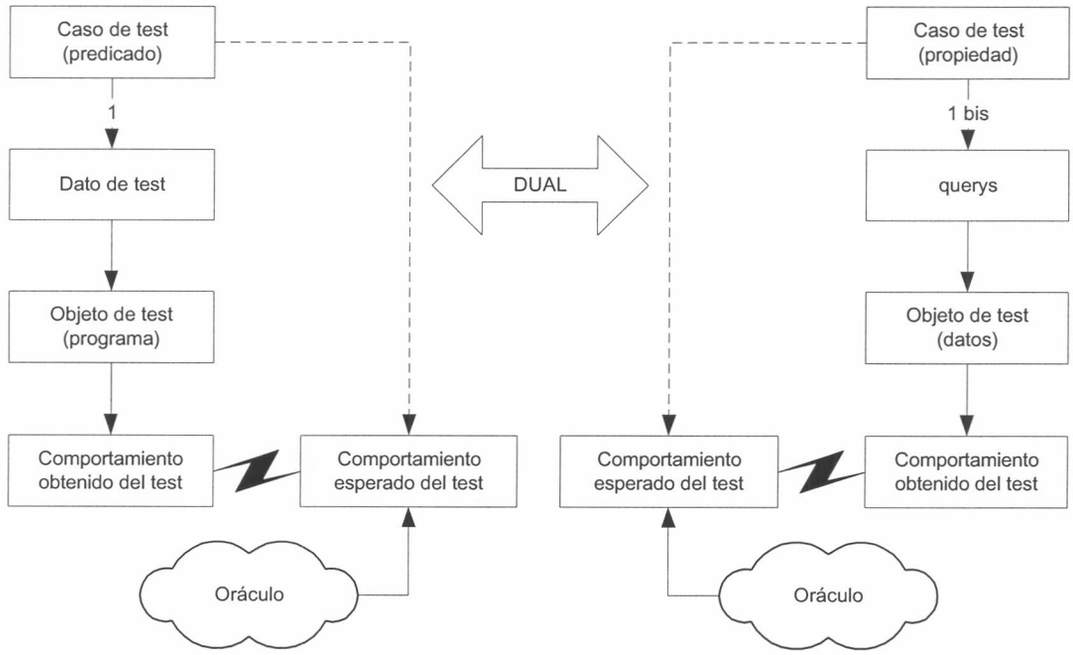


Ilustración 2. Modelo Dual

[1] Puede verse más detalle en [BEIZ90].

Mientras en el modelo para software el nexo entre los casos de test y el objeto está constituido por los datos de test, en el modelo dual este nexo es cubierto por un nuevo elemento: las queries.

Ahora, daremos una definición más precisa de cada elemento.

- **Objeto de test**. Es el conjunto de datos sobre el cual se practica el testing.
- **Caso de test**. Aquí el caso de test es una propiedad deseada en el objeto de test[2]. Dicha propiedad está expresada en términos de los datos que constituyen el objeto de la prueba.
- **Queries del test**. Las queries del test constituyen un conjunto de consultas, escritas en un lenguaje adecuado, que permite verificar si la propiedad a la cual hacen referencia se cumple o no en los datos que están siendo testeados.
- **Comportamiento esperado**. Es una predicción sobre la salida que deberá tener el objeto testeado. Esta salida es definida sobre la base de un *oráculo*[3].
- **Comportamiento obtenido**. Es la salida obtenida a partir de la ejecución del test. Esto es, la ejecución de las queries sobre los datos del test.

Como dijimos, la actividad de testing sigue siendo la misma aún habiendo cambiado el objeto de test. Por este motivo, la mecánica de trabajo en el nuevo esquema es similar a la enunciada para el modelo anterior. La resumimos a continuación:

1. Definir los casos de prueba (propiedades) y el comportamiento esperado para ellos.
2. Encontrar las queries necesarias para probar las propiedades definidas (para probar los casos).
3. Ejecutar los casos de prueba mediante la ejecución de las queries definidas sobre el conjunto de datos de prueba (el objeto del test).
4. Comparar el comportamiento obtenido en el objeto con el comportamiento esperado.
5. Si los comportamientos no coinciden, nos encontramos frente a una falla.
6. Si coinciden, el caso se considera exitoso.

---

[2] En el libro [BEIZ95], el autor define al caso de test como "un comportamiento deseable en el objeto del test", por esta razón es lógico establecer la definición dual de esta manera.

[3] El concepto de oráculo es usado aquí con la misma acepción que en testing de programas.

4. Discusión

En esta sección profundizaremos sobre algunos aspectos asociados al modelo definido. Los objetivos perseguidos con este análisis son dos: mejorar algunas definiciones en el modelo y resaltar las implicancias surgidas a partir del mismo.

Caso vs. Test

En [BEIZ95], el autor define una serie de nociones que brindan una visión algo distinta del modelo de testing de programas. Estas nociones ayudan a delimitar mejor el alcance de los conceptos de nuestro modelo separando la idea de *definición* de la *ejecución* de un test.

Haciendo el ejercicio de establecer definiciones paralelas en el testing de datos de los conceptos señalados por el autor, puede verse que los elegidos para el paralelo respetan la misma estructura.

	Testing de programas (TT)	**Testing de datos (TD)**
Sub test	La parte más pequeña del testing compuesta de: objeto (programa), estado inicial, entrada (datos de test) y resultado esperado.	Ídem TT. Está compuesta de: objeto (conjunto de datos), entrada (query) y resultado esperado.
Test	Secuencia de sub tests concatenados de alguna forma.	Ídem TT.
Feature	Comportamiento deseable en un objeto (programa).	Cualidad deseable en un objeto (conjunto de datos).
Caso	Agregación de features.	Idem TT.

Tabla 1. Definiciones de test y caso de test

Como se ve, las nociones de *test* y *sub test*, que agrupan los conceptos relacionados con la ejecución, quedan separadas de las de *caso* y *feature*, que están relacionadas con el aspecto de definición del test. Sin embargo, las definiciones presentadas en nuestro modelo encajan sin problemas.

El caso de test

El concepto central en ambos modelos, tanto en el MT como en el MTD, es el caso de test ya que a partir de él se estructura toda la teoría del testing. Ahora daremos aquí una definición más precisa.

> ***Definición 1***. *Un caso de test es una agregación features. A su vez, cada feature constituye una cualidad o una regla deseable en los datos que será escrita por medio de una propiedad. Esta propiedad será equivalente al predicado lógico utilizado en el MT.*

Esta definición se corresponde con lo dicho en la sección 3 de este capítulo. Además, a partir de ella, vemos que la mínima unidad en el modelo es el *feature*. No obstante esto, en lo que sigue del documento los términos *caso* y *feature* se utilizarán indistintamente por cuestiones de comodidad.

Lenguaje de escritura para los casos de test

En el MT, una feature es un predicado lógico[4], el cual generalmente es escrito en lógica de primer orden. El mismo se traduce como una restricción del dominio de entradas posibles al objeto que se pretende probar (programa).

Por el otro lado, en el MTD, para escribir una feature utilizamos el concepto de propiedad. Si bien en una primer instancia, una propiedad también puede ser descripta como un predicado lógico, consideramos que es necesario profundizar el análisis sobre este punto puesto que la elección de un lenguaje más adecuado puede facilitar la tarea de derivación de casos. Por ejemplo, un lenguaje que disponga de operadores especiales para el trabajo con datos puede otorgar mayor simplicidad y expresividad en la escritura de los casos (aún cuando su costo implementación sea más alto).

Dominio de origen

Si bien podemos encontramos con diferencias en el lenguaje con el cual se escriben los casos en cada modelo, el objetivo final de ambos es constituirse como una restricción en la entrada al objeto de test. Esta entrada, que como hemos dicho en el MT es el dominio del programa, está constituida por el conocimiento existente sobre los datos en el MTD.

Esta diferencia produce un fenómeno interesante entre ambos modelos: la inversión entre las entidades de datos y las entidades ejecutables. Hablando ligeramente, en el MT, los datos sirven de entrada para el programa (la entidad de datos es entrada de la entidad ejecutable); mientras que en el MTD, las queries (programas) son la entrada para los datos (la entidad ejecutable es la entrada para la entidad de datos).

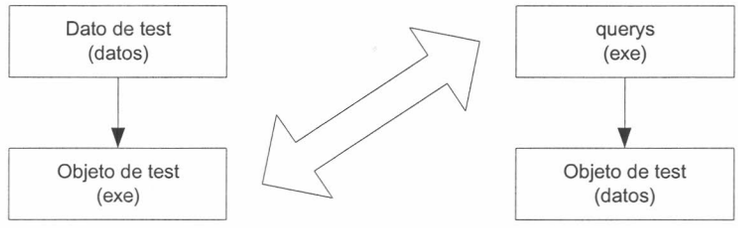


Ilustración 3. Intercambio de roles entre ambos modelos

Ahora, este hecho, es el principal causante de las diferencias existentes en la noción de instancia de en uno y otro modelo. Afortunadamente, esta inversión se encuentra encapsulada en el concepto de test y, si bien cambia la forma en que se ejecutará el mismo, para los fines prácticos del modelo, el paralelo se conserva.

---

[4] Una sentencia o expresión que toma un valor lógico de verdadero o falso, [BEIZ95].

Instanciabilidad

En cada uno de los modelos se utiliza al caso para poder realizar acciones concretas sobre el objeto de test a fines de evaluar su calidad. En el caso del MT, se generan datos de test a partir de él, que servirán luego para ejecutar el programa.

En el MTD, se generan queries las cuales serán evaluadas en el correspondiente objeto de test (el conjunto de datos a probar). Es decir, que en ambas situaciones debe ser posible "instanciar" los casos. Por esto, en el próximo capítulo procuraremos establecer con mayor exactitud la noción de instancia para el testing de datos.

Sin embargo, es importante mencionar en esta discusión, que la propiedad de *intercambiabilidad* de instancias para un caso de test dado permanece intacta. Este principio de intercambiabilidad posibilita utilizar distintos representantes para probar el mismo caso de test[5].

Inclusive, al igual que para el caso de software, es posible utilizar más de un representante por caso.

Comportamiento esperado

El comportamiento esperado se deriva a partir del caso de test definido. Para ello se utiliza el concepto de oráculo, el cual permite establecer cómo debe comportarse el objeto de test. Luego, al confeccionarse los test a ejecutar, dicho comportamiento esperado deberá ser re escrito en términos de las instancias definidas para el caso.

La idea enunciada en el párrafo anterior permanece intacta a la hora de cambiar el modelo de testing. Sin embargo, el comportamiento esperado en sí mismo y el oráculo utilizado para su derivación, pueden ser radicalmente distintos en un modelo y en otro (esto se menciona en el ejemplo citado en el punto "*La función de instanciación en el MTD*".

El Test

Si la idea de *caso* se asocia a la parte estática (o de definición) de la actividad de testing, el concepto de *test* debe ser asociado a la dinámica (o de ejecución).

El concepto de test se deriva a partir de la idea de caso. Mientras el caso constituye la hipótesis a testear, el test agrupa los elementos necesarios para llevar a cabo la prueba. Esto puede verse al repasar qué elementos componen un test. En el caso del MT el test está compuesto por[6],

- El programa u objeto de test.
- El estado inicial del objeto de test es decir, el contexto necesario para reproducir el caso.
- Los datos de entrada necesarios para reproducir el caso (las instancias del caso de test).
- El resultado esperado, pero escrito en términos del objeto de test

---

[5] La existencia de clases de equivalencia entre las instancias de casos de test es explotada en el testing funcional de programas para ahorrar esfuerzo y no ejecutar test que pertencen a una misma clase.
[6] Estas definiciones han sido extraídas de [BEIZ95].

Este conjunto de elementos permite realizar la ejecución del caso de test y el posterior análisis del resultado obtenido en esa ejecución. Paralelamente en el MTD, los componentes son,

- El conjunto de datos u objeto de test.
- El contexto necesario para reproducir el caso de test.
- Las queries de prueba, es decir, las instancias del caso de test.
- El resultado esperado, también en términos del objeto de test

Si bien para cada componente en el MT hay una contrapartida en el MTD, la idea de contexto en uno y otro modelo no parecen tener el mismo significado. El contexto en el MTD tiene que ver con tener al conjunto de datos en una determinada situación. Por ejemplo,

1. Si se pretende evaluar casos sobre los pagos cargados a un determinado sistema, esto debe hacerse luego de realizado el cierre del día, pues ése es el momento en el cual están consistentes.
2. Si se pretende evaluar casos sobre los datos de facturación de clientes, es necesario haber terminado el procesamiento de los consumos del mes.

Como conclusión de esto puede verse que de un modelo a otro el mecanismo de ejecución ha cambiado radicalmente.

Capítulo 3

Propiedades y queries: lenguaje de escritura

1. Introducción

Dentro del MTD encontramos dos elementos para los cuales es necesario elegir un lenguaje de escritura, a saber los casos de test (en realidad el lenguaje servirá para escribir las propiedades asociadas a ellos) y las queries (que servirán para realizar la ejecución del test).

Esta elección resulta importante debido a que marcará diversas pautas relacionadas con la actividad. Por ejemplo, el poder expresivo del lenguaje elegido para escribir las propiedades decidirá en gran medida el tipo de casos que será posible enunciar.

Por el otro lado, el lenguaje elegido para escribir las queries puede facilitar (o no) su ejecución sobre el conjunto de datos que pretende probarse.

En este capítulo se elegirán dichos lenguajes, además de presentarse una noción formal de instanciación de casos de test.

2. Expresividad

El tema de expresividad ha sido ampliamente tratado dentro del área de bases de datos. Dos referencias importantes del mismo son [ULLM89] y [ABIT95]. En ambos, se describen diversos lenguajes que permiten expresar consultas sobre datos. Cada uno de ellos posee sus propios mecanismos de expresión, los cuales nos permiten situarlo en un determinado nivel de expresividad. Dicho nivel se deduce en base al análisis de equivalencias entre él y los demás lenguajes.

Si bien no es el objetivo aquí presentar todos los posibles niveles de expresividad así como tampoco el sin fin de lenguajes existentes en la actualidad, resulta importante distinguir ciertos niveles básicos de expresividad que permitirán fundamentar la elección de un candidato para cada uno de los elementos del MTD.

La existencia de ciertos mecanismos expresivos importantes en uno u otro lenguaje determinarán en gran medida la elección a realizar. A continuación se discriminarán tres niveles de expresividad importantes que permitirán distinguir dichos mecanismos y facilitar la elección.

Primer nivel

El álgebra relacional (AR) resulta ser un representante conocido para el primer nivel de expresividad a la hora de trabajar con datos. Como se sabe este lenguaje trabaja sobre un esquema relacional y en él es posible realizar distintas operaciones: select operation, project operation, secuenciamiento de operaciones y renombramiento de atributos, join operation y el conjunto de operaciones de conjuntos. Como lenguajes equivalentes en cuanto a nivel expresivo es importante mencionar también al cálculo relacional safe (CR) y a DATALOG en su versión no recursiva y con negación estratificada.

Si bien las bases de datos relacionales se basan fuertemente en el AR para implementar sus lenguajes de consultas (por ejemplo SQL). Todos tienen una limitante importante: no permiten trabajar con la clausura transitiva de una relación. Esta posibilidad resulta importante al modelar jerarquías dentro de los datos. Por ejemplo, la consulta "cuáles son todos los jefes de A?" presentaría un problema irresoluble dentro del álgebra y una limitación inadmisible en una base de datos.

Si bien los lenguajes de consultas sobre bases de datos contemplan operadores para esto, el AR y por eso es necesario pasar al siguiente nivel.

Segundo nivel

DATALOG en su forma natural permite trabajar con recursión, pero no con negación. No obstante esto, constituye el escalón siguiente en la escala de expresividad.

La principal virtud de este lenguaje es que garantiza una ejecución en tiempo polinomial de sus consultas hecho a partir del cual es posible asegurar la computabilidad de las mismas.

Aquí consultas como por ejemplo "cuales son los nodos alcanzables desde el nodo A?" en un grafo dirigido son fácilmente expresables.

Sin embargo, la ausencia de negación todavía deja algunas otras fuera como por ejemplo, "cuales son los pares de nodos que no se encuentran conectados?"

Tercer nivel

Este nivel está representado por DATALOG con negación. En él es posible expresar la consulta mencionada en el nivel anterior. El poder expresivo alcanzado aquí resulta ser sumamente alto, aunque esto se traduce, obviamente, en un gran número de dificultades a la hora de implementarlo.

3. Propiedades

Como se dijo anteriormente, es necesario definir un lenguaje de escritura de propiedades. Para ello es necesario tener en cuenta algunos factores importantes.

- **Poder expresivo**. El lenguaje elegido debe poseer un poder expresivo razonable que permita escribir la mayor cantidad de consultas imaginables sobre el objeto de test. Si bien este conjunto resulta imposible de caracterizar, resulta obvio que la mayoría de los mecanismos mencionados en la sección anterior deberían estar resueltos en el mismo.
- **Modelo relacional**. El lenguaje elegido debe soportar al modelo relacional. Esto se debe a que la mayoría de los sistemas existentes están implementados según dicho modelo. Introducir modelos alternativos solamente complicaría la derivación de los casos de prueba.
- **Noción de instanciabilidad**. El lenguaje elegido debe permitir establecer una noción de instanciabilidad entre él y el lenguaje de queries seleccionado.

En concordancia con lo dicho, una elección razonable resulta ser el álgebra relacional, pero con ciertos agregados que permitan incrementar el poder expresivo. En [NAVA94] se sugiere agregar al álgebra un operador de clausura transitiva y ciertas "funciones agregadas" tales como la suma, la resta, el promedio, etc. Con ellos la expresividad aumenta situando al nuevo lenguaje en el nivel dos de la escala antes presentada[7].

Si bien es cierto que el nivel de expresividad del lenguaje elegido podría ser mayor no hay que perder de vista que la mayoría de las consultas realizables sobre un conjunto de datos pueden ser expresadas con este lenguaje. El hecho de que los lenguajes de bases de datos se basen en él sirve como prueba importante de esto. Además existe una noción simple de instanciación entre el AR y el lenguaje de escritura de queries elegido que asegura la posibilidad de derivar una query a partir de una propiedad en forma rápida.

4. Queries

En el caso del lenguaje de escritura de queries, también es posible enunciar algunos puntos de análisis importantes.

- **Computabilidad.** El lenguaje elegido debe ser computable puesto que el objetivo final del testing de datos es ejecutar los casos sobre el objeto de test y las queries son el medio para hacerlo.
- **Ejecutabilidad.** El lenguaje elegido debe permitir ejecutar a las queries con un esfuerzo de adaptación mínimo a la hora de realizar la ejecución. Por ejemplo, de nada serviría escribir la query en AR puesto que para ejecutarla sobre los datos deberíamos escribir un programa que la implemente.

Si bien algunos de los puntos resaltados para la elección del lenguaje de las propiedades también aplican para esta elección, sólo se mencionaron aquí los puntos nuevos todavía no tratados. Igualmente, los anteriores también han sido tenidos en cuenta en el análisis.

---

[7] No se dará aquí la definición de dicho lenguaje así como tampoco la de sus agregados. Para mayor detalles consultar las fuentes antes mencionadas.

El punto ejecutabilidad marca fuertemente la elección puesto que lo que se busca no es un lenguaje teórico que permita expresar cosas solamente, sino que además este lenguaje debería permitir ejecutar estas queries con poco esfuerzo. A partir de esto es que la elección recae sobre el SQL.

Este lenguaje, aún en su versión estándar posee las cualidades necesarias para el rol. Además de que algunas de sus implementaciones poseen operadores que aumentan considerablemente su poder expresivo (por ejemplo, operadores de clausura transitiva).

Aclaraciones sobre las elecciones realizadas

En primer lugar es necesario resaltar la naturalidad de las elecciones puesto que el álgebra relacional y el SQL son los lenguajes comúnmente usados para trabajar con los datos de una base de datos. Este no es un hecho casual y también ha sido tenido en cuenta en la elección.

Segundo, el hecho de haber realizado esta elección no invalida la utilización de otros lenguajes para la escritura de casos y queries. El objetivo aquí era elegir 2 candidatos que reunieran condiciones adecuadas en cuanto a la relación costo/beneficio existente entre los distintos factores relevantes como poder expresivo, amigabilidad del lenguaje, computabilidad, etc.

5. Las instancias del caso de test (nexo entre caso y test)

A partir del concepto de caso se articula el concepto de *dato* en el MT y el de *query* en el MTD. Si bien el objetivo de ambos es constituirse en "instancias" de los casos, la definición en cada uno es bastante distinta.

A continuación se presentará una definición formal de la idea de instanciación existente en el MTD a partir de la del MT. Para esto definiremos una función de instanciación asociada al caso de test que nos permitirá decidir si algo es o no una instancia de dicho caso.

La función de instanciación en el MT

Sea c el caso de test para el cual definiremos la función de instanciación, decimos que la función de instanciación de c es

$$f_c : \vec{T} \rightarrow Bool$$

donde T es una tupla de datos de test para el caso c. La idea de esta función es decidir si la tupla es una instancia del caso en cuestión o no. A partir de esto definimos a la función como

$$f_c(\vec{T}) = \begin{cases} true\ si\ p_c(\vec{T}) \\ false\ sino \end{cases}$$

donde p_c es el predicado asociado al caso c.

Ejemplo 1

Supongamos que estamos definiendo casos de test para la siguiente funcionalidad: una *función pertenece que dice si un punto pertenece a una recta dada*.

Las entradas para ella son el punto *p* y la recta *r* sobre la cual quiero saber si el punto está o no. Podemos definir dos casos de prueba interesantes los cuales presentamos aquí mediante sus predicados y datos asociados

Casos	Datos
$p \in r$	< (1,1) ; y = x + 1 >
$p \notin r$	< (1,0) ; y = x - 1 >

Observando los elementos mencionados, es posible ver que el primer dato cumple el caso 2 y que el segundo dato cumple el caso 1.

La función de instanciación en el MTD

Habiendo repasado la definición de instancia del MT, nos dedicaremos a definir su par en el MTD. Sea *c* el caso de test para el cual definiremos la función de instanciación, decimos que la función de instanciación de *c* es

$$f_c : Q \rightarrow Bool$$

donde Q es una query para el caso *c*. Y la idea de esta función es decidir si esta es una instancia del caso en cuestión o no. A partir de esto definimos a la función como

$$f_c(q) = \left\{ \begin{array}{l} true \; sii \; p_c(M) \supseteq f_q(M') \\ false \; en \; caso \; contrario \end{array} \right\}$$

donde p_c es la propiedad asociada al caso *c* la cual es evaluada en M (modelo relacional). A su vez, f_q es la evaluación de la query sobre M' (modelo físico).

Es importante notar la existencia de un isomorfismo entre el modelo relacional sobre el cual se evalúa la propiedad (M) y el modelo físico o de tablas sobre el que se evalúa la query (M'). Este hecho se da por definición de ambos modelos.

Por otro lado, la relación entre ambos miembros debe entenderse como una inclusión de conjuntos. Es decir que *una query es instancia de una determinada propiedad si y solo si el conjunto de tuplas que devuelve está incluido en el conjunto de tuplas que devuelve la propiedad.*

Ejemplo 2

A continuación veremos un ejemplo. El mismo nos servirá para mostrar la forma en que pueden escribirse tanto el caso de test como la query de test. Sea el siguiente modelo de datos:

Ilustración 4. Ejemplo para expresividad

En este modelo, la organización en cuestión mantiene una base de datos de clientes. Además, mediante un sistema, se cargan las ventas realizadas a cada cliente junto con la fecha en las que estas son llevadas a cabo. Un dato adicional es que la organización realiza las entregas a domicilio y la fecha que figura como atributo es la fecha de entrega.

Por este motivo una propiedad interesante que puede evaluarse aquí es la vigencia del domicilio del cliente. Sólo por simplicidad, elegiremos una definición básica de este atributo[8].

Si el cliente registra una venta posterior a una fecha determinada, quiere decir que el domicilio es vigente.

En primer lugar se muestra el caso expresado por medio de un predicado. Sea *a: Cliente* se pretende buscar a los clientes que cumplen esta condición

$$\exists v : Venta \quad tal\ que\ v.Fecha \geq FECHA\_VIGENCIA \wedge a.Id = v.IdCliente$$

Luego, es necesario determinar el resultado esperado para el caso. Dado que se espera que todos los clientes tengan su domicilio vigente, todos ellos deberían cumplir con esta condición.

Ahora se presenta la propiedad escrita en lenguaje relacional, es decir que se presenta el caso de prueba a ejecutar sobre nuestro objeto de pruebas.

$$\Pi_{Id}(\text{ Cliente}) - \Pi_{IdCliente}(\ \sigma_{Fecha \geq FECHA\_VIGENCIA}(\text{ Venta}))$$

El resultado esperado para la misma es que el conjunto de datos sea igual a $\varnothing$.

Ahora se presenta la query correspondiente

```
select ID From CLIENTE Where ID not in (
  select Idcliente from VENTA where FECHA >= FECHA_VIGENCIA)
```

El conjunto de datos arrojado en ambos casos debe ser igual, es decir que deben devolver las mismas tuplas. Por lo tanto la query en cuestión no debería devolver ningún elemento.

Obviamente, este resultado fijado de antemano para el caso y para la query es arbitrario. En determinados contextos, obtener que el 90% de las tuplas al ejecutar el caso también podría ser tomado como un caso de prueba exitoso. Todo dependerá de las circunstancias en las cuales se realiza la prueba, del nivel de calidad deseado y del dominio del problema.

---

[8] Esta dimensión, dependiendo del dominio del problema, puede resultar mucho más compleja en la práctica. Este precepto es aplicable también a las demás dimensiones de la calidad.

6. Conclusiones

A continuación se enumeran algunas conclusiones producto del análisis realizado.

1. El primer punto a resaltar de este análisis es el hecho de que el paralelo no puede hacerse de la forma más exacta. El debilitamiento de la noción de instancia originado por el gap existente entre p_c y f_T en la definición es el responsable directo de este hecho. Este debilitamiento puede ser un punto negativo en el paralelo pues es claro que el poder expresivo de los lenguajes del estilo SQL es menor que el de la lógica de primer orden o el equivalente elegido. A partir de esto, la dualidad en su sentido más formal no puede ser probada. Sin embargo, debería quedar claro que dicha expresividad alcanza para satisfacer nuestras necesidades en cuanto a la definición de propiedades y a la ejecución de los tests derivados de ellas (esto se verá en los experimentos de los próximos capítulos). Como corolario de lo dicho en este párrafo se desprende que las flechas marcadas como *1* y *1bis*, del diagrama de la sección *Modelo* , difícilmente simbolicen lo mismo.
2. A partir del análisis realizado, es posible notar la diferencia existente entre el comportamiento esperado en uno y otro modelo. Mientras en el MT el ideal teórico es expresar el resultado esperado en términos de verdadero o falso (a partir de la idea de caso como predicado lógico), en el MTD es posible hacer variar esta situación a fin de establecer una escala de valores más difusa que permita analizar más ricamente el resultado obtenido al ejecutar el caso. Ya no se piensa en que todos los elementos del conjunto de datos deben cumplir la propiedad, sino que en realidad se busca lograr cierto porcentaje de casos favorables. Este hecho está asociado a que la calidad de la información que está siendo evaluada no siempre puede establecerse de una manera tan absoluta. Por ejemplo, saber que el 90% de las tuplas de una entidad cumplen o no con determinada propiedad, puede ser importante. El nivel de calidad *necesario* no siempre deberá ser 100%, sino que en muchas ocasiones serán *suficientes* porcentajes menores.
3. También es necesario destacar que en el MTD se conserva la propiedad de intercambiabilidad de las instancias de los casos de test. Por ejemplo, el cambiar una query por otra (o un conjunto de queries por otro) puede solucionarnos por ejemplo problemas de performance. Una query extremadamente lenta podría ser cambiada por otra que consume menos tiempo. Sin embargo el cambio deberá ser cuidadosamente estudiado ya que la nueva query debe seguir siendo instancia del caso.

Capítulo 4

Niveles y tipos de testing

1. Introducción

Debido a la gran complejidad de los sistemas de software es común utilizar, a la hora de llevar a cabo cualquiera de las tareas relacionadas con el proceso de desarrollo, la estrategia de "dividir y conquistar". La misma consiste simplemente en atacar al problema por partes, buscando de esta manera, reducir su complejidad.

El testing de programas no es la excepción, y por lo tanto, es común encarar la prueba de los sistemas en base a niveles. Estos niveles permiten situar en contexto a la prueba y definir mejor el objetivo de la misma. De igual forma, el tipo de testing también constituye otro factor importante dentro de la actividad.

La definición de este tipo de esquemas resulta importante pues permite determinar varios puntos importantes en relación con la tarea de testing. A partir del nivel y tipo de testing elegidos, es posible establecer:

- el alcance de la prueba a realizar, entendiéndose por alcance los objetos que estarán involucrados en la misma;
- el objetivo perseguido durante la prueba (es decir, la estrategia para definir los casos de prueba), luego a partir de él;
- las técnicas aplicables en ella y, por consiguiente;
- el tipo de problemas o fallas que se encontrarán.

Creemos que estas mismas ideas son practicables en el MTD. Por lo tanto, en este capítulo definiremos un esquema de niveles de testing para el trabajo con los datos y estableceremos definiciones paralelas de *testing funcional* y *testing estructural* (los tipos de testing más comunes). Esto nos permitirá discutir los distintos tipos de fallas pueden encontrarse asociados a estos niveles y tipos de testing.

El objetivo final es establecer un marco teórico para la presentación de técnicas particulares en los capítulos siguientes. En ellos, se mostrarán dos técnicas suficientemente representativas con la idea de mostrar experimentos reales que den la idea de las posibilidades del modelo.

Se aclara sin embargo, que no se definirán técnicas para todas las posibles combinaciones de niveles y tipos de testing definidos aquí pues no es objetivo de este trabajo ser exhaustivos en ese sentido. Por el contrario, la meta es presentar un modelo de testing para datos que permita, en futuras investigaciones, descubrir más técnicas.

2. Niveles en el modelo de software

En [BEIZ90] se definen 4 niveles de testing, a saber, testing de unidad, testing de componente, testing de integración y testing de sistema. Para entender su significado es necesario conocer las definiciones de unidad, componente, integración y sistema:

- **Unidad.** Es la mínima pieza de software que tiene sentido testear.
- **Componente.** Es una agregación integrada de una o mas unidades.
- **Integración.** Es un proceso por el cual las componentes son agregadas unas a otras para crear componentes más largas.
- **Sistema.** Es un componente grande.

Como se dijo en la sección anterior, uno de los motivos por los cuales resulta conveniente el trabajo con niveles es la posibilidad de establecer mejor los diferentes tipos de errores. En este sentido, en [BEIZ90] se establecen dos tipos de errores importantes: de unidad y de componente. Notar la correspondencia existente con los niveles planteados pues los errores encontrados durante las pruebas de integración o de sistema, son también errores a nivel componente, esto puede verse a partir de las definiciones dadas.

3. Esquema de niveles propuesto

El esquema de niveles planteado para el testing de programas, se define en base a su estructura. Dicha estructura presenta un esquema de componentes organizado jerárquicamente. Así, el sistema está formado por componentes (el sistema es un componente) y cada componente es, a su vez, una agregación de unidades.

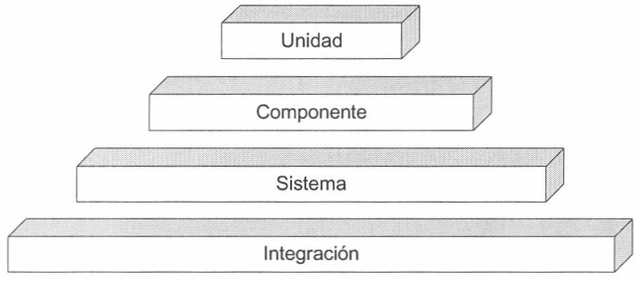


Ilustración 5. Niveles de testing

En el caso de los datos, puede aplicarse un planteo similar, es decir, utilizar su estructura para determinar los niveles de testing. El modelo relacional constituye entonces el punto de partida para dicha definición y es a partir de él que pueden entenderse las definiciones que presentaremos.

La definición de **unidad** mencionada para software en la sección anterior, puede también ser utilizada para los datos. En este caso la mínima unidad que tiene sentido testear es la tupla[9], es decir, una fila de alguna relación en particular del conjunto de datos. Si bien existen elementos más pequeños que la tupla dentro de la estructura relacional (los atributos), tomar como unidad algo más pequeño que una tupla tiene algunos inconvenientes:

- En primer lugar se pierde la idea de unicidad del dato. Por ejemplo, si tomáramos al Código Postal de una persona como una unidad, resultaría engorroso saber si el valor 1429 corresponde a la persona A o a la persona B (pues es posible que este atributo coincida para estas personas).
- En segundo lugar, es común ver atributos repetidos en diferentes relaciones de un modelo de datos, esto también agregaría confusión al trabajo.

Utilizando nuevamente las definiciones de la sección anterior, podemos ver que el equivalente a un **componente** en los datos es la **relación**. Justamente una relación en el modelo relacional es un conjunto (agregación integrada) de tuplas (unidades).

Para ser totalmente formales en este punto es importante aclarar que existe una diferencia entre software y datos: en software un componente es una unidad, mientras que en datos, un componente (relación) a lo sumo puede ser un conjunto de un elemento. Aún con este problema teórico, la definición resulta conveniente a nuestros propósitos.

El nivel de **sistema** tiene su equivalente para los datos. El esquema relacional que agrupa un conjunto de relaciones en forma consistente es este equivalente. En el testing de programas, el sistema está formado por un conjunto de componentes relacionados consistentemente que interactúan y lo mismo ocurre en el esquema relacional con las relaciones.

El último nivel que debemos tratar es el de **integración**. Este tipo de testing tiene por objetivo verificar que el comportamiento de las partes trabajando conjuntamente es correcto. En este contexto, se entiende por partes tanto a las componentes (relaciones) como a las unidades (tuplas) y la verificación se realiza bajo el supuesto de que estas partes funcionan correctamente [BEIZ90]. En el caso de los datos esta definición también es aplicable. De hecho, creemos que es el tipo de testing que mejor relación costo / beneficio trae (costo / cantidad de errores encontrados). A continuación se muestra una tabla con los términos en uno y otro contexto.

Software	Datos
Unidad	Unidad
Componente	Relación
Integración	Integración
Sistema	Esquema relacional

Tabla 2 . Comparación de términos entre el MT y el MTD

[9] Para más información ver [NAVA94].

Luego de establecer este esquema, es posible hacer un refinamiento del significado de la palabra testing en el contexto del testing de datos. Ahora, hablar de testing de unidad, de relación, de integración o de esquema presuponen alcances y objetivos distintos en la actividad.

Notar además, que el término esquema permite encuadrar a la actividad de testing no sólo sobre un único sistema sino entre varios de ellos. Este es un punto importante si pensamos que los datos generalmente son pensados como elementos asociados a un sistema y dependientes de él (aún cuando en muchas ocasiones esto no es del todo cierto).

4. Tipos de testing

En el MT la derivación de casos de prueba puede realizarse a partir de diversos criterios. Estos criterios, como hemos dicho, ayudan a situar en contexto las pruebas que se pretenden realizar. La división más común que se hace de ellos es la que distingue entre estructurales y funcionales. A continuación damos las definiciones para cada uno de ellos en dicho contexto (las mismas fueron extraídas de [BEIZ90].

- **Funcional.** El testing funcional o testing de caja negra tiene por objetivo probar el objeto de test en base a una definición de casos derivados a partir de las funcionalidades del mismo sin importar los detalles de implementación.
- **Estructural.** El testing estructural o de caja blanca tiene por objetivo probar el objeto de test en base a una definición de casos derivados a partir de la estructura que presenta el objeto de test.

Dichas definiciones también pueden ser utilizadas, al momento de diferenciar tipos de testing en el MTD. Sin embargo, resulta importante aclarar, que el concepto de funcionalidad en el testing funcional y el concepto de estructura en el testing estructural deberán ser interpretados en forma diferente dado que el objeto de test ahora está constituido por un conjunto de datos y no una pieza de software.

En esta sección se describirán los tipos de testing antes mencionados en el contexto del MTD.

Testing funcional en el MTD

En el MT los casos de test derivados a partir de criterios funcionales tienen por objetivo verificar que el objeto de la prueba, en este caso el sistema, cumple con la especificación y los requerimientos fijados para él (es decir que las funcionalidades han sido implementadas correctamente). Para realizar esta derivación de casos comúnmente se utiliza la información generada en las etapas de análisis y especificación del sistema. El conocimiento que surge del negocio también es una fuente importante para la derivación de casos funcionales.

Aplicando la misma idea en el MTD, los criterios funcionales para derivar casos tendrán que ver con verificar el cumplimiento de las especificaciones y/o requerimientos establecidos para los datos involucrados en la prueba.

El concepto de funcionalidad en este nuevo modelo está asociado ahora a los datos, y por esta razón, toda la documentación relacionada con el conjunto de datos que está siendo testeado será utilizada para la derivación de este tipo de casos.

En particular, la documentación referida a los modelos de datos involucrados, constituye un punto de partida importante. Por ejemplo, la información acerca de las dependencias funcionales del modelo o las especificaciones acerca de las reglas de negocio que gobiernan los datos de un determinado sistema son relevantes en este punto.

Testing estructural en el MTD

El testing estructural en el MT es el que basa su derivación de casos en la estructura del objeto de test. Para poder utilizar la misma definición que en el MT es necesario definir nuevamente el concepto de estructura ya que el objeto de test ha cambiado de un modelo a otro.

En el MT, el código determina la estructura del objeto de test. Por el otro lado, en el MTD, es el modelo de datos físico el que determina la estructura del objeto de test (el conjunto de datos).

En ambos modelos, el testing estructural presupone la derivación de casos sobre componentes concretos de la estructura, instrucciones del lado del software y entidades o relaciones del lado de los datos.

Los casos de test que pueden extraerse a partir de criterios como los aquí enunciados están principalmente orientados a analizar aspectos asociados a la representación de la información (teoría de bases de datos). Estos casos están íntimamente relacionados con ciertas dimensiones de la calidad asociadas a la representación (*consistencia o correctitud*) o al contexto (*completitud*) [WANG98]. De igual manera, dimensiones tales como relevancia u objetividad quedan fuera del alcance de este tipo de testing.

5. Clasificación de errores

En cada uno de los niveles y para cada uno de los tipos de testing definidos en el testing de programas es posible encontrar errores. Y, como corolario de la paradoja del pesticida de Beizer (ver [BEIZ90]), sabemos que los mismo responden a diferentes tipos de error. De otro modo habrían sido erradicados de los sistemas con alguna "milagrosa" técnica de testing de software.

A partir de este conocimiento es posible concluir que con el testing de datos sucederá algo similar. Es decir que tendremos diferentes tipos de error dependiendo del nivel del testing que realicemos y del tipo de testing que realicemos. Además sabemos ya de antemano que ninguna técnica sacará todos los errores existentes, pero todas ayudarán.

A continuación se presenta un diagrama que simboliza a la información disponible en una organización cualquiera. En ella existirán varios sistemas para cada uno de los cuales puede determinarse un modelo de datos. Entre ellos además, existe cierta información que es compartida o duplicada (esto se simboliza a través de las líneas que conectan los distintos modelos). Por ejemplo, en el diagrama, la entidad Cliente se encuentra replicada tanto en el sistema A como en el sistema B.

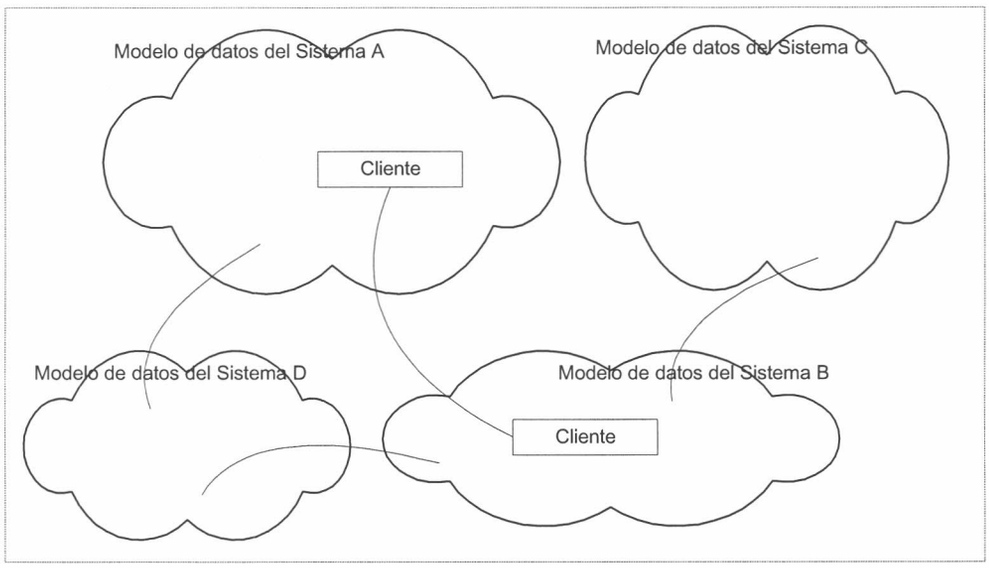


Ilustración 6. Esquema de niveles de información

Como puede verse en el diagrama, los niveles planteados definen el alcance que tendrá la actividad de testing. Así, el testing Unidad se practicará sobre cada registro de una entidad y estará aislado del resto del universo. Por ello se encontrarán aquí solamente problemas relacionados con los atributos: de *completitud* (atributos en cero o con valores nulos), de *integridad de dominio* (atributos que no se corresponden con el dominio del problema), de consistencia (reglas entre atributos que se conocen a partir del dominio del problema. Por ejemplo, en base a las dependencias funcionales existentes y no normalizadas).

El testing de módulo por su parte, se practicará sobre todos los registros de una entidad y estará aislado de las demás entidades. Se encontrarán aquí problemas relacionados con la *duplicidad* de los registros o la redundancia (Conciseness).

El testing de integración o de esquema se practicará sobre el modelo de datos correspondiente al conjunto de datos a probar. En este nivel es en donde se encuentran los problemas más importantes debido a que se evalúan las relaciones existentes entre las distintas entidades (ya sea dentro de un sistema o entre varios de ellos). Se encontrarán aquí problemas relacionados con la *correctitud* de la información, con su *vigencia*, con la completitud, etc. Es importante resaltar la importancia del concepto de integración aquí pues es común encontrar problemas de integración entre datos de diversos sistemas. Más aún si estos son del tipo legacy con escaso nivel de validaciones. Este tipo de testing se vuelve importante también en los sistemas del estilo data warehouse pues allí generalmente debe consolidarse información de diversas fuentes.

Capítulo 5

Modelo de cubrimiento

1. Introducción

Para realizar testing sobre un conjunto de datos es necesario definir casos de test y luego ejecutarlos. Si bien es posible encarar la tarea de derivación de casos directamente y sin ningún orden, resulta conveniente tener una medida de cuán adecuado es el conjunto de casos de prueba disponible para el objeto que pretende ser testeado.

El concepto de cubrimiento es un recurso de uso común en la actividad de testing de programas. Es utilizado tanto en técnicas estructurales como funcionales con buenos resultados. El mismo puede entenderse a partir de dos nociones:

1. La primera está asociada a la idea de "cubrir" un grafo elaborado a partir del objeto de testing sobre el cual se está trabajando.

   En el testing estructural, se utiliza el *grafo de control.* El mismo es derivado a partir del código del programa.

   En el testing funcional, el grafo es definido a partir de criterios funcionales y depende de la técnica utilizada. Por ejemplo, en el caso de la técnica Domain Testing, el grafo se confecciona en base al flujo existente en el procesamiento numérico de la funcionalidad a probar[10].
2. La segunda noción de cubrimiento está asociada a la determinación de cuando "parar" de hacer testing sobre un objeto determinado. Esto se traduce en tener una forma de determinar la cantidad de casos de testing necesarios para satisfacer un determinado nivel de calidad en el objeto de testing. Un ejemplo de esto, en el testing de programas, podría ser cumplir una premisa como la siguiente: "realizar determinadas pruebas sobre el 30% de las componentes de un sistema".

El objetivo ahora es definir nuevos criterios de cubrimiento para el modelo de testing de datos. A través de ellos será posible atacar sistemáticamente la tarea de derivación de casos de test en el nuevo modelo. Se buscará además establecer implicancias entre los criterios definidos que permitan deducir la utilidad de los casos de prueba en criterios para los cuales no habían sido pensados.

El concepto de cubrimiento se desarrollará a partir de la definición de un modelo de cubrimiento basado en un grafo de datos. Mostraremos un método de construcción para dicho grafo y definiremos ciertas anotaciones sobre el mismo que permitirán enriquecer los criterios antes mencionados.

Todo este análisis se hará utilizando un caso de estudio que permita mostrar las ideas teóricas sobre una situación real.

2. El modelo de datos

En el MT el llamado *grafo de control* es definido a partir de la estructura del objeto de pruebas. Este objeto, constituido por el código del programa, proporciona cierta semántica a cada uno de los elementos del grafo.

A partir de esta idea, en el MTD, es posible definir el *grafo de datos*. Este también estará definido a partir de la estructura del objeto de pruebas (el conjunto de datos a probar - CDP).

Dicha estructura no es tan homogénea como en el caso del MT pues el CDP puede estar constituido por elementos de diversa índole (los datos pueden abarcar más de un sistema, pueden involucrar varias plataformas o diferentes formatos, todo esto debido a la existencia de múltiples bases de datos o información en ambientes locales de usuarios como planillas Excel). Esta heterogeneidad hace necesario contar con alguna herramienta para mantener dicha información documentada y junta: El modelo de datos asociado al CDP.

En el caso de que el CDP este constituido por un único sistema o una única base de datos, el modelo será el del mismo sistema, sin embargo, si el CDP es más heterogéneo, el modelo deberá extenderse hasta contener toda la información necesaria. Por esta razón lo llamaremos modelo de datos extendido (MDE).

Idealmente, el MDE debería ser construido a partir del modelo de datos físico del o los sistemas involucrados. Esto se debe a que las querys obtenidas a partir de ese modelo serán fácilmente ejecutables luego.

Como normalmente se pretende evaluar la calidad de la información cruzando distintas fuentes, el MDE deberá contener todas las entidades involucradas en el CDP y no sólo las asociadas a un determinado sistema o base de datos.

3. Definición del grafo

Como primer paso para poder definir criterios de cubrimiento sobre el *grafo de datos*, es necesario proveer al mismo de cierta semántica, es decir, dar una definición exacta de un grafo de datos. Para ello en esta sección se enumeran todos y cada uno de los elementos involucrados en él a la vez que se provee de una semántica apropiada.

- **Nodos**. Los nodos representarán a cada una de las entidades de datos existentes en el MDE. La idea aquí es similar al DER y la correspondencia entre entidades y nodos es uno a uno.
- **Arcos**. Los arcos representarán las relaciones existentes entre las entidades al igual que el DER. Sin embargo, en el grafo de datos no figurarán las cardinalidades.

---

[10] La técnica mencionada se encuentra ampliamente mencionada en [BEIZ95]. Para mayor detalle, referirse a dicha fuente.

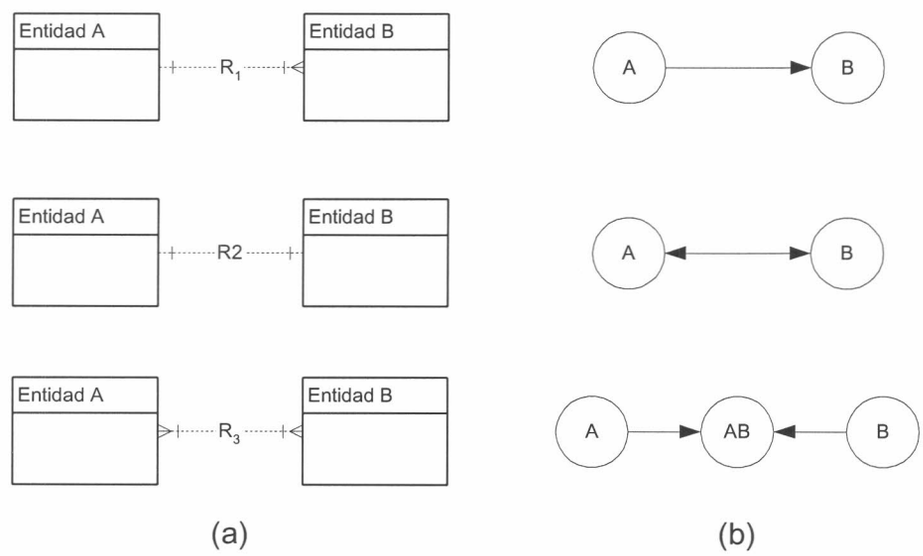
- **Dirección**. El sentido del grafo estará determinado por la *precedencia de carga* existente entre la información. Una flecha del nodo A al nodo B significará que es necesario cargar cierta información en A para luego poder cargar información en B[11]. Esta dependencia podrá deducirse a partir de la cardinalidad de las relaciones y del conocimiento existente sobre el CDP.

4. Construcción del grafo de datos

Como se ha dicho en la sección anterior, los nodos del grafo de datos estarán dados por las entidades del MDE y la correspondencia será uno a uno. Además, habrá tantos arcos como relaciones en dicho modelo.

El punto importante a resolver en la construcción de estos grafos es el sentido que se les dará a los ejes. Y como se ha dicho con anterioridad este dependerá en primer lugar de la cardinalidad de las relaciones.

Un punto importante a resaltar es que si bien la obligatoriedad de la relación es un punto importante a tener en cuenta en la definición de casos, a los fines prácticos de la construcción del grafo de datos, no cuenta.



Los caso que hay que resolver son los mostrados en el lado (a) de la figura. Y, como se dijo, esto se hará en base a la cardinalidad como muestra el lado (b):

1. En el caso de R_1 el sentido será de A hacia B. Puesto que siempre se necesitará tener información cargada en A para poder usarla en B.

[11] Esta *necesidad de carga*, obviamente puede ser explícita (constraints entre entidades) o implicita (la información en B no tiene sentido sin la información correspondiente en A).

2. En el caso *de* R_2, la dependencia es mutua y las entidades están "sincronizadas", esto hace que el arco deba tener aquí un sentido bidireccional. La relación uno a uno plantea una biyección entre las entidades.
3. En el caso de R_3, la dependencia también es mutua, pero algo más complicada que la anterior. Por esta razón, en el grafo se introduce un nodo intermedio que simula el reemplazo de la relación muchos a muchos por dos relaciones uno a muchos y una entidad débil[12].

5. Discusión

En esta sección se analizarán algunos puntos relacionados con el grafo definido y el método de construcción empleado.

Sobre la estructura y apariencia del grafo

A partir de lo descripto y de los conocimientos existentes sobre los grafos de control (GC) del MT, es posible hacer algunas observaciones sobre la estructura y apariencia del grafo de datos (GD) resultante en el MTD:

- Mientras el GC generalmente contiene un único nodo inicio (si hay más de uno se unifican de alguna manera), el GD presentará varias ocurrencias del mismo[13]. Dado que el grafo simboliza la precedencia de carga, las nociones de nodo inicial y final no serán importantes aquí. El punto crítico en este esquema consiste en establecer, dados dos nodos, quién necesita información y quién la provee. Esto se verá al momento de derivar casos de prueba en los próximos capítulos.
- Para ambos tipos de grafos, el nodo final puede repetirse varias veces (aunque en el caso del GC no es usual y se deba solamente a una mala programación). Aquí valen los mismos comentarios que en el punto anterior.
- Mientras en los GC pueden presentarse caminos largos en los GD está característica estará más ausente. La noción de camino en el testing de datos pierde significativamente su importancia puesto que los casos serán derivados entre pares de nodos. En este contexto, un camino del grafo sólo indicará si tiene sentido o no derivar casos de prueba entre un par de nodos cualquiera. Los pares conectados por un camino son candidatos a formar un caso de prueba (dependiendo del criterio a satisfacer).

---

[12] Es interesante notar aquí que la forma de implementar este tipo de relaciones físicamente en una base de datos es justamente agregando una tabla que relacione los extremos A y B, por lo que la introducción de un nodo resulta bastante intuitiva. Sin embargo, como preferentemente estaremos trabajando con el modelo físico deberíamos encontrar este caso en raras ocasiones.

[13] En este contexto, un nodo de inicio es aquel que no recibe flechas y un nodo final es aquel que sólo recibe flechas.

- Los ciclos, al igual que los caminos, también pierden su significado. En el GD, un grafo con ciclos no tiene diferencias significativas respecto de otro sin ciclos. Su existencia significará solamente que habrá más pares de nodos candidatos sobre los cuales derivar casos de prueba.
- En los GD pueden existir varios ordenes parciales que satisfacen un grafo determinado. Esto quiere decir que no hay una manera determinística de recorrerlo.
- En ambos casos el grafo es conexo.

Automatización de la construcción

Un punto a destacar sobre la construcción de este tipo de grafos es que ésta puede ser realizada automáticamente. Dado un diagrama de entidad relación, es posible obtener un GD asociado por medios automáticos. Este hecho se debe a que el grafo no es otra cosa que un vista restringida del diagrama de entidad relación asociado al conjunto de datos sobre los cuales se está realizando testing. Además es posible ver que el método de construcción enunciado en la sección anterior es fácilmente implementable.

Por otro lado, en muchos casos puede resultar conveniente agregar mayor información luego de la construcción. Esta información adicional se traduce en nuevos arcos o nodos no existentes en el DER original, pero que aportan conocimiento relevante a la hora de definir casos de test (reglas derivadas del negocio, dependencias funcionales, etc.). Por ejemplo, cuando el MDE está compuesto por varios sistemas diferentes, es común encontrar cierta información en común (el sistema A tiene una tabla de clientes que debe ser igual a la del sistema B).

El uso de heurísticas que ayuden a desarrollar intuiciones sobre como realizar este tipo de tareas es fundamental. Por ejemplo, en el caso de estudio que se presentará en las próximas secciones se introducirán entidades que inicialmente no formaban parte del modelo de datos, pero que resultan fundamentales a la hora de analizar la calidad.

Granularidad del grafo

El grafo resultante del proceso de construcción muestra tantos nodos como entidades existen en el modelo de datos (llegado el caso, pueden introducirse nodos en base a entidades externas al modelo, pero que forman parte del contexto de los datos).

El grafo se plantea aquí a nivel entidad, sin embargo, sería posible plantearlo a nivel atributo?

En primer lugar es necesario señalar que el grafo resultante del proceso de construcción planteado es un grafo anotado (ver más adelante) y dichas anotaciones están basadas en los atributos de cada una de las entidades. De algún modo entonces, el nivel de detalle llega hasta el atributo. Si bien no todos los atributos son tratados, los más relevantes para el test sí forman parte de las anotaciones. Este hecho resulta conveniente porque en la mayoría de los modelos de datos existe un alto porcentaje de campos informativos que resultan importantes a la hora de realizar un análisis de la calidad de la información.

Segundo, la introducción de nodos que simbolicen atributos causaría algunos problemas de indefinición. Por ejemplo,

- la necesidad de tener que definir "sentido" entre los arcos que van de los atributos a la entidad sin una noción de cardinalidad asociada.
- el nodo "entidad" quedaría desvirtuado en su significado al tener estos nodos atributos alrededor.

Un tercer punto a mencionar es el hecho de que la estrategia de plantear el grafo a nivel atributo explícitamente ocasionaría un aumento considerable en el tamaño del grafo de datos volviendo más difícil aún las tareas posteriores de análisis.

Otra posibilidad, al momento de analizar la granularidad, es evaluar lo que significa cubrir una entidad, es decir, plantear criterios de cubrimiento de los datos una entidad particular.

Análisis estático vs. dinámico

Dado un objeto de test es posible verificar su nivel de calidad mediante la aplicación de técnicas de análisis estáticas y/o dinámicas. La diferencia fundamental entre ambas es que en las primeras el objeto de test no es ejecutado, mientras que en las segundas si lo es.

Existen ciertos criterios de calidad que pueden ser verificados estáticamente sobre el grafo de datos sin necesidad de llegar a la ejecución de casos de test. Ejemplos de este tipo de análisis pueden verse en el terreno de las bases de datos con sus técnicas de normalización. Si bien este tipo de técnicas mejoran la calidad de la información y son más fáciles de aplicar que las dinámicas, el beneficio obtenido es menor que en el caso dinámico.

A partir del uso de la información soportada por el modelo, es decir, los datos, es posible analizar también la calidad de la información, pero esta vez en forma dinámica. La idea es obtener, mediante la ejecución de casos, información acerca de la calidad del objeto de test. Dicha ejecución estará ligada a la utilización de los datos y no de su estructura, lo cual hace a la técnica más potente pues el nivel de granularidad del análisis es mayor.

El grafo de datos es de vital ayuda a la hora de utilizar estas técnicas puesto que permite realizar el análisis en forma sistemática y ordenada.

6. Presentación del experimento

Con el objetivo de mostrar la aplicabilidad del modelo de cubrimiento y hacer más comprensibles las explicaciones acerca de los criterios definidos para él, se utilizará un caso de estudio pequeño que servirá como ejemplo en diversos pasajes de la definición. En esta sección se dará una breve introducción al mismo que servirá para fijar el contexto asociado.

Las revisiones o inspecciones de código son una técnica de verificación de software bastante común dentro de las actividades de la Ingeniería del Software. Estas revisiones se realizan por diversas razones como por ejemplo, para la detección y corrección de fallas relacionadas con problemas de año 2000.

El sistema que utilizaremos para el presente experimento sirve para asistir en la tarea de administración de las sesiones de este tipo de revisiones. Esta herramienta permite, entre otras cosas:

- Almacenar sesiones de revisión.
- Almacenar las componentes revisadas en cada sesión (se entiende por componente a un fuente de código en algún lenguaje).
- Almacenar los incidentes encontrados en cada una de las sesiones asociados a las componentes revisadas asociándoles además un origen y un tipo de problema (atributo que involucra una criticidad asignada al incidente).
- Realizar el seguimiento de los incidentes y de las sesiones de manera de saber el estado de cada uno de ellos.
- Se permite asociar a cada componente, el sistema en el cual se usa y la procedencia del mismo.

El proceso seguido para realizar las revisiones de código puede resumirse de la siguiente manera:

1. Se toman los fuentes a revisar en una sesión.
2. Se revisan y se genera una planilla de incidentes (Excel).
3. Se carga la planilla de incidentes en la herramienta a la vez que se le entrega la planilla a las personas encargadas de la corrección de dichos incidentes.
4. Los correctores realizan las modificaciones en el código volcando dicha solución en la misma planilla.
5. Se carga la planilla de corrección a la base.
6. Cuando todos los incidentes de una sesión se han cerrado, se cierra la sesión.

La herramienta posee una serie de ABMs que permiten modificar cierta información básica, así como también algunos reportes pre establecidos.

El intercambio entre la herramienta y las planillas EXCEL no siempre se hace automáticamente sino que en algunas ocasiones es necesario realizar la actualización en forma manual. Y es en este punto en el que se producen la mayor cantidad de problemas de calidad. A continuación mostramos el MDE asociado.

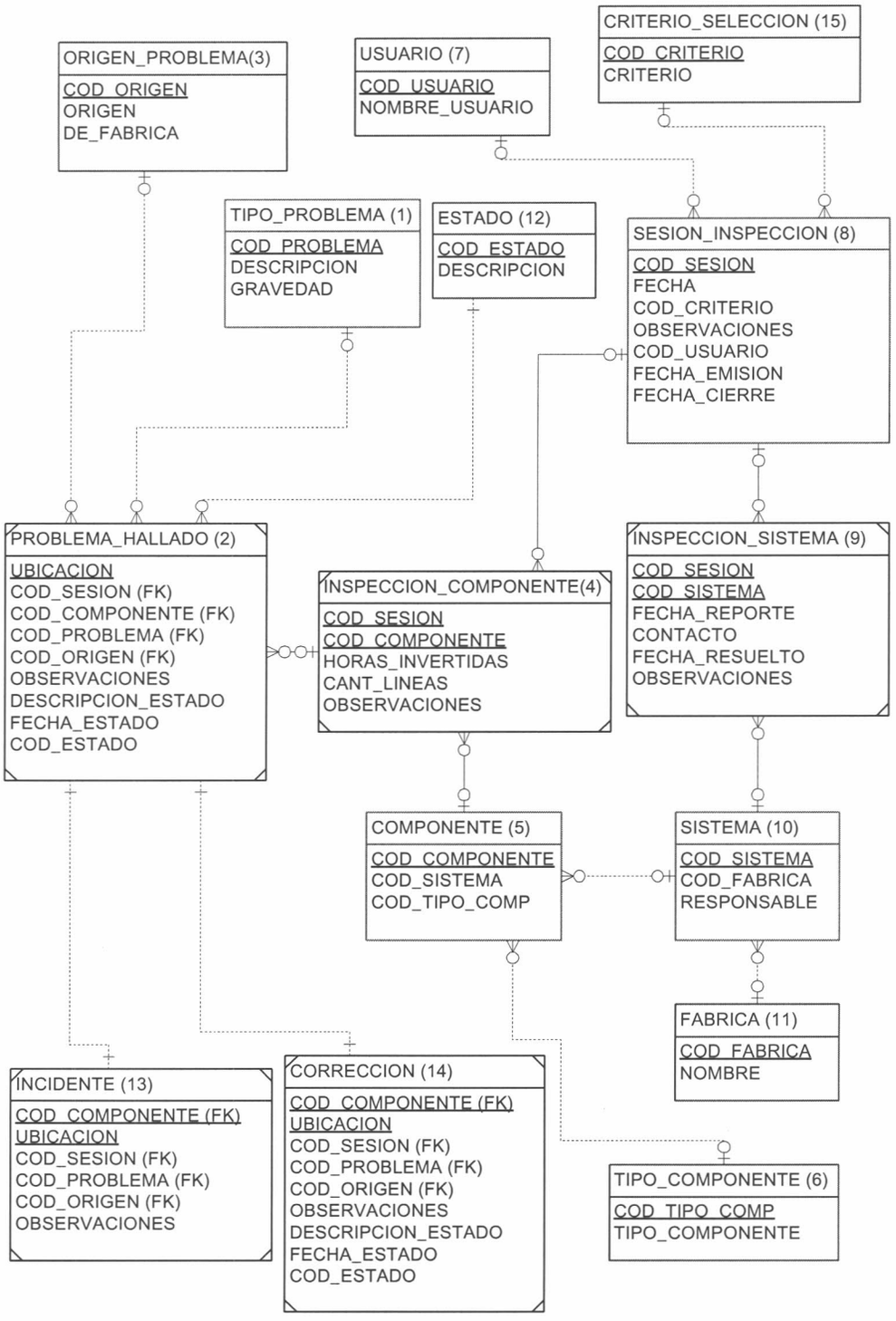


Ilustración 7. Diagrama de entidad relación asociado al experimento

Ahora, utilizando el método de construcción planteado anteriormente, generamos el grafo de datos correspondiente.

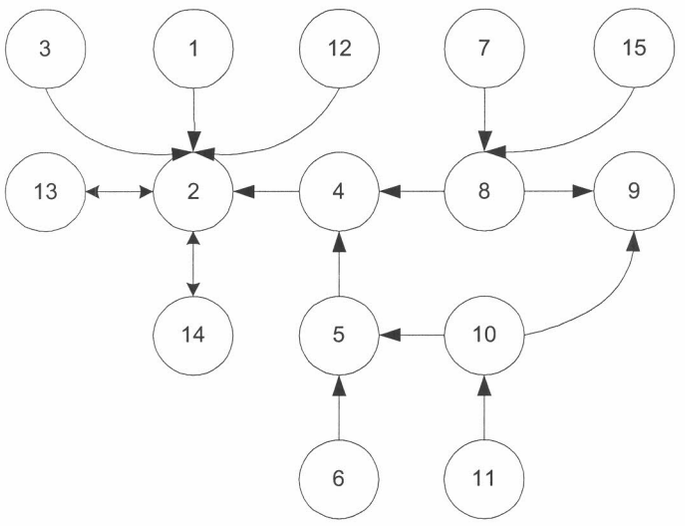


Ilustración 8. Grafo de datos asociado al experimento

En este grafo no hay relaciones muchos a muchos por lo que no fue necesario agregar ningún nodo. Sin embargo, se hace notar que el nodo 4 es el producto de la relación muchos a muchos entre los nodos 8 y 5. Esto surge a partir de observar que la entidad *inspección_componente* es una entidad débil que asocia a las entidades *sesion_inspección* y *componente*.

Las características marcadas anteriormente de este tipo de grafos de datos son observables en el ejemplo. Los caminos son cortos y hay varios nodos iniciales y finales. Además pueden verse dos ciclos (en este caso de relaciones uno a uno) que muestran la dependencia mutua entre entidades. Idealmente, esta biyección entre entidades significa que la información existente en ellas debería estar sincronizada. Sin embargo, en la práctica esto no ocurre pues 13 y 14 son planillas de cálculo separadas de las tablas de la aplicación (esto dificulta enormemente la sincronización).

Capítulo 6

Criterios de cubrimiento

1. Introducción

Un criterio de cubrimiento determina la forma en que se cubrirá el grafo de datos. Como dijimos anteriormente, el objetivo es establecer un mecanismo que sirva como guía para la tarea de definición de casos de prueba.

Para enunciar estos criterios de cubrimiento es posible utilizar distinto tipo de información como por ejemplo la estructura del grafo de datos; o también, el conocimiento existente del modelo de datos a partir del cual se derivó el grafo.

En primer lugar se trabajará con los que denominamos criterios *puramente estructurales*, que se basan exclusivamente en la estructura del grafo de control, para luego trabajar con los *semi estructurales*, que utilizan mayor cantidad de información. En ambos casos se analizarán las implicaciones existentes entre ellos.

2. Criterios Puramente estructurales

Como vimos anteriormente, a partir del objeto de test es posible derivar un grafo de datos asociado. La obtención de este grafo permite enunciar ciertos criterios de cubrimiento basados fuertemente en su estructura.

- **Todos los nodos (P_1)**. Un conjunto de casos de test satisface este criterio si cada nodo del grafo de datos es ejercitado al menos por algún caso de test.
- **Todos los arcos (P_2)**. Un conjunto de casos de test satisface este criterio si cada arco del grafo de datos es ejercitado al menos por un caso de test.

Los criterios enunciados mantienen una fuerte similitud con los criterios de Control Flow del MT. Esto se debe a que estos también se basan en la estructura de un grafo (el de control). De hecho, estos criterios no son los únicos posibles y para comprobar esto basta pensar en criterios como *todos los caminos* del MT para imaginar criterios de cubrimiento alternativos en el MTD.

Un punto importante a resaltar es que estos criterios resultan útiles únicamente al hacer un análisis dinámico del objeto de test puesto que se basan en la idea de ejercitar tanto a los nodos como a los arcos del grafo pero a nivel de datos.

Qué significa cubrir o ejercitar?

El punto inicial del análisis de estos criterios, es definir la noción de "cubrimiento" existente entre el caso de test y elemento del grafo correspondiente (el nodo o el arco respectivamente).

A continuación presentaremos el significado de "cubrir", o "ejercitar", para cada uno de los elementos significativos del grafo de datos.

Definición 2. *Un caso de test cubre un nodo o entidad n, si algún atributo del mismo se incluye en el enunciado del caso.*

Definición 3. *Un caso de test cubre un eje (e_1,e_2), siendo e_1 y e_2 entidades, si se incluyen, dentro del enunciado del caso, atributo/s de e_1 y atributo/s de e_2.*

Notar que en esta definición no se tiene en cuenta la forma en que se usa el elemento. Aquí es importante el aspecto estructural del grafo y si el nodo (o el arco) ha sido utilizado o no. Ahora bien, a partir de estas dos definiciones es posible inferir cierto conocimiento adicional el cual es reflejado en el siguiente lema.

Lema 1. *Sea c un caso cualquiera que cubre a un eje (e_1, e_2), entonces c cubre tanto al nodo e1 como al nodo e2.*

La demostración de este lema se deduce inmediatamente de las definiciones.

Implicaciones

Una vez definidos los criterios y el significado de "cubrir" en cada uno de ellos, resulta interesante preguntarse si existen implicaciones entre ellos. Es decir, si un conjunto de casos que satisface un criterio también satisface a algún otro.

Este conocimiento nos daría información importante a la hora de derivar casos de prueba, pues, por ejemplo, no sería necesario seguir derivando casos para el criterio C_1 si sabemos que $C_2 \Rightarrow C_1$ y los casos que tenemos ya satisfacen a C_2.

En vistas de lo dicho, el primer paso es definir formalmente la noción de implicancia.

Definición 4. *Un criterio C_2 implica a otro C_1 ($C_2 \Rightarrow C_1$), si todo conjunto de casos de prueba que satisface a C_2 satisface a C_1.*

En [BEIZ90] se establece la implicancia existente entre *Todos los caminos*, *Todos los arcos* y *Todos los nodos* en base al grafo de control.

$$P_\infty \Rightarrow P_2 \Rightarrow P_1$$

Ahora, en el MTD nos disponemos a probar la validez de la misma implicación entre *todos los arcos* y *todos los nodos*.

Lema 2. *$P_2 \Rightarrow P_1$*

Demostración 1. *Sea c(P_2) el conjunto de casos de test definidos para satisfacer el criterio P_2 sobre un grafo arbitrario G. Si c es un caso de test perteneciente a c(P_2), c cubre a un determinado arco (s, t) y por lo tanto c cubre tanto a s como a t (Lema 1). Como el grafo siempre es conexo, entonces todo nodo perteneciente a él, está asociado a un arco. De modo que c(P_2) satisface también a P_1*

3. Criterios semi estructurales

Los criterios puramente estructurales presentan una forma clara de cubrir al objeto de test, sin embargo, todavía existe más información útil dentro del conjunto de datos que puede ayudar a mejorar el conjunto de casos de test derivado. Disponer de criterios de cubrimiento que contemplen la mayor cantidad de información existente, permitirá aumentar el grado de confianza sobre este conjunto de casos.

En este caso, es posible destacar información adicional en forma de anotaciones dentro del grafo de datos. Para esto, en primera instancia se definirán las posibles anotaciones a realizar dentro del grafo para luego enunciar nuevos criterios que las tengan en cuenta.

Anotaciones

Las anotaciones permitirán reflejar información adicional en el grafo de datos. Luego, en base a esta información, será posible enunciar criterios más sofisticados que la contemplen.

- **Definición (d)**. Una definición está formada por un par [atributo, nodo] y especifica el lugar en el cual un determinado dato se origina.
- **Uso (u)**. Un uso está formado, al igual que la definición, por un par [atributo, nodo] y especifica el lugar en el cual un determinado dato es utilizado.

Para ejemplificar lo dicho, se muestra un extracto del grafo de datos confeccionado marcando los usos y definiciones existentes.

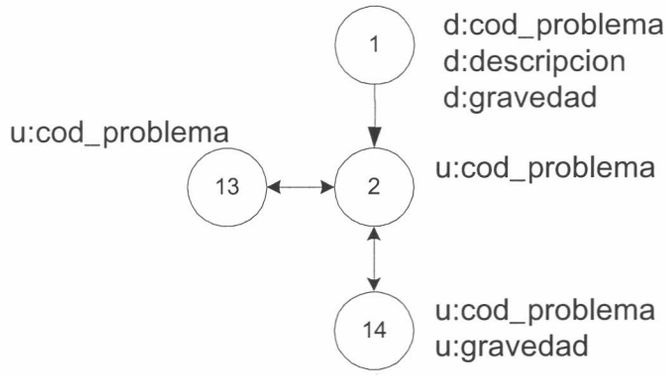


Ilustración 9. Extracto del grafo de datos asociado al experimento

Como se ve en el extracto, la definición del atributo *cod_problema* se realiza en la entidad *tipo_problema* y luego este código es usado en las entidades *problema_hallado, incidente y corrección.*

La forma en que puede determinarse si un atributo corresponde a un uso o a una anotación en una entidad determinada en principio es subjetiva y responde al criterio de la persona que está realizando la definición. No obstante esto, existen algunas heurísticas que ayudan a la hora de encontrar dichas anotaciones.

- Un atributo que forma parte de la clave primaria de la entidad en la que está, debe ser una definición.
- Un atributo sobre el cual existe una clave foránea, debe ser un uso.

Es importante notar que el atributo Gravedad de nuestro ejemplo, constituye un caso patológico para estas heurísticas. Este atributo no forma parte de la clave ni en el nodo 1 ni en el nodo 14. Además tampoco forma parte de ninguna clave foránea. Sin embargo es claro que, en el nodo 1, se trata de una definición y en el nodo 14 se trata de un uso.

Análisis estático

La verificación estática es la forma más simple de análisis de calidad posible y se realiza en base al grafo anotado. Utilizándolo, es posible validar cierto grado de correctitud en el modelo. Esta forma de trabajo es más parecida a la utilización de un checklist que a la ejecución de un caso de test.

A continuación se presentan algunos ejemplos de problemas encontrados a partir de este tipo de análisis:

- **Usos sin definiciones**. Se refiere a la existencia de un *uso* para determinado campo que no ha sido definido previamente en el modelo. Este tipo de problemas suelen ser inmediatamente descubiertos y solucionados durante la etapa de diseño del modelo de datos pues se trata de algo muy intuitivo. El valor agregado en este caso es mínimo.

- **Definiciones sin usos**. Se refiere a la existencia de una *definición* de un determinado campo que luego, no ha sido usada en el modelo. Este tipo de problemas en un modelo de datos extenso pueden llegar a producirse con mayor frecuencia que los del criterio anterior. Si bien el valor agregado que se proporciona en este caso es mayor que el anterior, en realidad, la característica fundamental es sencillamente señalar cierta redundancia en el modelo.
- **Múltiples definiciones**. Se refiere a la existencia de múltiples definiciones para un mismo campo en el modelo. Otra vez, en un modelo extenso, es posible que este tipo de problemas sean pasados por alto. Este punto tiene mayor valor agregado que los anteriores debido a que señala eventuales puntos de conflicto en el modelo. En este sentido, la aparición de este tipo de problemas alertan sobre posibles inconsistencias en la información.

Como puede verse, la contribución del análisis estático es limitada, sin embargo, es necesario resaltar que resulta extremadamente barato implementarlo pues, tanto la obtención del grafo anotado con usos y definiciones como la verificación de cada uno de los criterios, puede hacerse en forma automática.

- El grafo de datos puede obtenerse automáticamente a partir de un diagrama de entidad relación, esto es deducible del proceso de construcción detallado anteriormente.
- La anotación de dicho grafo puede hacerse a partir de la información de dicho diagrama haciendo cierto análisis ad-hoc de la misma. Por ejemplo, en el extracto, es posible determinar que en 1 la aparición de *cod_problema* resulta ser una definición y en 2 resulta ser un uso. Esto se debe al carácter inicial del nodo 1 (el nodo 2 no es inicial). Si bien este tipo de reglas no son estrictas, siempre es posible hacer una primer marcación automática sin mayores problemas (la cual resultará bastante aceptable en muchos casos). Luego, esta marcación, puede ser complementada manualmente con el fin de otorgar mayor exactitud al grafo.

Además, es importante aclarar que el término "problema" no es el más adecuado para nombrar a los elementos encontrados en dicho análisis estático. En este sentido, es posible que el nombre "candidato a problema" o "warning" sea más exacto. Al encontrar uno de estos elementos es necesario analizar si nos encontramos ante un problema o no. Es común que ciertas decisiones de diseño, conceptualmente incorrectas desde el punto de vista del modelado de los datos, sean tratadas correctamente por la aplicación. En estos casos, la calidad de la información no se resiente.

Análisis dinámico

Los mismos criterios enunciados anteriormente pueden aplicarse también en forma dinámica. La forma estática se basa en una comprobación sintáctica sobre las anotaciones del grafo (en definitiva, los campos del modelo de datos). Esta diferencia entre la aplicación estática y la dinámica puede verse más claramente en un ejemplo.

En el extracto del grafo de datos, se verifica que el uso del campo *cod_problema* en el nodo 2 tenga su correspondiente definición en algún otro nodo (nodo 1). Si esto es no es así, estamos en presencia de un error en el modelo que luego originará fallas en los datos.

La forma dinámica, por su lado, busca ampliar el detalle del análisis. Para seguir con el ejemplo y sabiendo que el campo *cod_problema* tiene un uso en el nodo 2 y una definición en el nodo 1 (es decir que cumple el caso estático), se busca analizar si todas las instancias de *cod_problema* están en dicha situación. En otras palabras, se busca ver que para cada registro de la tabla del nodo 2, existe un registro con el mismo *cod_problema* en el nodo 1 y para esto podría definirse un caso de test que evaluara dicha propiedad.

A continuación se presentan algunos problemas encontrados a partir del uso de esta técnica[14]:

- **Definiciones sin usos**. Existen registros en el conjunto de datos de test que sirven de definición, sin embargo, estas definiciones no están siendo usadas por nadie. Esto provoca un problema de *Conciseness* en la base, hay información que no es necesaria para la organización.
- **Usos sin definiciones**. Existen registros en el conjunto de datos de test que constituyen información relevante para la organización, pero que no posee su correspondiente definición. Esto denota un problema de *completitud* de la base de datos.

Criterios

A partir del grafo de datos anotado es posible enunciar criterios de cubrimiento que utilizarán esta nueva información disponible. A continuación se presentan estableciéndose una definición paralela a partir del MT.

		Testing de programas[15]	**Testing de datos**
D_2	Todos los usos	Se requiere que al menos un camino libre de definición, que vaya de cada definición de cada variable hasta cada uso de esa definición, sea ejercitado bajo algún test.	Se requiere que al menos un camino libre de definición, que vaya de cada definición de cada atributo hasta cada uso de esa definición, sea ejercitado bajo algún test.
D_1	Todas las definiciones	Se requiere que toda definición de toda variable sea cubierta por lo menos una vez por un uso de dicha variable.	Se requiere que toda definición de todo atributo sea cubierto por lo menos una vez por un uso de dicho atributo.

Tabla 3. Definiciones de criterios semi estructurales

[14] Se notará que los tipos de problemas son similares a los encontrados con las técnicas estáticas, sin embargo, existe una diferencia fundamental. Aquí se encuentran a nivel instancia o registro (información en principio más valiosa), mientras que en el anterior se encuentra a nivel modelo.

[15] Las presentes definiciones han sido extraídas de [BEIZ90], al igual que la definición de DU path y camino libre de definiciones.

Al igual que para los criterios puramente estructurales, es necesario definir aquí dos puntos fundamentales:

1. El significado de "cubrir" para los nuevos elementos introducidos
2. Las implicancias existentes entre ellos (y en relación a los criterios anteriores)

En relación al primer punto, se presentan a continuación las definiciones correspondientes.

Definición 5. *Un caso de test cubre a una definición (a, e), con a atributo perteneciente a la entidad e, si el mismo es incluido en el enunciado del caso.*

Definición 6. *Un caso de test cubre a un uso (a, e), con a atributo perteneciente a la entidad e, si el mismo es incluido en el enunciado del caso.*

El segundo punto no resulta tan intuitivo como en el caso puramente estructural. De hecho, la conjetura es que no hay implicaciones entre estos criterios (ya sea entre sí o en relación a los puramente estructurales) y hasta en algún caso es posible demostrarlo con un contra ejemplo (esto se verá en más detalle en los resultados del experimento). A continuación se presenta una consecuencia importante surgida a partir de las definiciones..

Lema 3. *Si un caso c cubre a una definición (d, e) o a un uso (u, e), entonces el caso c cubre al nodo e.*

La demostración de este lema se deduce inmediatamente de las definiciones.

4. Resultados del experimento

En este capítulo hemos definido 4 criterios a partir del modelo de cubrimiento descripto. Además hemos establecido una relación implicancia entre dos de estos criterios y conjeturado que no había más relaciones de ese estilo entre los demás.

Ahora para finalizar, presentaremos como resultado del experimento, los casos de test derivados para satisfacer cada uno de los criterios definidos. Mostraremos algunas de las queries instanciadas a partir de ellos junto con los resultados obtenidos al ejecutarlas. Además mostraremos el grafo anotado a partir del cual se ha generado el conjunto.

Grafo de datos anotado

El grafo de datos junto con el método de construcción del mismo han sido explicados en el capítulo anterior. A continuación se detallan, en forma de tabla, las anotaciones para dicho grafo. Las definiciones han sido marcadas con una "D" y los usos han sido marcados con una "U". También han sido marcadas las definiciones sin usos asociados y, en el caso de los usos, han sido marcadas las entidades y atributos que definen lo definen.

Una vez construido el grafo, es necesario "anotar" los atributos asociados a cada uno de los nodos. Para esto resulta conveniente recorrer el grafo por todos sus caminos (desde los nodos iniciales hasta los finales).

Los nodos iniciales típicamente no contendrán usos debido a que no tienen información precedente (por eso son iniciales). Los nodos no iniciales podrán contener tanto definiciones como usos.

Un punto importante a resaltar es que durante la anotación es posible encontrar definiciones sin usos en el modelo. Dichos faltantes pueden ser encontrados en salidas propias de los sistemas que manejan estos datos (por ejemplo en las pantallas o los reportes). En este sentido podría ser necesario introducir nuevos agregados al grafo para tratar a esta clase de elementos. En el presente trabajo, sin embargo, hemos atacado solamente a aquellas definiciones que sí poseen usos dentro del modelo.

#	Entidad	Atributo	Anotación
1	Tipo_problema	Cod_problema	D
		Descripcion	D
		Gravedad	D
2	Problema_hallado	Cod_componente	U, componente.cod_componente
		Ubicacion	U, incidente.ubicación
		Cod_sesion	U, inspeccion_sistema.cod_sesion
		Cod_problema	U, tipo_problema.cod_problema
		Cod_origen	U, origen_problema.cod_origen
		Observaciones	U, incidente.observaciones
		Descripcion_estado	R,U, correccion.descripcion_estado
		Fecha_estado	R,U, correccion.fecha_estado
		Cod_estado	R,U, estado.cod _estado
3	Origen_problema	Cod_origen	D
		Origen	D
		De_fabrica	D *(no tiene uso asociado)*
4	Inspeccion_componente	Cod_sesion	U, inspeccion_sistema.cod_sesion
		Cod_componente	U, componente.cod_componente
		Horas_invertidas	D *(no tiene uso asociado)*
		Cant_lineas	D *(no tiene uso asociado)*
		Observaciones	D *(no tiene uso asociado)*
5	Componente	Cod_componente	D
		Cod_sistema	U, Sistema.cod_sistema
		Cod_tipo_comp	U, tipo_componente.cod_tipo_comp
6	Tipo_componente	Cod_tipo_comp	D
		Tipo_componente	D *(no tiene uso asociado)*
7	Usuario	Cod_usuario	D
		Nombre_usuario	D *(no tiene uso asociado)*
8	Sesion_inspeccion	Cod_sesion	D
		Fecha	D *(no tiene uso asociado)*
		Cod_criterio	U, criterio_seleccion.cod_criterio
		Observaciones	D *(no tiene uso asociado)*
		Cod_usuario	U, usuario.cod_usuario
		Fecha_emision	D *(no tiene uso asociado)*
		Fecha_cierre	D *(no tiene uso asociado)*
9	Inspeccion_sistema	Cod_sesion	U, sesion_inspeccion.cod_sesion
		Cod_sistema	U, Sistema.cod_sistema
		Fecha_reporte	D *(no tiene uso asociado)*
		Contacto	D *(no tiene uso asociado)*
		Fecha_resuelto	D *(no tiene uso asociado)*
		Observaciones	D *(no tiene uso asociado)*
10	Sistema	Cod_sistema	D
		Cod_fabrica	U, Fabrica.cod_fabrica
		Responsable	D *(no tiene uso asociado)*
11	Fabrica	Cod_fabrica	D
		Nombre	D *(no tiene uso asociado)*
12	Estado Incidente	Cod_estado	D
		Descripción	D
13	Incidente	Cod_sesion	U, inspeccion_sistema.cod_sesion
		Cod_componente	U, componente.cod_componente
		Ubicacion	D

#	Entidad	Atributo	Anotación
		Cod_problema	U, tipo_problema.cod_problema
		Cod_origen	U, origen_problema.cod_origen
		Observaciones	D
14	Correccion	Cod_componente	U, componente.cod_componente
		Ubicacion	U, incidente.ubicación
		Cod_sesion	U, inspeccion_sistema.cod_sesion
		Problema	U, tipo_problema.descripcion
		Origen	U, origen_problema.origen
		Observaciones	U, incidentes.observaciones
		Criticidad	U,tipo_problema.gravedad
		Descripcion_estado	D
		Fecha_estado	D
		Estado	U, estado.descripcion
15	Criterio_selección	Cod_criterio	D
		Criterio	D *(no tiene uso asociado)*

Tabla 4. El grafo anotado

Obtención de un caso de test

Una vez construido el grafo de datos anotado, es necesario hacer la definición de casos de test. Un caso de test para satisfacer un criterio de testing estructural o semi estructural se define en base a dos nodos (uno contendrá una definición y el otro un uso).

Además, dependiendo del criterio a satisfacer, será conveniente pensar el caso partiendo de la definición o partiendo del uso. Veamos un ejemplo de cómo derivar un caso de prueba.

Ejemplo 3

Supongamos que se pretende cubrir la definición del atributo *cod_problema* existente en el nodo TIPO_PROBLEMA. Debido a que un caso de test en este contexto debe plantearse entre una definición y un uso (un DU path), es necesario elegir un uso para el atributo. Por ejemplo, el nodo PROBLEMA_HALLADO contiene un uso para dicho atributo. En caso de tener conocimiento funcional del problema, esta elección puede hacerse con más precisión.

A esta altura es necesario decir que el caso de prueba dependerá fuertemente del criterio de cubrimiento elegido. Supongamos entonces que pretendemos satisfacer el criterio de *todas las definiciones*. El objetivo de dicho criterio es verificar que todas la instancias definidas poseen un uso asociado en las demás entidades, por este motivo, el caso puede enunciarse de la siguiente forma:

$$\Pi_{cod\_problema}(TIPO\_PROBLEMA) - \Pi_{cod\_problema}(PROBLEMA\_HALLADO)$$

Intuitivamente, lo que se pretende decir con él es que no debe haber *cod_problema* definidos en la entidad TIPO_PROBLEMA que luego no sean usados en PROBLEMA_HALLADO.

Para completar el caso de test es necesario determinar cual es su resultado esperado. Para definirlo hay básicamente dos posibilidades. La primera opción, es tomar el siguiente criterio: no debe haber tuplas seleccionadas en el resultado (es decir $\varnothing$). La segunda opción es establecer un porcentaje máximo de tuplas permitidas. Por ejemplo hasta un 10% del total de la entidad.

Como se verá en las tablas de casos presentadas a continuación, se ha tomado la postura más arbitraria para la evaluación del experimento.

Es importante notar que el caso propuesto cubre la definición del nodo 1 (la noción de "cubrir" explicada en secciones anteriores establece que es posible utilizar cualquier uso y no todos los usos). Definiendo los casos necesarios para cubrir las demás definiciones es posible terminar de satisfacer el criterio.

El procedimiento general para satisfacer un criterio determinado, es recorrer definir casos de test que "cubran" cada elemento (definiciones, usos, nodos o arcos) de cada atributo. Luego de ello, se definen las queries.

Todas las definiciones

A continuación se presentan los casos derivados para satisfacer el criterio de *Todas las definiciones*. Se hace notar que todas aquellas definiciones carentes de uso no serán cubiertas. Esto se debe a que una definición sin uso declarado, nunca tendrá una instancia de dicho uso.

	Caso	R.E.
TD01	$\Pi_{cod\_problema}$(TIPO_PROBLEMA) - $\Pi_{cod\_problema}$(PROBLEMA_HALLADO)	∅
TD02	$\Pi_{descripcion,\ gravedad}$(TIPO_PROBLEMA) - $\Pi_{problema,\ criticidad}$(CORRECCION)	∅
TD03	Π_{cod_origen}(ORIGEN_PROBLEMA) - Π_{cod_origen}(INCIDENTE)	∅
TD04	Π_{origen}(ORIGEN_PROBLEMA) - Π_{origen}(CORRECCION)	∅
TD05	$\Pi_{cod\_componente}$ (COMPONENTE) - $\Pi_{cod\_componente}$ (INCIDENTE)	∅
TD06	$\Pi_{cod\_tipo\_comp}$ (TIPO_COMPONENTE) - $\Pi_{cod\_tipo\_comp}$ (COMPONENTE)	∅
TD07	$\Pi_{cod\_usuario}$ (USUARIO) - $\Pi_{cod\_usuario}$ (SESION_INSPECCION)	∅
TD08	Π_{cod_sesion} (SESION_INSPECCION) - Π_{cod_sesion} (INSPECCION_SISTEMA)	∅
TD09	$\Pi_{cod\_sistema}$ (SISTEMA) - $\Pi_{cod\_sistema}$ (INSPECCION_SISTEMA)	∅
TD10	$\Pi_{cod\_fabrica}$ (FABRICA) - $\Pi_{cod\_fabrica}$ (SISTEMA)	∅
TD11	Π_{cod_estado} (ESTADO_INCIDENTE) - Π_{cod_estado} (PROBLEMA_HALLADO)	∅
TD12	$\Pi_{descripcion}$ (ESTADO_INCIDENTE) - Π_{estado} (CORRECCION)	∅
TD13	Π_{fecha_estado} (CORRECCION) - Π_{fecha_estado} (PROBLEMA_HALLADO)	∅
TD14	$\Pi_{ubicacion,cod\_sesion,cod\_componente,cod\_problema}$(INCIDENTE)-$\Pi_{ubicacion,cod\_sesion,cod\_componente,cod\_problema}$ (PROBLEMA_HALLADO)	∅
TD15	$\Pi_{Observaciones}$(INCIDENTE)-$\Pi_{Observaciones}$ (PROBLEMA_HALLADO)	∅
TD16	$\Pi_{descripcion\_estado}$ (CORRECCION) - $\Pi_{descripcion\_estado}$ (PROBLEMA_HALLADO)	∅
TD17	$\Pi_{cod\_criterio}$ (CRITERIO_SELECCION) - $\Pi_{cod\_criterio}$ (SESION_INSPECCION)	∅

Tabla 5. Casos derivados para el criterio Todas las Definiciones

A partir de la derivación de estos casos es posible hacer algunas observaciones importantes.

- Varios de los casos derivados podrían ser planteados indistintamente para otras entidades. Por ejemplo el caso 2 podría ser planteado con las entidades PROBLEMA_HALLADO o INCIDENTE en forma indistinta. Se deduce de esto que no existe una única forma de cubrir una definición. Sin embargo, si al momento de la definición se cuenta con más información relacionada con los datos, será posible elegir mejor los usos a utilizar.

- En el caso 2, se incluyen dos definiciones, una para el atributo gravedad y otra para el atributo descripción. Esto se debe a que ambas definiciones se necesitan entre si a la hora de derivar el caso, para identificarse en forma más precisa. Como ocurrirá en otras ocasiones también, el derivar el caso con un predicado menos restrictivo lo debilitará permitiendo que se filtren tuplas con información inconsistente.
- En el caso 14, el cubrimiento de la definición podría realizarse solamente con *Ubicacion,* sin embargo el introducir más campos refuerza el caso. Por ejemplo, podría suceder que se encontraran registros con esa ubicación, pero que pertenecieran a otras sesiones o componentes. En el caso 16, podría hacerse algo parecido al caso 14, pero la probabilidad de ocurrencia de los casos más detallados es demasiado baja. Además el objetivo aquí es cubrir la definición y esto ya ha sido logrado con el caso presentado (se muestran ambos ejemplos para marcar la diferencia existente)[16].
- En 8 se derivó el caso con la entidad INSPECCION_SISTEMA, sin embargo, bien podría haber sido usada la entidad INSPECCION_COMPONENTE. Esta elección arbitraria tiene una connotación importante a resaltar. El caso elegido cubre el eje (8,9) y no el (4,8) como habría sucedido si se elegía la otra entidad. A partir de esto es posible ver que el eje (4,8) no será cubierto por este conjunto de casos. Y, de esta manera, el criterio de Todos los ejes no será cubierto. Este hecho sirve para demostrar que *Todas las definiciones* no implica *Todos los ejes* a diferencia de lo que ocurre en el modelo de testing de programas.

Todos los nodos

Una de las formas más comunes de derivar casos de prueba en el modelo de software es generar los casos necesarios para satisfacer un determinado criterio y luego, en base a este, evaluar el porcentaje de cubrimiento alcanzado con ese conjunto de casos en otros criterios. Precisamente esto es lo que se busca ahora.

En base al conjunto de casos definidos en la sección anterior es posible calcular el porcentaje de cubrimiento alcanzado en *Todos los nodos*. A partir de la noción de qué significa "cubrir" un nodo es posible ver que todos ellos han sido cubiertos por los casos definidos anteriormente, de esta manera, puede concluirse que el conjunto de datos de la sección anterior cubre 100% el criterio de todos los nodos. Si bien en este caso particular la implicancia se cumple, no puede inferirse del ejemplo que el criterio de *todas las definiciones* implique al de *todos los nodos*.

Todos los ejes

Utilizando la misma idea, es posible verificar que el porcentaje de cubrimiento alcanzado por el conjunto de casos definido sobre *Todos los ejes*.

---

[16] Notar que aquí estamos haciendo uso de cierto conocimiento funcional sobre los datos.

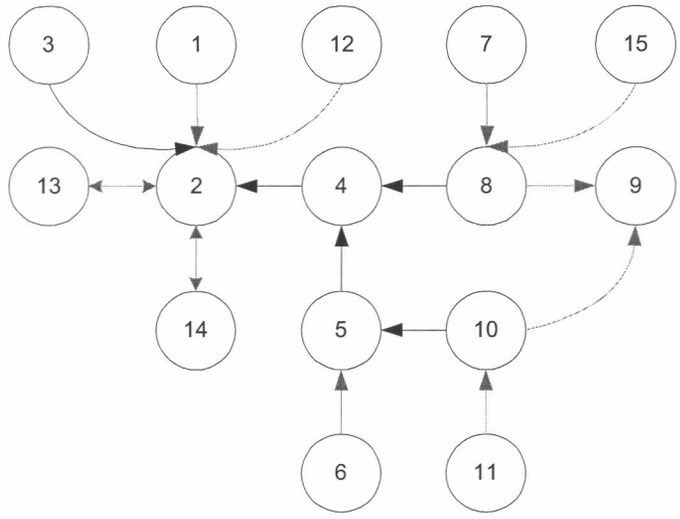


Ilustración 10 . Grafo cubierto

Se utiliza la noción de "cubrir" un eje y se colorean con rojo los ejes cubiertos por el conjunto de casos. A partir del grafo coloreado es posible ver que los ejes del grafo no han sido cubiertos en su totalidad. Solamente se han cubierto 10 de los 15 ejes por lo que el porcentaje de cubrimiento asciende al 67%.

Este resultado permite sacar una conclusión importante, ya antes anticipada. No es cierto que el criterio de *todas las definiciones* implique al de *todos los* ejes. Este hecho constituye una diferencia fundamental con el modelo de testing de programas.

Ahora, se enunciarán el resto de los casos necesarios para cubrir el 100% del criterio.

	Caso	RE
TE01	Π_{cod_origen}(ORIGEN_PROBLEMA) - Π_{cod_origen}(PROBLEMA_HALLADO)	∅
TE02	$\Pi_{cod\_componente}$(COMPONENTE) - $\Pi_{cod\_componente}$(INSPECCION_COMPONENTE)	∅
TE03	$\Pi_{cod\_sistema}$ (SISTEMA) - $\Pi_{cod\_sistema}$ (COMPONENTE)	∅
TE04	Π_{cod_sesion} (SESION_INSPECCION) - Π_{cod_sesion} (INSPECCION_COMPONENTE)	∅
TE05	$\Pi_{cod\_sesion,cod\_componente}$ (INSPECCION_COMPONENTE) - $\Pi_{cod\_sesion,\ cod\_componente}$ (PROBLEMA_HALLADO)	∅

Tabla 6. Casos derivados para el criterio Todos los Ejes

Se obtiene entonces, con el agregado de estos nuevos, un conjunto de casos de test con el cual se observa un porcentaje de cubrimiento del 100% para el criterio de todos los ejes.

Notar sin embargo, que la tarea se simplifica debido a la existencia del conjunto definido para el criterio anterior. Esta es una propiedad bastante común en el testing de programas que se da inclusive entre criterios funcionales y estructurales.

Todos los usos

Hasta aquí hemos derivado un conjunto de casos que satisface *todas las definiciones* y luego lo hemos aumentado para lograr satisfacer *todos los ejes*.

Para obtener el conjunto de casos que satisfaga el criterio de *Todos los usos* podríamos seguir el mismo procedimiento. Sin embargo, definiremos el conjunto de casos desde cero. Esto se debe a que los casos para verificar este criterio es conveniente pensarlos a partir de los usos que se pretenden cubrir. En este caso, la forma de derivación de casos se invierte. Se recorren todos los usos, eligiendo para cada uno una definición que complete el DU path.

	Caso	RE
TU01	$\Pi_{cod\_componente}$ (PROBLEMA_HALLADO) - $\Pi_{cod\_componente}$ (COMPONENTE)	∅
TU02	Π_{cod_sesion} (PROBLEMA_HALLADO) - Π_{cod_sesion} (INSPECCION_SISTEMA)	∅
TU03	$\Pi_{cod\_problema}$ (PROBLEMA_HALLADO) - $\Pi_{cod\_problema}$ (TIPO_PROBLEMA)	∅
TU04	$\Pi_{ubicación, cod\_componente, cod\_sesion, cod\_problema}$ (PROBLEMA_HALLADO) - $\Pi_{ubicación, cod\_componente, cod\_sesion, cod\_problema}$ (INCIDENTE)	∅
TU05	Π_{cod_origen} (PROBLEMA_HALLADO) - Π_{cod_origen} (ORIGEN_PROBLEMA)	∅
TU06	Π_{cod_estado}(PROBLEMA_HALLADO) - Π_{cod_estado} (ESTADO_INCIDENTE)	∅
TU07	$\Pi_{observaciones}$ (PROBLEMA_HALLADO) - $\Pi_{observaciones}$ (INCIDENTE)	∅
TU08	$\Pi_{descripcion\_estado}$ (PROBLEMA_HALLADO) - $\Pi_{descripcion\_estado}$ (CORRECCION)	∅
TU09	$\Pi_{fecha\_estado, cod\_estado}$ (PROBLEMA_HALLADO) - $\Pi_{fecha\_estado, cod\_estado}$ (CORRECCION)	∅
TU10	$\Pi_{cod\_componente}$ (INSPECCION_COMPONENTE) - $\Pi_{cod\_componente}$ (COMPONENTE)	∅
TU11	Π_{cod_sesion} (INSPECCION_COMPONENTE) - Π_{cod_sesion} (INSPECCION_SISTEMA)	∅
TU12	$\Pi_{cod\_sistema}$ (COMPONENTE) - $\Pi_{cod\_sistema}$ (SISTEMA)	∅
TU13	$\Pi_{cod\_tipo\_comp}$ (COMPONENTE) - $\Pi_{cod\_tipo\_comp}$ (TIPO_COMPONENTE)	∅
TU14	$\Pi_{cod\_criterio}$(SESION_INSPECCION) - $\Pi_{cod\_criterio}$ (CRITERIO_SELECCIÓN)	∅
TU15	$\Pi_{cod\_usuario}$ (SESION_INSPECCION) - $\Pi_{cod\_usuario}$ (USUARIO)	∅
TU16	Π_{cod_sesion}(INSPECCION_SISTEMA) - Π_{cod_sesion} (SESION_INSPECCION)	∅
TU17	$\Pi_{cod\_sistema}$ (INSPECCION_SISTEMA) - $\Pi_{cod\_sistema}$ (SISTEMA)	∅
TU18	$\Pi_{cod\_fabrica}$ (SISTEMA) - $\Pi_{cod\_fabrica}$ (FABRICA)	∅
TU19	$\Pi_{cod\_componente}$ (INCIDENTE) - $\Pi_{cod\_componente}$ (COMPONENTE)	∅
TU20	Π_{cod_sesion} (INCIDENTE) - Π_{cod_sesion} (INSPECCION_SISTEMA)	∅
TU21	$\Pi_{cod\_problema}$ (INCIDENTE) - $\Pi_{cod\_problema}$ (TIPO_PROBLEMA)	∅
TU22	Π_{cod_origen} (INCIDENTE) - Π_{cod_origen} (ORIGEN_PROBLEMA)	∅
TU23	$\Pi_{cod\_componente}$ (CORRECCION) - $\Pi_{cod\_componente}$ (COMPONENTE)	∅
TU24	Π_{cod_sesion} (CORRECCION) - Π_{cod_sesion} (INSPECCION_SISTEMA)	∅
TU25	$\Pi_{cod\_problema}$ (CORRECCION) - $\Pi_{cod\_problema}$ (TIPO_PROBLEMA)	∅
TU26	$\Pi_{ubicación, cod\_componente, cod\_sesion, cod\_problema}$ (CORRECCION) - $\Pi_{ubicación, cod\_componente, cod\_sesion, cod\_problema}$ (INCIDENTE)	∅
TU27	Π_{cod_origen} (CORRECCION) - Π_{cod_origen} (ORIGEN_PROBLEMA)	∅
TU28	$\Pi_{problema, criticidad}$ (CORRECCION) - $\Pi_{descripcion, gravedad}$ (ESTADO_INCIDENTE)	∅
TU29	$\Pi_{observaciones}$ (CORRECCION) - $\Pi_{observaciones}$ (INCIDENTE)	∅

Tabla 7. Casos derivados para el criterio Todos los Usos

A partir de los visto hasta ahora, es posible hacer más observaciones:

- Los casos derivados a partir de un uso son conceptualmente distintos a los derivados a partir de una definición. En el primer caso se intenta ver datos qué han sido introducidos sin una definición previa, mientras que en el segundo, se intenta ver qué definiciones no están siendo usadas y son superfluas. Digamos que mientras el primer caso tiene que ver con completitud, el segundo tiene que ver con conciseness.
- La tarea de derivación de casos de test debería resultar conocida e intuitiva puesto que un caso puede entenderse como una regla o propiedad sobre un determinado esquema relacional. La escritura de dichas reglas es una actividad común durante el diseño del modelo de datos de cualquier sistema de software. En particular, lo que se propone aquí es que el conjunto de propiedades definidas satisfaga ciertos criterios que harán más completo al conjunto de casos de prueba. Esta idea es usada comúnmente en el testing de software al momento de definir casos de prueba.
- Como se mencionara anteriormente, el concepto de camino no tiene la misma importancia que en el testing de software. Aquí los caso se establecen entre pares de nodos conectados y el camino sirve solamente para establecer si dichas conexiones son pertinentes.

Queries

A partir de los casos definidos para cada uno de los criterios, es necesario definir las queries a ejecutar sobre los datos. Gracias a la definición de instanciación presentada en el capítulo 3 es simple ver la forma en que estas deben derivarse. El método se resume a, dada una expresión escrita en lenguaje relacional, obtener una sentencia SQL que devuelva un sub conjunto de las tuplas devueltas por el caso asociado (obviamente, conviene que los conjuntos coincidan exactamente en la medida en que el caso lo permita).

A continuación presentamos las tablas correspondientes a las queries definidas para algunos de los casos anteriormente mencionados. Estás queries también están agrupadas por criterio, además se muestra para cada una de ellas, el resultado esperado y el resultado obtenido[17] en el experimento.

	Caso	RE	R. Obt.[18]
TD01	`Select * from TIPO_PROBLEMA Where COD_PROBLEMA not in (select COD_PROBLEMA from PROBLEMA_HALLADO)`	∅	2 / 24 regs 4 %
TD02	`Select * from TIPO_PROBLEMA where DESCRIPCION not in (select Problema from Correcciones)`[19]	∅	15 / 24 regs 62.5%
TD03	`Select * from ORIGEN_PROBLEMA`	∅	Ok

[17] Se hace notar que no se mostrarán los resultados obtenidos para todos los casos definidos puesto que no se cuenta con toda la información relacionada con las tablas Incidentes y correcciones.
[18] Aquí se coloca la cantidad de registros devueltos por la query sobre la cantidad de registros totales de la tabla y el porcentaje asociado.
[19] Esta query no respeta estrictamente el caso definido.

	Caso	RE	R. Obt.[18]
	where COD_ORIGEN not in (select COD_ORIGEN from PROBLEMA_HALLADO)[20]		
TD06	Select * from TIPO_COMPONENTE where COD_TIPO_COMP not in (select COD_TIPO_COMP from COMPONENTE)	∅	Ok
TD07	Select * from USUARIO Where COD_USUARIO not in (select COD_USUARIO from SESION_INSPECCION)	∅	3 / 12 regs 12.5 %
TD08	Select * from SESION_INSPECCION where COD_SESION not in (select COD_SESION from INSPECCION_SISTEMA)	∅	Ok
TD09	Select * from SISTEMA where COD_SISTEMA not in (select COD_SISTEMA from NSPECCION_SISTEMA)	∅	Ok
TD10	Select * from FABRICA where COD_FABRICA not in (select COD_FABRICA from SISTEMA)	∅	Ok
TD11	Select * from ESTADO_INCIDENTE where COD_ESTADO not in (select COD_ESTADO from PROBLEMA_HALLADO)	∅	4 / 10 regs 40 %
TD17	Select * from CRITERIO_SELECCION Where COD_CRITERIO not in (select COD_CRITERIO from SESION_INSPECCION)	∅	5 / 10 regs 50 %

Tabla 8. Queries derivadas para el criterio de Todas las definiciones

A partir de la ejecución de los casos relacionados con *Todas las definiciones* se han encontrado problemas de Conciseness en los datos. Cinco de las tablas evaluadas poseen registros que no son usados luego en el resto del modelo. Si bien este hecho por si sólo no es indicador de problemas de calidad, los porcentajes mayores al 40% hacen inducen a esta conclusión.

	Caso	R Esp.	R. Obt.
TE01	Select * from ORIGEN_PROBLEMA where COD_ORIGEN not in (select COD_ORIGEN from PROBLEMA_HALLADO)	∅	Ok
TE02	Select * from COMPONENTE where COD_COMPONENTE not in (select COD_COMPONENTE from INSPECCION_COMPONENTE)	∅	2890 / 10970 regs 26.3 %
TE03	Select * from SISTEMA where COD_SISTEMA not in (select COD_SISTEMA from COMPONENTE)	∅	Ok
TE04	Select * from SESION_INSPECCION where COD_SESION not in (select COD_SESION from INSPECCION_COMPONENTE)	∅	Ok
TE05	Select * from INSPECCION_COMPONENTE where COD_SESION not in (select COD_SESION from PROBLEMA_HALLADO) and COD_COMPONENTE not in (select COD_COMPONENTE from PROBLEMA_HALLADO)	∅	3938 / 12610 regs 33.2 %

Tabla 9. Queries derivadas para completar el criterio de Todos los ejes

El mismo problema señalado anteriormente se manifiesta aquí en dos tablas más. En la primera, por ejemplo, el problema resulta grave ya que funcionalmente hablando significa que hay 2890 componentes cargados en la sistema que no han sido revisados puesto que no figuran en ninguna sesión[21].

---

[20] Esta query se implementó sobre la entidad Problema_Hallado en lugar de Incidentes.

	Caso	R Esp.	R. Obt.
TU01	Select * from PROBLEMA_HALLADO where COD_COMPONENTE not in (select COD_COMPONENTE from COMPONENTE)	∅	Ok
TU02	Select * from PROBLEMA_HALLADO where COD_SESION not in (select COD_SESION from INSPECCION_SISTEMA)	∅	Ok
TU03	Select * from PROBLEMA_HALLADO where COD_PROBLEMA not in (select COD_PROBLEMA from TIPO_PROBLEMA)	∅	Ok
TU05	Select * from PROBLEMA_HALLADO where COD_ORIGEN not in (select COD_ORIGEN from ORIGEN_PROBLEMA)	∅	Ok
TU06	Select * from PROBLEMA_HALLADO where COD_ESTADO not in (select COD_ESTADO from ESTADO_INCIDENTE)	∅	Ok
TU10	Select * from INSPECCION_COMPONENTE where COD_SESION not in (select COD_SESION from INSPECCION_SISTEMA)	∅	Ok
TU11	Select * from INSPECCION_COMPONENTE where COD_COMPONENTE not in (select COD_COMPONENTE from INSPECCION_SISTEMA)	∅	Ok
TU12	Select * from COMPONENTE where COD_SISTEMA not in (select COD_SISTEMA from SISTEMA)	∅	Ok
TU13	Select * from COMPONENTE where COD_TIPO_COMP not in (select COD_TIPO_COMP from TIPO_COMPONENTE)	∅	Ok
TU14	Select * from SESION_INSPECCION where COD_CRITERIO not in (select COD_CRITERIO from CRITERIO_SELECCION)	∅	Ok
TU15	Select * from SESION_INSPECCION where COD_USUARIO not in (select COD_USUARIO from USUARIO)	∅	Ok
TU16	Select * from INSPECCION_SISTEMA where COD_SESION not in (select COD_SESION from SESION_INSPECCION)	∅	Ok
TU17	Select * from INSPECCION_SISTEMA where COD_SISTEMA not in (select COD_SISTEMA from SISTEMA)	∅	Ok
TU18	Select * from SISTEMA where COD_FABRICA not in (select COD_FABRICA from FABRICA)	∅	Ok

Tabla 10. Queries derivadas para el criterio de Todos los usos

Las queries derivadas para satisfacer el criterio de todos los usos no han reportado problemas de calidad en los datos. Esto puede explicarse a partir del hecho de que la interfase del sistema es robusta y las constraints del modelo se respetan en ella.

Resumen

1. Ya desde el análisis estático del grafo anotado es posible sacar conclusiones sobre la calidad del los datos de este sistema. La gran cantidad de atributos *"sin uso asociado"* existentes muestra que la información almacenada no es concisa. Evidentemente, habría que hacer un análisis funcional para determinar la gravedad de este problema de conciseness (ver si toda la información *"sin usos asociados"* es utilizada en formularios o reportes), pero el hecho de tener, ya desde el grafo anotado, una visión general que nos permita ver este tipo de cosas constituye un factor positivo para la técnica.

---

[21] Por convención de carga en la herramienta, para una sesión se ingresan todos los componentes revisados tengan o no incidentes reportados.

2. Este primer análisis estático del grafo que en esta ocasión sólo nos mostró problemas de conciseness, puede en otros casos mostrarnos problemas más serios en el modelo de datos.
3. Las implicaciones entre criterios postuladas a lo largo del capítulo han sido cumplidas por los casos derivados. Inclusive, el experimento proporciona una demostración de la no implicación entre dos de los criterios.
4. A partir de los casos de test derivados para el experimento se han corrido 28 queries sobre los datos. El resultado obtenido en cada ejecución puede ser categorizado de la siguiente manera:

Categoría	**Resultado**
Ok. Para las queries cuya ejecución ha devuelto cero tuplas (este era el resultado esperado en todos los casos)	21
Bajo. Esto quiere decir que se han detectado problemas de calidad con esta query, pero el porcentaje de tuplas con problemas es bajo (menor al 30%).	3
Medio. Esto quiere decir que se han detectado problemas de calidad con esta query, pero el porcentaje de tuplas con problemas es medio (entre 30% y 70%).	4
Alto. Esto quiere decir que se ha detectado un alto porcentaje de tuplas con problemas de calidad (mayor a 70%).	0

Se aclara sin embargo, que no se ha hecho un análisis exhaustivo del dominio del problema que permita establecer con un mayor nivel de certeza los límites superiores e inferiores para la categorización de los problemas encontrados. Evidentemente, un 30% de tuplas con problemas puede resultar a la vez inaceptable en un determinado sistema e insignificante en otros. En este caso, los rangos han sido puestos arbitrariamente.

5. Como se dijo anteriormente, el conjunto de queries definidas para satisfacer el criterio de todos los usos no ha detectado problemas. Este hecho puede atribuirse a la buena interfase del sistema que realiza las validaciones pertinentes durante la carga de los datos. La completitud de los datos evaluados parece tener un nivel aceptable en base a estos tests. Por su parte, el conjunto de queries definidas para satisfacer el criterio de todas las definiciones sí ha detectado problemas de calidad en los datos. En este sentido, el porcentaje de tuplas erróneas encontradas ha alcanzado hasta un 62%.
6. A partir del experimento se ha comprobado que la tarea de derivación tanto de casos como de queries es mecánica y automatizable en gran medida. Esto hace a la técnica muy atractiva aún ante el hecho de no presentar resultados contundentes hasta el momento.
7. A partir del análisis de los resultados obtenidos en las ejecuciones es posible ver que la decisión de definir el resultado esperado $\varnothing$ no siempre es acertada. Dentro de las queries ejecutadas existen ejemplos en los cuales el caso puede considerarse como exitoso, aún habiendo un resultado obtenido no vacío. El punto importante aquí es que cada caso de test debe ser evaluado por separado a la hora de decidir el resultado esperado.
8. Un interrogante importante que surge de este experimento es qué pasaría si se utiliza esta técnica en datos asociados a un sistema con tecnología más primitiva. Evidentemente, la efectividad de la técnica debería mejorar, pero resultaría muy útil poder determinar la magnitud de esa mejora.

Capítulo 7

Testing Funcional

1. Introducción

En los capítulos anteriores nos hemos ocupado de derivar casos de prueba partiendo de aspectos exclusivamente estructurales. Utilizando el objeto de pruebas se definió una abstracción de su estructura (grafo de datos) y luego criterios de cubrimiento asociados a ella. De este modo se lograba un conjunto de casos de prueba estructurales que permitía evaluar la calidad de los datos garantizando cierto cubrimiento de su estructura.

En este capítulo trabajaremos en la derivación de casos a partir de aspectos funcionales. Como se dijo anteriormente, el testing funcional tiene que ver con el uso del conocimiento proveniente "del negocio" al momento de la derivación de casos de prueba y no de su estructura. En este sentido, el objeto de test debe ser visto aquí como una caja negra[22]. En el caso del testing de datos, esto significa no hacer énfasis en la estructura del conjunto de datos sino en los aspectos relacionados con el negocio y la funcionalidad de los sistemas que los generan.

En este contexto es posible utilizar un concepto que hasta ahora no habíamos tenido en cuenta para la derivación de casos: *la dimensión*. El hecho de tener en cuenta las reglas del negocio en la derivación de casos, otorga cierta semántica a los objetos de prueba que no estaba presente al momento de la derivación de casos estructurales. Esta semántica puede ser caracterizada, por ejemplo, a partir de las distintas dimensiones de calidad definidas para los datos. Mostraremos aquí que el uso del concepto de dimensión permite orientar la tarea de derivación de casos al punto de transformarla en un ejercicio bastante intuitivo.

Entonces, el objetivo de este capítulo es establecer casos de test funcionales que permitan encontrar y/o prevenir problemas de calidad en un conjunto de datos. En particular, presentaremos una técnica que permite derivar este tipo de casos asociados a una dimensión. Estos casos servirán para validar calidad sobre un grupo de registros de una misma entidad. Para llevar a cabo nuestro propósito describiremos un modelo simple de la estructura de estos casos.

---

[22] El testing funcional es también llamado testing de caja negra en lo que a software se refiere [BEIZ90].

2. Modelo

Para poder utilizar una técnica para derivar casos de testing es necesario verificar que los supuestos establecidos para su aplicabilidad han sido satisfechos. Generalmente, estos supuestos son dados por medio de un modelo. Por ejemplo, para aplicar nuestros criterios de cubrimiento fue necesario contar con el modelo de datos extendido. A su vez, la existencia del modelo permite conocer de antemano qué clases de resultados serán obtenidos a partir de él. En otras palabras, a partir de este modelo es posible establecer el modelo de fallas asociado.

En este caso el modelo está compuesto por los siguientes elementos:

- Una **dimensión**. A partir de ella se establecerán las propiedades a evaluar en el objeto de pruebas.
- Una **entidad** (o más precisamente por un grupo de registros de dicha entidad). Éste será el objeto de pruebas sobre el cual se derivarán los casos de prueba y se ejecutarán los tests.
- Una **función de calidad** asociada a la dimensión. Será una o varias propiedades enunciadas sobre el registro a evaluar. Esta propiedad permitirá establecer un nivel de calidad del registro y por ende de la entidad. Sin embargo, es posible que la misma se establezca a partir de un predicado que relacione a varias entidades además de la que está siendo evaluada. Esta función puede describirse con lógica de primer orden, pero luego deberá ser traducida al lenguaje de escritura de los casos de prueba[23].

La idea aquí es trabajar sobre una dimensión a la vez para disminuir la complejidad de la prueba. A partir de esto, la función de calidad debería ser simple y su escritura no debería presentar problemas. Esto es importante porque el diagnóstico de los datos será realizado a partir de su ejecución.

Es importante mencionar que los casos de prueba (la función de calidad) se derivan sobre registros de una misma entidad, hecho que sitúa a la técnica en el nivel de testing de componente.

Los dos factores mencionados en el párrafo anterior determinan fuertemente el modelo de fallas asociado a la técnica puesto que los problemas encontrables estarán relacionados directamente con esa dimensión y esa entidad, aún cuando estos puedan ser un síntoma de problemas aún más generales. Esto puede ser visto como una restricción de la técnica, pero también como una ayuda al momento de aplicarla.

Obviamente, todavía es posible que el procedimiento mejore su performance. Es decir, que la relación casos de prueba / problemas encontrados mejore. Esto será posible mediante sucesivos refinamientos de la técnica. Por ejemplo tratando más de una dimensión a la vez o involucrando funciones complejas que trabajen sobre más de una entidad.

---

[23] Este punto es importante debido a que no será posible expresar cualquier función en el lenguaje definido.

3. Técnica

La técnica para derivar puede resumirse por medio del siguiente conjunto de pasos:

1. Se selecciona la dimensión con la que se va a trabajar
2. Se selecciona la entidad sobre la que se va a trabajar. Y el grupo de registros dentro de ella si fuera necesario.
3. Se establece la o las funciones de calidad que evaluarán a dichos registros.
4. Se escribe el caso correspondiente a cada función. Habrá un caso de pruebas por cada función de calidad.
5. Se ejecuta la query asociada al caso de prueba.

Evidentemente la complejidad de la técnica reside en los pasos 3 y 4. Para llevar a cabo el paso 3 es necesario tener un conocimiento funcional de los datos que se están testeando debido que deberán asumirse ciertas reglas sobre la entidad con la cual se trabaja.

El paso 4 por su parte consiste simplemente en escribir el caso de pruebas asociado a la función de calidad. Pero si bien es cierto que ambos pasos pueden hacerse en forma simultánea, en determinadas situaciones puede ser productivo hacerlo en forma separada (ver punto 4 de la discusión).

Para entender mejor la forma en que se realizan estos pasos presentaremos un ejemplo ya conocido. Sea el Ejemplo 2 del *Capítulo 3*

Propiedades y queries: lenguaje de escritura.

Ejemplo 4

En él se ha seleccionado la dimensión **vigencia** y se está trabajando sobre la entidad **cliente** (pasos 1 y 2). La función de calidad para un cliente *a* es (paso 3)

$$f(a) = \left\{ \begin{array}{c} true\ si\ \exists v : Venta \quad tal\ que\ v.Fecha \geq FECHA\_VIGENCIA \wedge a.Id = v.IdCliente \\ false\ en\ caso\ contrario \end{array} \right\}$$

Luego dicha función es embebida en el caso de pruebas (paso 4).

$$\Pi_{Id}(\text{ Cliente}) - \Pi_{IdCliente}(\ \sigma_{Fecha \geq FECHA\_VIGENCIA}(\text{ Venta}))$$

A partir de este caso es posible derivar la query asociada que nos servirá para ejecutar el test (esto se verá en el experimento).

4. Aplicabilidad sobre dimensiones y entidades

La tarea de definición de casos de prueba funcionales puede ser bastante engorrosa si el conjunto de datos a probar es grande. Por ejemplo, aplicar la técnica antes descripta en forma exhaustiva para toda dimensión conocida y para toda entidad del conjunto de datos puede no resultar demasiado redituable.

Por ello, se vuelve importante explorar formas de seleccionar en base a qué dimensiones y sobre qué datos se derivarán y ejecutarán casos.

Dimensiones

No todas las dimensiones de calidad son igualmente aplicables a la hora de definir casos de prueba y esto es aún más cierto si hablamos de casos funcionales derivados a partir de la técnica enunciada. Esto se debe a que puede resultar altamente complejo encontrar funciones de calidad para dimensiones como *relevance* o *reliability*. No obstante esto, otras dimensiones no sólo son utilizables como guía para la derivación de casos funcionales sino que también son utilizables a la hora de derivar casos estructurales como es el caso de *completeness* o *conciseness* (ver capítulos relacionados con cubrimiento).

En este contexto no es posible determinar en forma estricta cuál es el conjunto de dimensiones sobre el cual es posible derivar casos de prueba, puesto que esta decisión también depende del contexto del conjunto de datos que se está probando; sin embargo sí es posible enumerar algunas sobre las que puede ser muy aconsejable utilizar esta técnica.

Del conjunto de dimensiones enunciado en [BOBR98] podemos distinguir a:

- **Amount of data**. Número de hechos almacenados.
- **Consistency**. No hay contradicciones entre los datos almacenados.
- **Timeliness.** El dato está actualizado; la frecuencia de actualización es adecuada.
- **Conciseness**. El mundo real se encuentra representado con el mínimo conjunto de datos requerido.

Tipos de datos

Un posible camino para seleccionar qué entidades es conveniente testear se vislumbra a partir de clasificarlas en base a las características que tienen en común. Esto permitirá determinar de manera más precisa qué dimensiones de la calidad pueden tener mayor incidencia sobre ellas y cuál es su importancia real dentro del conjunto de datos que se pretende probar.

Las clasificaciones posibles para las entidades pueden variar significativamente. Aquí presentamos una bastante básica basada principalmente en la frecuencia de actualización.

1. **Dinámicos.** Los datos que caen dentro de esta categoría son aquellos que varían periódicamente. Esta variación puede darse mediante la modificación de un registro determinado o sencillamente por la aparición de registros nuevos. Ejemplos de ellos son el saldo de la cuenta corriente de un cliente en un banco, el consumo de su tarjeta de crédito, el total de puntos acumulados en una aerolínea, etc.

   La idea aquí es que el deterioro de la calidad de la información se dará en forma más abrupta, pues el ritmo de cambio es mayor. Por lo tanto una entidad de este tipo constituye un fuerte candidato a la hora de derivar casos funcionales sobre ella.

2. **Estáticos.** Son aquellos que poseen un ritmo de cambio bajo, esta información permanecerá vigente por mucho más tiempo al igual que su nivel de calidad. Dentro de esta categoría, es posible hilar más fino en la clasificación y distinguir dos sub categorías:

   - **De referencia.** Este tipo de datos estarán fijos en los sistemas casi eternamente (salvo esporádicas excepciones, como por ejemplo rectificaciones por errores de carga). Típicamente las entidades de referencias poseen una mínima cantidad de campos pues el objetivo de su existencia es codificar información. Aquí puede citarse como ejemplo entidades como provincias o localidades de un país. El trabajo de derivación de casos funcionales tiene menos sentido aquí.
   - **Básicos**: Este tipo de información, si bien es fija, podría cambiar llegado el caso más frecuentemente que la anterior. Esta categoría se refiere generalmente a los datos maestros de los sistemas. Como ejemplos de este tipo de datos podemos mencionar a los empleados de un banco o sus domicilios, etc. Éste es un tipo intermedio entre los datos dinámicos y los de referencia.

Esta clasificación permite ver que no todas las entidades tienen igual importancia a la hora de evaluar el nivel de calidad de la información. Evidentemente los casos de test relacionados con entidades más dinámicas serán de mayor utilidad y merecen mayor atención. Por ejemplo, el nivel de calidad en las entidades que cambian frecuentemente, puede disminuir más rápidamente que el de las entidades más estables.

5. Experimento y resultados obtenidos

En esta sección presentaremos algunos ejemplos y resultados obtenidos a partir del uso de la técnica. Continuaremos utilizando para ello el experimento presentado en el capítulo 5. Trabajaremos sobre dos dimensiones, a saber, *vigencia* y *consistencia*, haciendo especial hincapié en la primera.

Vigencia

Esta definición es lo suficientemente general como para permitirnos idear diversas técnicas de derivación de casos de prueba. En particular es practicable aquí la nuestra. Por esta razón hemos definido funciones de calidad para algunas tablas y hemos derivado los casos correspondientes.

En estos ejemplos se utilizó una fecha de vigencia como tope para la expiración de algunos registros, si bien es una idea simple, se verá en los ejemplos que es bastante efectiva. Por otro lado siempre es posible perfeccionar las funciones de calidad en base al conocimiento del negocio que se posea.

Ejemplo 5

"Una sesión de inspección es vigente si tiene fecha de cierre o si fecha de inicio es demasiado reciente."

Para *a: Sesión_Inspección* la función de calidad es

$$f(a) = \left\{ \begin{array}{l} true\ si\ \ fecha\_cierre \neq nulo \vee (fecha \neq nulo \wedge fecha \geq FECHA\_VIGENCIA) \\ false \quad\ \ en\ caso\ contrario \end{array} \right\}$$

Negando la condición, el caso resulta

$$\sigma_{fecha\_cierre=nuo \wedge (fecha=nulo \vee fecha<FECHA\_VIGENCIA)}(sesión\_inspección)$$

Y por lo tanto la query es

```
Select * From SESIÓN_INSPECCIÓN
 Where FECHA_CIERRE is null and (FECHA is null or FECHA < FECHA_VIGENCIA)
```

Obviamente, el resultado esperado aquí es que esta query no devuelva ningún registro. Sin embargo, al ejecutarla la misma devolvió 270 registros sobre un total de 282[24], es decir que el 95.7% de los registros de esta tabla son erróneos en base a este criterio de vigencia.

Ejemplo 6

"La inspección de un sistema, dentro de una sesión, es vigente si tiene fecha de resuelto o su fecha de reporte es demasiado reciente."

Para *a: Inspección_Sistema* la función de calidad es

$$f(a) = \left\{ \begin{array}{l} true\ \ si\ \ fecha\_resuelto \neq nulo \vee (fecha\_reporte \neq nulo \wedge fecha\_reporte \geq FECHA\_VIGENCIA) \\ false \quad\ \ en\ caso\ contrario \end{array} \right\}$$

Negando la condición, el caso resulta

$$\sigma_{fecha\_resuelto=nuo \wedge (fecha\_reporte=nulo \vee fecha\_reporte<FECHA\_VIGENCIA)}(Inspección\_Sistema)$$

La query es

```
Select * from INSPECCION_SISTEMA
 Where FECHA_RESUELTO is null and (FECHA_REPORTE is null or FECHA_REPORTE < FECHA_VIGENCIA)
```

Al igual que en el ejemplo anterior, el resultado esperado aquí es que la query no devuelva registros. Al ejecutarla devolvió 621 registros sobre un total de 783, es decir que el 79.3% de los registros de la tabla son erróneos en base a este criterio de vigencia.

---

[24] Se uso una fecha de vigencia de 3 meses hacia atrás. Esto se hará también en los próximos ejemplos.

Ejemplo 7

"Un problema es vigente si:

- *se encuentra en estado pendiente o derivado y la fecha de inicio de la sesión es demasiado reciente*
- *O se encuentra en un estado terminal (Cerrado, Obsoleto o Resuelto)."*

Para *a: Problema_Hallado* la función de calidad es

$$f(a) = \begin{cases} true & si\ (cod\_estado = 1 \vee cod\_estado = 7 \wedge \\ & \wedge \exists s : Sesión\_Inspección\ tal\ que\ s : cod\_Sesión = a.cod\_Sesión \\ & \wedge\ s.fecha \geq FECHA\_VIGENCIA\ ) \vee cod\_estado = 6 \vee \\ & \vee\ cod\_estado = 5 \vee cod\_estado = 2 \\ false & en\ caso\ contrario \end{cases}$$

A partir de la función se llega al caso de prueba. Debido a la complejidad del mismo, se ha subdividido la expresión siendo D el caso completo.

$$A = \Pi_{cod\_sesión, cod\_componente, Ubicación}(\sigma_{cod\_estado \neq 1 \wedge cod\_estado \neq 7}(\Pr oblema\_Hallado))$$

$$B = \Pi_{cod\_sesión, cod\_componente, Ubicación}(\Pr oblema\_Hallado \infty_{s.cod\_sesión = a.cod\_sesión}$$

$$\sigma_{fecha < FECHA\_VIGENCIA}(Sesión\_Inspección))$$

$$C = \Pi_{cod\_sesión, cod\_componente, Ubicación}(\sigma_{cod\_estado \neq 6 \wedge cod\_estado \neq 5 \wedge cod\_estado \neq 2}(\Pr oblema\_Hallado))$$

$$D = (A \cup B) \cap C$$

La query por su parte es

```
Select distinct p.COD_SESION, p.COD_COMPONENTE, p.UBICACION
  from PROBLEMA_HALLADO as p, SESION_INSPECCION as s
 Where ((p.COD_ESTADO<>1 and p.COD_ESTADO<>2)
    or (s.COD_SESION = p.COD_SESION and s.FECHA < fecha_vigencia))
   and (p.COD_ESTADO<>6 and p.COD_ESTADO<>5 and p.COD_ESTADO<>2)
```

El resultado esperado aquí es que la query no devuelva registros. Al ejecutarla devolvió 30 registros sobre un total de 1233, es decir que el 2.4% de los registros de la tabla son erróneos en base a este criterio de vigencia.

Consistencia

Las contradicciones que pueden ocurrir en los datos son de diversa índole: falta de validaciones por parte de los sistemas, bases de datos sin restricciones en sus tablas, mal modelado de los datos, falta de normalización, etc.

Cualquiera de estos factores puede ser atacado a partir del testing de datos. Aquí elegimos como ejemplo trabajar con el último motivo enunciado.

Ejemplo 8

"La inspección de un sistema, dentro de una sesión es consistente, si existe algún problema reportado para él."

Para *a: Inspección_Sistema* la función de calidad es

$$f(a) = \begin{cases} true & si\ (\exists\ p : \text{Pr}\ oblema\ \_\ Hallado\ tal\ que\ a.cod\ \_\ sesión\ =\ p.cod\ \_\ sesion) \wedge \\ & \wedge\ fecha\ \_\ reporte\ = nulo \\ false & en\ caso\ contrario \end{cases}$$

La query es

```
Select distinct COD_SESION from INSPECCION_SISTEMA
 Where FECHA_REPORTE is null and COD_SESION in (select COD_SESION from PROBLEMA_HALLADO)
```

El resultado esperado aquí es que la query no devuelva registros, no obstante, al ejecutarla devolvió 36 registros sobre un total de 783, es decir que el 4.6% de los registros de la tabla son erróneos en base a este criterio.

6. Discusión

En resumen entonces, a partir de cada uno de los casos de test derivados con esta técnica se ha definido una query para la ejecución del test. Por lo tanto se han ejecutado cuatro queries obteniéndose los siguientes resultados en todas las ejecuciones.

Para las queries definidas en base a criterios de vigencia, se han detectado problemas de calidad de niveles tanto altos como bajos (dos con niveles altos y una con nivel bajo).

Para la query definida en base a criterios de consistencia, se ha detectado un nivel bajo de problemas de calidad.

1. La técnica puede ser aplicada sobre cualquier dimensión y cualquier entidad. Sin embargo, encontrar una función de calidad para algunas combinaciones puede resultar una tarea bastante compleja y los resultados obtenidos luego pueden carecer de interés. Dimensiones como vigencia o consistencia son particularmente aplicables.
2. Generalmente resulta más cómodo escribir el caso de pruebas usando la negación de la función de calidad debido a que de esta manera se obtienen los registros con problemas en el resultado. Esto ya fue mencionado también para los casos estructurales y ésa es la forma en que se ha derivado el caso en el ejemplo antes mencionado.
3. Es importante notar que el caso definido en el ejemplo 3 cubre a la entidad desde el punto de vista estructural (es decir al nodo correspondiente del grafo de datos), pues cumple con la definición de "cubrir" planteada en el capítulo sobre cubrimientos. Este hecho hace a los casos funcionales derivados por medio de esta técnica aprovechables para verificar cubrimiento en el testing estructural. En este sentido, la combinación de técnicas funcionales y estructurales en el testing de datos produce un fenómeno similar al que ocurre en el testing de programas, donde es común analizar el grado de cubrimiento estructural alcanzado por el conjunto de casos funcionales definidos.

4. Ciertas funciones de calidad escritas en lógica de primer orden pueden no ser traducibles a AR (recodar que el poder expresivo de nuestro lenguaje de casos no es el mismo que el de la lógica). En esta situación no se logrará un caso a partir de dicha función, pero seguramente podrán derivarse varios sub casos de prueba.
5. Uno de los objetivos perseguidos al definir esta técnica es encontrar problemas de calidad en los datos. No obstante esto, la ausencia de problemas al ejecutar los tests no debe ser vista como un fracaso. La existencia de un conjunto de casos funcionales que al ser ejecutados garanticen cierta calidad en los datos es también un logro importante.

   Generalmente este tipo de casos encuentran errores en las primeras corridas y luego dejan de hacerlo pues los datos van corrigiéndose. De esta manera, las corridas sin problemas de este tipo de casos deben verse como un testing regresión sobre los datos que garantiza la no inclusión de nuevos problemas.
6. Esta técnica no constituye el único modo de derivar casos de prueba funcionales sobre los datos, sin embargo es un método simple con un amplio campo de aplicación lo cual constituye un buen punto de partida. Hemos visto que es aplicable sobre la *vigencia* o *consistencia*, pero bien podría trabajar sobre dimensiones más complejas y subjetivas como *relevancia* (en este caso, la definición de la función de calidad debería ser realizada con la ayuda de los usuarios de la información). También podría estudiarse su uso sobre combinaciones interesantes de dimensiones (por ejemplo, consistencia y conciseness).

Capítulo 8

Caso de estudio

1. Introducción

El objetivo del caso de estudio es evaluar las técnicas estructurales propuestas sobre un conjunto de datos distinto al evaluado en el experimento anterior. Los supuestos definidos para la elección del sistema fueron dos:

1. Debía estar construido con tecnología más antigua que el anterior (en lo posible, sin un motor de base de datos)
2. No debíamos tener conocimiento funcional alguno del sistema

El primer experimento correspondía a un sistema con un motor de base de datos que garantizaba ciertas "buenas propiedades" en ellos. Ahora, en el segundo, se pretendía observar el comportamiento sobre un conjunto de datos en el que el motor no tuviera incidencia. Este es el origen del primer supuesto.

El segundo supuesto nos asegura que el sesgo originado en el conocimiento funcional al momento de derivar casos será menor que en el experimento anterior. De esta manera el mérito al encontrar problemas de calidad será otorgado, en mayor grado, a la técnica utilizada.

A partir de estos requisitos, se eligió un sistema del área de explotación de petróleo como caso de estudio. Este sistema se encarga de almacenar la información relacionada con la exploración, explotación y perforación de pozos de petróleo en una determinada compañía.

En el contexto antes presentado, no es necesario incluir más detalles sobre aspectos funcionales. Sin embargo, sí es importante comentar ciertos puntos relacionados con la implementación del sistema, pues ayudarán a comprender mejor el experimento realizado.

El sistema en cuestión brinda la posibilidad a los usuarios de usar "popups". Esta funcionalidad consiste en que el sistema, para determinados campos de datos que deben ser completados por pantalla, permite cargar la información a partir de un menú de opciones. Obviamente, es posible no utilizar las opciones que aparecen en estas listas, si es que uno no lo desea y es justamente ahí en donde se produce el deterioro de la calidad. La organización, en este sentido, ha realizado diversos esfuerzos por unificar dichos "menú popups" y de este modo hacer la información más uniforme.

El objetivo del experimento, definido en términos de los datos, fue medir la calidad de la información existente en los distintos menú popups por medio de criterios estructurales.

2. Alcance del experimento

Para el presente experimento se utilizó un sub conjunto de los datos existentes en el sistema. Es decir el conjunto de datos a probar estaba compuesto por las 5 tablas más importantes del modelo. También fueron tenidas en cuenta las relaciones existentes entre dichas tablas.

Por otro lado, fueron incluidas dentro del conjunto, todas las tablas popup relacionadas con las tablas seleccionadas. De los 104 campos popups existentes en el sistema fueron tenidos en cuenta en este conjunto 20, es decir un 19.03%.

3. Desarrollo

Grafo anotado

A partir del conjunto de datos a probar se construyó el grafo de datos asociado utilizando el método presentado anteriormente. Los campos existentes en los nodos del grafo fueron anotados siguiendo el mismo método. Durante esta tarea los campos con definiciones sin usos dentro del conjunto de datos a probar fueron descartados. Por otro lado, es importante aclarar que las anotaciones se hicieron utilizando solamente la cardinalidad y la intuición desarrollada a partir de la lectura del modelo de datos.

A partir de realizado este procedimiento, se encontraron 22 definiciones con usos asociados y 26 usos relacionados con dichas definiciones.

Conjunto de casos de prueba

El paso siguiente fue derivar el conjunto de casos de prueba. A partir del grafo anotado, se derivaron 52 casos de prueba que satisfacían en un 100% los criterios estructurales definidos en capítulos anteriores. Es decir, Todas las definiciones, Todos los usos, Todos los nodos y Todos los ejes.

El resultado esperado definido para estas queries es la ausencia de tuplas en todas las ejecuciones. Esto quiere decir que, luego durante la ejecución del test, cualquier caso que no devuelva un conjunto vacío de tuplas será considerado como error.

Conjunto de queries a ejecutar

A partir del conjunto de caso de prueba se escribieron las queries asociadas. Se escribió una query por cada uno de los casos, por lo que el conjunto de queries definido contenía 52 elementos a ejecutar.

Ejecución y análisis del resultado obtenido

El paso siguiente fue ejecutar todas las queries del conjunto definido. Para analizar dicha ejecución se tuvieron en cuenta los siguiente elementos:

1. Cantidad de Tuplas Obtenidas en la query (CTO)

2. Cantidad Total de Tuplas existentes en la tabla sobre la cual se ejecuta la query (CTT)
3. Porcentaje de Error obtenido a partir de los elementos anteirores (PE)

Para poder analizar estos elementos fue necesario establecer pautas de análisis de los resultados. Esto posibilitaría determinar cuales ejecuciones habían terminado en forma incorrecta y cuáles no. Los criterios fijados para determinar la existencia de error fueron:

- Una query cuya CTO es cero, fue considerada como exitosa (OK).
- Una query cuya CTO no es cero, fue considerada como *errónea*.

Además se establecieron tres categorías de error posibles, debido a que los porcentajes entre las distintas ejecuciones variaban bastante.

- Gravedad Alta. Porcentajes mayores a 70%
- Gravedad Media. Porcentajes entre 30% y 70%
- Gravedad Baja. Porcentajes menores al 30%

4. Resultados

A partir de los resultados obtenidos sobre el caso de estudio es posible obtener conclusiones tanto sobre los datos evaluados como sobre el modelo y los criterios definidos. En esta sección trabajaremos sobre el primer punto, dejando para la sección de conclusiones los detalles relativos al segundo punto.

A continuación se presenta una tabla que resume los errores encontrados a partir de la ejecución. En la misma es posible ver la cantidad de queries ejecutadas con éxito y la cantidad de queries que dieron error para cada una de las tablas que forman parte del conjunto de datos.

Con el fin de permitir un análisis más detallado de los problemas encontrados, la tabla contendrá una discriminación por tipos de errores[25]. A su vez, se aclara que la clasificación en *alta, medio* o *baja* criticidad responde a los mismos rangos utilizados en el primer experimento.

[25] Defs: la query que produce el error ejercita de una definición de un popup, Usos: la query que produce el error ejercita un uso de un popup, Clave: la query que produce el error ejercita una definición o un uso de un campo clave.

Tabla	Resultado Obtenido					*Nro popups*
	Alta	Media	Baja	Tot. Err	Ok	
Tabla A	**14**	**2**	**5**	**21**	**3**	***9***
Defs	8		1	9	1	
Usos	6		3	9	1	
Clave		2	1	3	1	
Tabla B		**2**		**2**	**1**	***1***
Defs		1		1	1	
Usos		1		1		
Clave						
Tabla C	**3**	**2**	**4**	**9**		***3***
Defs	2	1		3		
Usos	1		2	3		
Clave		1	2	3		
Tabla D	**7**	**3**	**5**	**15**	**1**	***7***
Defs	2	1	4	7		
Usos	5	1	1	7		
Clave		1		1	1	
Tabla E	**1**			**1**	**1**	
Defs						
Usos						
Clave	1			1	1	
	25	**9**	**14**	**48**	**6**	**20**

Tabla 11. Resultados obtenidos en el caso de estudio

A partir de estos resultados es posible hacer varios comentarios:

1. La cantidad de queries con problemas es muy alta. El 88.46% de las queries detectan errores.
2. En cuatro de las cinco tablas existen problemas con atributos que forman parte de la clave. Por ejemplo, existen usos de campos claves que no han sido definidos previamente.
3. En muchas de las queries con criticidad alta de error, el porcentaje de falla fue de 100%. Esto quiere decir que hay campos que directamente no han sido cargados nunca a pesar de que tienen un popup asociado para facilitar dicha carga.
4. El 52.08% de las queries con error son de criticidad alta.

5. Resumen 2do experimento

La idea de incluir un caso de estudio en el presente trabajo surge a partir de la necesidad de validar los modelos de testing y cubrimientos definidos, así como también los criterios enunciados.

Si bien el primer experimento llevado a cabo durante la definición del modelo de cubrimiento ayudó a tales fines, creímos necesario mostrar más experiencias prácticas que demuestren la aplicabilidad de los conceptos enunciados. Por este motivo hemos realizado un segundo experimento: el caso de estudio presentado en este capítulo. El mismo, ha sido realizado en circunstancias diferentes al primero y por esto permitió evaluar a las técnicas propuestas en forma complementaria.

En esta sección presentaremos algunas conclusiones importantes surgidas a partir de dicho análisis:

- En primer lugar es justo decir que el modelo de testing definido fue utilizado en ambos experimentos sin inconvenientes. El esquema de testing de datos con sus definiciones paralelas de caso, query, objeto de test, resultado esperado y obtenido, ha sido respetado en ellos sin necesidad de formular correcciones o adaptaciones dependientes de la experiencia particular.
- El poder expresivo del lenguaje de escritura de los casos y de las queries fue adecuado a las necesidades planteadas en ambos experimentos puesto que permitió formular los distintos casos apropiadamente. En este sentido, no fue necesario introducir notación complementaria para expresar nuevos casos (siendo este uno de los síntomas más comunes de la falta de expresividad de un lenguaje).
- El modelo de cubrimiento y los criterios asociados también resultaron útiles ya que, a partir de los casos formulados para satisfacerlos, se han encontrado diversos problemas de calidad en los datos. Esto se demuestra en los porcentajes de error encontrados en cada uno de los experimentos.
- Algunos de los problemas descubiertos podrían haber sido encontrados por medios distintos a los planteados en el presente trabajo. Sin embargo, el verdadero valor agregado del mismo es que ahora disponemos de un modelo que permite repetir el procedimiento y aprovechar la experiencia adquirida. No es casualidad que los buenos resultados se hayan repetido en ambos experimentos.
- Si bien el conjunto de casos de prueba derivado para satisfacer el criterio de todos los usos no demostró la existencia de problemas de calidad en el primer experimento, un conjunto de casos derivados para satisfacer el mismo criterio en el segundo experimento si arrojó resultados importantes y mostró la existencia de problemas de calidad importantes en el conjunto de datos. A partir de esto pueden extraerse algunas observaciones, a saber:
    1. Los criterios definidos no serán aplicables en cualquier situación y dependerán de diversos factores. Por ejemplo, el tipo de sistema que genera los datos o la tecnología utilizada.
    2. Los criterios definidos no sólo sirven para encontrar errores sino también para demostrar su ausencia. Es decir que la ausencia de errores debe ser vista como exitosa también puesto que demuestra que el nivel de calidad de los datos con respecto a los criterios planteados (nuestro conjunto de casos) ha sido adecuado. Esta es una consideración más general que puede postularse no sólo sobre el testing de datos sino sobre el testing de software. Esta actividad tiene como objetivo principal la disminución de la probabilidad de error del objeto de pruebas.
- De los resultados obtenidos en los dos experimentos no se desprenden nuevas conclusiones sobre las implicancias existentes entre los distintos criterios enunciados. Es decir que el caso de estudio no nos ayuda a obtener nuevas relaciones de implicancia fuera de las comentadas en el capítulo sobre *Criterios de cubrimiento*.

- Es importante aclarar los factores por los que el nivel de detalle alcanzado en uno y otro experimento varía. Con el primer experimento se busca explicar en forma detallada las técnicas y los mecanismos para encontrar los casos de prueba. Se usa el ejemplo como hilo conductor durante esta explicación y se muestran todos los productos intermedios (grafo, anotaciones, casos, queries, etc.) del proceso hasta llegar al resultado (los problemas). En el segundo experimento se persigue el objetivo de evaluar la técnica estructural sobre una aplicación tecnológicamente diferente. Desgraciadamente, existen para este caso de estudio restricciones relacionadas con la confidencialidad de la información que no hacen posible mostrar ni los productos intermedios ni los problemas encontrados.

Capítulo 9

Conclusiones generales y trabajo futuro

1. Introducción

En este capítulo se presentarán, en primera instancia, los resultados obtenidos en el trabajo. Luego, se presentarán las conclusiones más importantes. Y, por último, se describirán brevemente algunas posibilidades de investigación futura.

2. Resultados obtenidos

A partir del presente trabajo se obtienen dos resultados. El primero, y principal, es un modelo teórico de testing para datos. Dicho modelo, permite practicar las actividades de testing, que hasta ahora eran propiedad exclusiva del software, en un objeto diferente: los datos.

El marco de referencia teórico, constituido por el modelo, resuelve dos problemas importantes en el área de Data Quality:

1. Brinda un marco de referencia para las diversas técnicas de data quality ya existentes. En este sentido, ya no son técnicas aisladas sino que forman parte de la actividad del testing de datos y pueden ser analizadas desde el punto de vista de la ingeniería de software. Por ejemplo, la técnica presentada en [BOBR99] basada en GQM puede ser vista ahora como una técnica alternativa de derivación de casos de prueba funcionales cuyo objetivo final es obtener un conjunto de métricas (casos) con el cual medir la calidad de los datos.
2. Establece las ideas fundamentales para definir nuevas técnicas de aseguramiento de la calidad en los datos. Un ejemplo de esto lo constituyen las técnicas presentadas aquí: a partir de dimensiones clásicas de la calidad es posible establecer casos de test que evalúen el nivel de calidad de la información.

Es decir que, a partir de su aparición, este modelo se constituye en punto de partida natural para cualquier intento de definición de técnica, método o herramienta relacionada con la calidad de la información tanto desde el punto de vista de la ingeniería del software como del de los sistemas de información.

El segundo resultado, originado a partir del primero, está constituido por dos técnicas de derivación de casos de test. Como ya se ha mencionado, estas técnicas posibilitan un análisis de la calidad de los datos en forma sistemática y ordenada puesto que proveen la entrada esencial para el modelo de testing: los casos de test. A partir de estos casos, será posible derivar queries con las que podrá llevarse a cabo la ejecución de los tests. Y a

su vez, dichos test servirán para evaluar la calidad de la información sobre la cual se hace testing.

3. Conclusiones

Si bien en la actualidad existen técnicas de verificación de la calidad sobre los datos, siempre desde el punto de vista de la ingeniería del software, éstas están dirigidas al modelo que los soporta y no a los datos en particular (a las instancias del modelo).

El presente trabajo constituye un avance en este sentido, puesto que tanto el modelo propuesto como las técnicas definidas presuponen no sólo la verificación de la estructura de los datos, sino también de las instancias existentes en ella. No sólo es importante ver que el modelo definido para los datos es correcto desde el punto de vista del software que se está construyendo (por ejemplo que está normalizado razonablemente), sino también es necesario ver que los datos ya cargados (o por cargarse) dentro de él, cumplen con los criterios de calidad fijados (esta calidad está pensada tanto desde el software como desde la organización).

En este sentido, se ve claramente la necesidad de técnicas dinámicas, además de estáticas, para la verificación de la calidad[26]. El presente modelo permite la definición de dichas técnicas a imagen y semejanza de las existentes en software. La completitud de las entidades entendida desde la presencia o ausencia de valores en cada atributo, la consistencia de los datos entendida a partir de los distintos chequeos cruzados que puedan hacerse (ya sea conociendo o ignorando el negocio particular) son algunos de los puntos sobre los cuales es posible trabajar a partir de él.

Así, el problema de establecer el nivel de calidad de ciertos datos se ha convertido en un problema de testing sobre un nuevo objeto de pruebas. Los pasos a seguir por consiguiente, son conocidos puesto que han sido probados con éxito durante más de 20 años[27]. Evidentemente, la mayoría de las técnicas de derivación de casos de prueba existentes para software no pueden ser aplicadas al nuevo objeto, pero hemos mostrado que es posible definir nuevas.

El trabajo es principalmente teórico y presenta un modelo para realizar testing sobre datos establecido a partir del modelo ya conocido de testing de programas. Constituye, como dijimos, un primer paso para el desarrollo de las técnicas de testing referidas al análisis de la calidad de la información existente en los sistemas de software, un framework para la investigación pues a partir de él es posible estudiar y definir en forma más ordenada dichas técnicas.

Obviamente, por ser en cierto modo "inicial", se espera que en el futuro el modelo evolucione de forma de cubrir las necesidades de los investigadores que lo utilicen, así como ha cubierto las nuestras. Como resultado de este trabajo entonces, nos encontramos con los fundamentos esenciales para seguir trabajando y construyendo.

---

[26] Ocurre aquí un fenómenos similar al ocurrido en el testing de software en donde la verificación de la calidad comenzó con la simple inspección o revisión de programas y luego derivó en el testing.
[27] The art of software testing fue publicado en 1979 [MYER79].

El modelo propuesto ha sido validado exitosamente mediante la definición y el uso de dos técnicas simples de derivación de casos de prueba. Estas dos técnicas, una de tipo estructural y otra de funcional, posibilitaron obtener casos de prueba para evaluar la calidad de los datos. Los mismos fueron utilizados en un experimento y un caso de estudio.

El éxito del modelo propuesto se concluye por el hecho de que éste no sufrió cambios a partir de ninguna de las experiencias. Los conceptos existentes en el testing de programas han sido dualizados sin problemas, es decir que se han mapeado en forma natural y razonable. A su vez, los lenguajes seleccionados para expresar casos de prueba y queries resultan conocidos, adecuados y de elección bastante intuitiva. Los conceptos de caso y test han sido utilizados sin cambios significativos y los diferentes niveles y tipos de testing han sido caracterizados en forma similar al testing de programas.

Como dijimos, este framework, permite atacar el mayor problema dentro de la actividad de testing, a saber, la definición de casos de prueba. Este objetivo se encaró desde dos perspectivas distintas: la estructural y la funcional.

A partir de la estructura de los datos fue posible enunciar un modelo y criterios de cubrimiento para asegurar cierto nivel de testing. El mismo fue validado, como se dijo antes, con dos experimentos que arrojaron buenos resultados. Los casos derivados para satisfacer los distintos criterios mostraron la presencia de problemas de calidad en la información. No obstante esto, la gravedad de los problemas dependió en gran medida del tipo de sistema que se estaba evaluando. Y, en este sentido, tanto la diferencia de tecnología utilizada, como el grado de importancia otorgado a los datos durante la carga en cada producto fueron factores determinantes para los resultados en cada ocasión.

A partir de los experimentos fue posible también demostrar la no implicancia de algunos de los criterios de cubrimiento enunciados y verificar la correctitud de las implicancias postuladas. En este sentido no han podido lograrse las implicaciones existentes en software, pero se tiene una visión más clara de las posibilidades que presentan los datos en cuanto a cubrimientos definidos sobre grafos.

Hemos visto también en base a los experimentos que la definición misma del resultado esperado varia significativamente según el contexto de los datos testeados. En este sentido y, en ciertas situaciones, será absurdo pretender que el 100% de los datos cumplan nuestros criterios de calidad y deberemos conformarnos con porcentajes menores. Esto mismo puede entenderse también desde el siguiente punto de vista. En ciertas situaciones, las técnicas definidas nos mostrarán candidatos a problemas sobre los cuales profundizar el análisis y no problemas a resolver. Aún aquí el trabajo es valioso pues permite reducir el espacio de búsqueda en forma significativa.

Por el otro lado, a partir del concepto de dimensión de calidad se definió un método funcional para derivar casos de pruebas que mostró dos aspectos interesantes ya conocidos en software: la necesidad de conocer el dominio del problema (con sus reglas de negocio) para derivar casos funcionales y la conveniencia de los métodos funcionales sobre los estructurales con la consiguiente necesidad de profundizar la búsqueda de este tipo de métodos. Los resultados obtenidos con los casos funcionales derivados son prometedores y el presente trabajo constituye sólo el inicio del trabajo sobre este punto.

Las conclusiones enunciadas permiten replantear el problema de la calidad en los datos, por lo menos desde el punto de vista de la ingeniería de software. Dicha calidad ahora puede ser atacada a partir del concepto de testing de datos el cual postula que la actividad de testing, tal y como la conocemos para software, puede ser utilizada también sobre los datos (los mismos conceptos básicos, un modelo similar, etc.). Ya no es necesario pensar "desde cero" la forma de tratar a la calidad de los datos, sino que es posible utilizar un modelo conocido, una forma de trabajo conocida y, lo más importante, un proceso conocido para atacar el problema.

El proceso de testing de software con sus etapas permanece intacto en su paso a los datos, la misma metodología de trabajo cuya efectividad ha sido probada con éxito en el pasado para el software, ahora sirve para los datos. No es necesario agregar al proceso de desarrollo de software nuevas etapas relacionadas con los datos, sólo es necesario replantearse el objetivo del testing en dicho proceso. Ahora, las actividades de testing deben estar dirigidas al software y a los datos, pero por suerte el modelo de testing necesario para cada uno de ellos es el mismo.

El objetivo principal del testing de datos enunciado ya desde la introducción puede ser alcanzado a partir del modelo propuesto. Testear el nivel de calidad de cierto conjunto de datos significa utilizar técnicas generales, y aplicables a distintas situaciones, que nos permitan encontrar los problemas existentes de manera sistemática. Disponer de un modelo nos permite encontrar dichas técnicas.

4. Trabajo Futuro

Como dijimos en la sección anterior este es un trabajo "inicial" y teórico. La presencia de un modelo constituye un avance importante, sin embargo, el camino por recorrer es largo y hay muchos interrogantes por responder. A continuación se presentan algunos puntos de interés para el trabajo futuro.

Sobre el modelo

Al momento de definir el poder expresivo de los lenguajes para escribir casos de prueba y queries se tomaron ciertas decisiones de modo de facilitar el trabajo de derivación y escritura de los mismos. No obstante esto, una continuación natural surge a partir de la existencia de otros niveles de expresividad, los cuales podrían otorgar beneficios extra a los casos derivados. Por ejemplo, es posible utilizar un único lenguaje para expresar ambas cosas?: propiedades y queries.

Sobre las técnicas de derivación

En el presente trabajo, y para validar el modelo propuesto, se definieron 2 técnicas de verificación de la calidad. En este sentido, es posible continuar la investigación del tema de modo de: encontrar nuevas técnicas de derivación (por ejemplo, a partir de técnicas de testing de programas) y encuadrar los métodos de análisis de data quality existentes en el modelo propuesto.

Con respecto a las técnicas presentadas, existen varias posibilidades de extensión. Para el modelo de cubrimiento estructural, es posible avanzar en dos sentidos principalmente.

En primer lugar es posible ampliar la cantidad de criterios de cubrimiento disponibles. Esto puede hacerse por ejemplo, a partir de la paralelización con el MT *(todos los caminos, Todos los DU paths)*. En este sentido, siempre es conveniente analizar las implicancias existentes entre ellos.

En segundo lugar, a partir de la definición de nuevos tipos de anotaciones es posible, no sólo enriquecer los criterios ya existentes, sino también definir nuevos. Algunos ejemplos de estas anotaciones son:

- Los usos y definiciones compuestas, es decir anotaciones que involucran no sólo a un atributo. Este tipo de anotaciones puede ser de especial importancia al tratar de trabajar con atributos complicados como por ejemplo *nombre* y *apellido* en una entidad *Cliente*.
- Las redefiniciones, es decir la anotación de definiciones que no se dan en nodos iniciales del grafo. Por ejemplo el atributo gravedad en el primer experimento presentado aquí.
- Los predicados, es decir anotaciones formadas por un par atributo nodo y acompañadas por un predicado. Estas anotaciones tienen especial importancia al trabajar en casos definidos a partir de criterios funcionales en los cuales importa marcar una regla del negocio existente entre atributos.

Sobre el testing de datos y el proceso de desarrollo de software

En el proceso de desarrollo de software, los datos generalmente tienen un papel secundario. A partir del surgimiento de un modelo para el testing de datos y de técnicas de derivación de casos, resulta conveniente investigar la relación existente entre las etapas del desarrollo de software y el testing de datos. En este sentido, es posible plantearse diversos interrogantes. Por ejemplo, en el análisis de requerimientos. Debería incorporar información adicional en los requerimientos del sistema? Las herramientas de especificación utilizadas, permiten la derivación de casos de prueba para datos razonablemente? Es necesario extender la notación de los diagramas existentes (por ejemplo los diagramas de entidad relación).

Sobre herramientas y automatización del testing

El modelo de cubrimiento propuesto en el presente trabajo tiene una característica fundamental a su favor (al igual que su par en el testing de programas): la mayoría de los pasos a realizar durante la derivación de casos pueden ser automatizados. Una continuación importante entonces es la generación de herramientas que permitan realizar las tareas con menos costo humano. La existencia de las mismas otorgaría un papel más fundamental a este tipo de técnicas, que si bien no son tan efectivas como las funcionales, podrían ser considerablemente más baratas de implementar.

5. Agradecimientos

A Daniel Yankelevich, por compartir conmigo sus conocimientos y su capacidad para la investigación. Además de ser mi director, ha sido un ejemplo durante todos estos años (académica y profesionalmente). A Mónica Bobrowski y a Marcelo Campo por haber enriquecido mi trabajo con sus comentarios y observaciones. A Martina Marré y a Carlos Blanco por su paciencia y predisposición para charlar algunos temas. A mis compañeros y amigos, quienes han tenido que padecerme durante estos años de facultad. Muy especialmente a Dan, Daniel, Darío, Diana, Diego, Martín, Rodolfo y Sabrina con quienes he vivido "momento extremos" en esas épocas en las que teníamos que entregar nuestros trabajos de grupo. A esos profesores que me han marcado el rumbo (aún teniendo que repetirme, no puedo dejar de nombrarlos a todos juntos): Daniel, Martina, Miguel, Mónica, Nelson, Santiago y Victor. Además de buenos docentes, han resultado ser buenos amigos. A mis viejos. Y a Laura, por haber estado siempre que la necesité.

Bibliografía

ABIT95 Abiteboul S., Hull R., Vianu V. *Fundations of Databases*. Addison-Wesley, 1995.

BALZ98 Balzamo, Martín. *Testing sobre bases de datos activas, tesis de licenciatura.* Depto. computación FCEyN Universidad de Buenos Aires, 1998.

BEIZ90 Beizer, Boris. *Software Testing Techniques*. Van Nostrand Reinhold, 1990 (2nd ed.).

BEIZ95 Beizer, Boris. *Black-Box Testing, Techniques for funcional Testing of Software and Systems*. John Wiley & Sons, 1995.

BOBR98 Bobrowski, M, Marré, M. Yankelevich, D. *A Software Engineering View of Data Quality*. Proceedings of 2nd european Quality Week, November 1998 Brussels, Belgium.

BOBR99 Bobrowski, M, Marré, M. Yankelevich, D. *An Homogeneous Framework to Measure Data Quality*, 1999.

BROO87 Brooks, F. *No Silver Bullet - Essence and Accidents of Software Engineering".* IEEE Computer, Abril de 1987.

FENT97 Fenton, N.E., Pfleeger, S.L. *Software Metrics – A rigorous & Practical Approach*. ITP Press, 1997 (2nd ed.).

JACO93 Jacobson, Ivar. *Object Oriented Software Engineering, A Use Case Driven Approach*. Addison Wesley, 1993 (4ta edición).

MEND89 Mendelzon, A.O. *Inductive pebble games and the expressive power of datalog.* Dept. of Computer Science, University of Toronto, Canada, 1989.

MORI99 Morillaz, S. *Una metodología para medir calidad de datos, tesis de licenciatura*. Depto. de computación, FCEyN, Universidad de Buenos Aires, Argentina, 1999.

MYER79 Myers, G.J. *The art of software testing*. John Wiley, 1979.

NAVA94 Elsmasri, R y Navathe S. *Fundamentals of Database Systems*. Addison Wesley, 1994 (2nd ed.)

REDM96 Redman, T. *Data Quality for the information Age*. Artech House, 1996.

ULLM89 Ullman, J.D. *Principles of Database and Knowledge-Base systems*. Computer Science Press, New York, 1989.

WANG99 Wang, R., Lee Y. Y Huang K. *Quality Information and Knowledge.* Prentice Hall, 1999.

WANG98 Wang, R. *A Product perspective on Total Data Quality Management.* ACM Communications. Vol 41. Nro. 2, February 1998.

WEYU82 Rapps, S., Weyuker, E.. *Data Flow Analisys Techniques for the Test Data Selection,* ICSE, 1982.

WEYU88 Frankl, P., Weyuker, E.. *An Applicable Family of Data Flow Testing Criteria*. IEEE Transactions on Software Engineering. Vol 14 No 10, October 1988.

Referencias

1. Capítulos

2. Ilustraciones

3. Tablas

4. Ejemplos