**LIBRO BLANCO** | SEPTIEMBRE DE 2015

# Una crítica de las pruebas

ca technologies

# Índice

**Sección 1**

# Peras y manzanas

## Necesidad de objetividad

Este trabajo es fruto de la necesidad de contar con un análisis comparativo de las distintas metodologías de diseño de casos de prueba. Durante la compilación del análisis, no tardé en descubrir que no existen criterios objetivos y cuantitativos que se puedan aplicar a todas las metodologías de diseño de casos de prueba que se emplean. Cierto es que hay multitud de tratamientos subjetivos y cualitativos. La mayoría sirven como herramientas de marketing para uno u otro distribuidor. El problema de estos tratamientos es que resulta imposible comprobar sus méritos relativos. Todo el trabajo tiene como base anécdotas y datos, en lugar de asentarse en demostraciones según unos principios previamente definidos. Es decir, nos enfrentamos a comparaciones hechas a posteriori. En términos de evolución de sistemas, un análisis a posteriori solamente nos permite avanzar un trecho, pero después es preciso demostrar el avance a priori.

## Oportunidad

Con el fin de hallar criterios a priori, se reveló que era necesario dejar atrás todo rastro de subjetividad y basarse en un marco de trabajo fundamental como cimientos para trabajar, que nos permitiese efectuar un análisis objetivo. Este paso supuso una ruptura radical tanto en el plano de la filosofía como en el de las matemáticas: al crear un marco de trabajo basado en principios definidos previamente (lo que se denomina un marco de trabajo axiomático), ambas disciplinas podían avanzar a un ritmo prodigioso, puesto que ya no era necesario realizar comprobaciones a posteriori por medio de los datos. Así, los responsables de pruebas se encuentran en una posición óptima para percibir qué disparate suponía la verificación a posteriori, que hoy constituye el método de trabajo aceptado habitualmente. En lugar de poner a prueba las ideas desde su propia concepción, se considera aceptable realizar pruebas durante y después de la implementación. Algo similar a comprobar la fortaleza de un edificio sin verificar antes que sus cimientos son sólidos.

## Finalidad

La finalidad de este artículo es establecer un marco de trabajo objetivo, que se dividirá en tres conceptos fundamentales relacionados entre sí:

1. La información como lenguaje universal, dividida en las categorías mensurable y no mensurable. Todas las etapas del ciclo de vida del desarrollo de software (SDLC) se consideran como información de diferentes formas.
2. La incertidumbre, modelada como entropía informativa: En esta categoría caben los defectos inobservables y el software que no es posible someter a pruebas.
3. Transformaciones de la información: desde la idea hasta el diseño y desde el diseño hasta la implementación. Dado que se considera que el SDLC consta de diferentes formas de información, las etapas intermedias se modelan como transformaciones de la información. Para gran parte de este tratamiento, las contemplaremos como máquinas de Turing.

Durante la compilación del análisis, se descubrió que no existen criterios objetivos y cuantitativos que se puedan aplicar a todas las metodologías de diseño de casos de prueba que se emplean.

La mayoría de estas ideas derivan del ámbito de la mecánica cuántica, que podríamos concebir sencillamente como la creación de modelos de información a un nivel extremadamente granular. A través de este trabajo, emplearemos con profusión el concepto de “escala de observación”, por un hecho muy sencillo: si observamos un mismo sistema desde diferentes escalas, siempre sigue siendo un sistema, en realidad solo cambian los detalles específicos. Por ejemplo, con frecuencia los analistas empresariales definen requisitos para un sistema a diferentes escalas: de alto nivel, seguido por un nivel alto inferior, al tiempo que proporcionan detalles de implementación muy precisos para el nivel más bajo. Otro ejemplo más: entre los desarrolladores, es habitual implementar un sistema de gran tamaño en forma de colección de sistemas más pequeños, dotados de conectores entre sí para garantizar que funcionen como una unidad. Puesto que tanto los requisitos como las implementaciones son información y dado que ambos elementos se pueden contemplar desde escalas distintas, tiene sentido que introduzcamos la escala como ingrediente para nuestro tratamiento de la información.

**Sección 2**

# Los principios de las pruebas

Tras dejar establecido que la objetividad es imprescindible, el siguiente paso consiste en definir los axiomas necesarios para nuestro análisis objetivo. Como hemos Como hemos visto anteriormente, los axiomas se asientan sobre el concepto de información según la mecánica cuántica. La información, como concepto abstracto, debería concebirse como un conjunto de instrucciones que describen algo o permiten actuar. La información descriptiva debería ser directa y fácil de comprender: dado un objeto u entidad, la información es el medio que nos permite describirlo. Por ejemplo, supongamos un objeto; utilizaríamos adjetivos para describir sus características estéticas y adverbios para explicar cómo actúa dentro del mundo. Por otro lado, la información activa está compuesta por instrucciones que explican cómo construir un objeto determinado. Por ejemplo, las instrucciones de montaje de un mueble se consideran información activa.

## La información se transforma

La distinción entre los dos categorías está cargada de sutileza, pero es muy importante para nuestro tratamiento, ya que consideraremos solamente la información activa. Así, los documentos de diseño y requisitos se conciben como instrucciones para montar un sistema o un elemento de software. Dado que los sistemas y el software se pueden abstraer y considerar por sí mismos como un conjunto de instrucciones para manipular datos, es obligatorio que los consideremos información, como los datos. Este es uno de los conceptos absolutamente esenciales que utilizaremos, así que es importantísimo comprenderlo al detalle. De aquí surge el segundo de nuestro conceptos más destacados: la información se transforma. En palabras sencillas, se trata de transformaciones que reciben la información de una forma y la emiten de otra forma distinta. El ejemplo más fácil de entender es el desarrollador: un desarrollador transforma la información que recibe en forma de requisito o diseño; la convierte en un elemento de software o un diseño.

Se revela así que el SDLC al completo se puede contemplar como una cadena de diferentes formas de información, unidas entre sí por transformaciones. Todo se puede considerar como información de alguna forma, desde los requisitos hasta el producto acabado.

### La mensurabilidad y la incertidumbre aparejadas a la información

Sin embargo, hasta el momento no hemos afrontado el concepto de calidad. Para ello estamos obligados a invocar otros dos conceptos relacionados: la mensurabilidad y la incertidumbre asociadas a la información. Dicho llanamente, la mensurabilidad de la información es una medida que contabiliza cuánta información es posible observar realmente. El hecho de que la información exista es completamente independiente de nuestra capacidad para capturarla. La información no mensurable es, en esencial, inútil para nuestras necesidades, aunque es importante que tengamos al menos cierta idea de cuál es en realidad el volumen existente de información no mensurable. En otro apartado abordaremos la metainformación (o información de segundo orden). Por el momento, nos centraremos en la información de primer orden. La incertidumbre es un concepto asociado a la mensurabilidad de la información: vamos a denominarlo como “entropía de la información”. Tomaremos su definición prestada directamente de la mecánica cuántica, un mecanismo que produce información diferente como resultado. Para ayudarnos a conceptualizar la distinción entre información de primer y segundo orden, pensemos en la diferencia entre datos y metadatos: los metadatos existen para describir propiedades de los datos. Ese es un ejemplo de información de primer y segundo orden respectivamente. Nos ocuparemos de definir correctamente la metainformación en el apartado “Metamatemáticas y metadesarrollo”.

Esta es, en efecto, una generalización de las ambigüedades en los requisitos: las ambigüedades desembocan en incertidumbres dentro de las instrucciones y las múltiples interpretaciones posibles permiten llegar a varios resultados distintos (o implementaciones de esos requisitos). Si el responsable de las pruebas y el desarrollador eligen dos interpretaciones diferentes del mismo requisito, es casi seguro que los casos de prueba no reflejarán la implementación.

Hay un último grupo de definiciones necesario antes de enunciar los principios; debemos distinguir entre dos tipos de entropía:

- Epistémica: información oculta como omisiones, grados perdidos de libertad debidos a que la información, sencillamente, no está presente. En nuestro mundo, el resultado es que la información, simplemente, debe "inventarse" durante el proceso de transformación. Por ejemplo, a veces sucede que un desarrollador tiene que rellenar huecos cuando los requisitos son incompletos o demasiado vagos. En principio, se trata de una cantidad mensurable. En la práctica, exige un volumen de trabajo considerable: incluso cuando los desarrolladores adoptan asunciones, siempre se manifiesta en el código resultante. Este es un ejemplo de lo que significa que una transformación sea reversible.
- Sistémica: la entropía inherente al sistema, que no es posible medir. Es comparable al principio de incertidumbre de Heisenberg dentro de la mecánica cuántica. En nuestro mundo, se debe al hecho de que los conjuntos de instrucciones o sistemas de axiomas nunca pueden ser coherentes y completos simultáneamente. Como tal, esta no es una cantidad mensurable, pero si aplicamos la noción de tamaño de conjunto relativo, sí es posible al menos cuantificar exactamente cuánta entropía sistémica existe, ya que el conjunto formado por los conjuntos no mensurables sí se puede medir, algo que ciertamente atenta contra la intuición.

Las bases fundamentales de las pruebas son seis principios que son consecuencias directas del marco de trabajo teórico de la información que hemos establecido. Se pueden resumir así:

1. No es posible poner todo a prueba. Dentro de un sistema dado siempre existe una entropía sistémica, consecuencia directa del teorema de la incompletitud de los sistemas axiomáticos de Gödel (o del teorema de la indecibilidad de Turing).
2. Siempre es posible saber qué se puede poner a prueba. Dada una cantidad de información suficiente, es posible anular la entropía epistémica y también es posible medirla.
3. Los defectos observables representan la entropía mensurable de un plan de pruebas. Puesto que los defectos (o la carencia de ellos) son los indicadores de calidad de un fragmento de software, es razonable tratarlos como incertidumbres. Concretamente, los mejores planes de pruebas son los que convierte en observable el mayor número de defectos. Para vincular estos dos indicadores y fundirlos en una medición, una práctica habitual consiste en asignar un nivel de gravedad a los defectos. Entonces, la medición se convierte en un promedio compensado, aunque es preciso actuar con precaución, ya que las compensaciones arbitrarias introducen cierto grado de subjetividad, que conviene evitar.
4. La cobertura es la medida de la fiabilidad de la estrategia o plan de pruebas. Dicho de otro modo, indica si la estrategia o el plan de pruebas refleja con precisión los requisitos, ya que eso revelará todos los defectos observables posibles y nos permitirá percibirlos. Una estrategia o un plan de pruebas con deficiencias provocará que sea imposible observar muchos defectos que de otra manera sí serían observables.
5. La entropía nunca se puede perder. Esto deriva directamente de la mecánica cuántica. En nuestro ámbito, implica que si alguna de las etapas del SDLC aporta entropía, nunca será posible eliminarla, aunque todas las demás etapas subsiguientes sean fiables al 100 %. Concretamente, la calidad de un sistema de software es directamente proporcional a la calidad de los requisitos y la estrategia de pruebas. Tenga en cuenta que esto no implica que jamás será posible obtener un software que funcione: en palabras sencillas, significa que nunca será posible lograr un software perfecto.
6. No existe ningún modelo que pueda detectar todos los defectos. Como consecuencia del primer principio, no es posible someter a pruebas todos los elementos, pero el uso de varios modelos conlleva que podamos hacer observables diferentes conjuntos de defectos. Sin embargo, revelar todos los defectos requiere, como mínimo, un número infinito de modelos. Esto es consecuencia directa de los teoremas de incompletitud de Gödel.

**Sección 3**

# Metamatemáticas y metadesarrollo

Antes, al profundizar en la información, nos hemos ceñido exclusivamente a la información de primer orden, es decir: información pura. No obstante, también es posible contar con "información sobre la información", que denominaremos metainformación o información de segundo orden. Un buen ejemplo de esto son las estadísticas de agregación: dada una población, se puede deducir un gran volumen de información del promedio de altura (y de la desviación estándar), aunque desconozcamos la altura de cada individuo.

Para comprender la utilidad de la metainformación, categorizamos la información según los términos de lo que podríamos denominar matriz de Rumsfeld (bautizada en honor del Secretario de defensa de EE. UU.), que contiene los siguientes elementos:

- Elementos conocidos que conocemos
- Elementos conocidos que desconocemos
- Elementos desconocidos que conocemos
- Elementos desconocidos que desconocemos

Se puede representar como una rejilla. En ella hemos introducido todos los conceptos que hemos abordado hasta el momento:

| | Elementos conocidos | Elementos desconocidos |
|---|---|---|
| Que conocemos | Información | Entropía epistémica<br>Defectos observables |
| Que desconocemos | Entropía epistémica<br>Defectos observables | Entropía sistémica |

La metainformación es muy útil para, como mínimo, calibrar el alcance de lo que desconocemos, aunque carezcamos de la información. En esencia, este es el fin para el que se desarrolló la disciplina de las metamatemáticas: establecer qué se puede demostrar y qué no se puede demostrar, por medio del análisis de los marcos de trabajo matemáticos. Aunque resulta imposible demostrar una gran proporción de las proposiciones matemáticas, como la hipótesis del continuo relacionada con la cardinalidad del conjunto de los números reales, esto no causa problemas para la comunidad matemática, ya que sí es posible centrarse en las proposiciones que sí son demostrables. Este mismo principio se debería aplicar a la realización de pruebas: utilizando métodos análogos, es posible concentrarse en los factores que podemos someter a pruebas (es decir, los elementos conocidos que conocemos) y conformarnos con la metainformación del resto de factores. Los elementos desconocidos que desconocemos (es decir, la entropía sistémica) suponen un problema, ya que no se pueden medir. Pero las otras dos secciones (los elementos desconocidos que conocemos y los elementos conocidos que desconocemos) sí podemos, al menos medirlos. Por tanto, es posible derivar estadísticas, algo tremendamente útil para el análisis de riesgos.

La idea central de esta analogía, que yo denomino provisionalmente "metadesarrollo" es que deberíamos centrarnos en el software que puede someterse a pruebas (o sea, los elementos conocidos que conocemos) y dirigir todos los esfuerzos para potenciar al máximo la información disponible y observable. Para lo demás, la metainformación es suficiente. Por desgracia, las metodologías actuales de diseño de casos de prueba no explotan convenientemente la información disponible, como veremos en nuestra crítica. Si bien es cierto que "la mayoría" de los algoritmos, hablando en términos de máquinas de Turing, no se pueden poner a prueba (como demostró el problema de parada o problema de Halting), todavía queda un número infinito de configuraciones que sí es posible someter a pruebas, al menos hasta conseguir una cobertura 100 % funcional. En este contexto, yo aplico el concepto del tamaño relativo (tomado de la teoría de las mediciones) para comparar conjuntos infinitos: el conjunto de algoritmos que no se pueden someter a pruebas conforma la mayoría del número total de algoritmos (es decir, una cantidad infinita e incontable), mientras que el conjunto de algoritmos que sí se pueden poner a prueba conforma un subconjunto desdeñable (un número infinito contable, que en la teoría de las mediciones se considera como un "conjunto de medida 0" o "conjunto nulo", los cuales a efectos prácticos son de un tamaño infinitesimalmente pequeño por comparación).

**Sección 4**

## Una crítica de los métodos de diseño de casos de prueba

Una vez establecidos los principios fundamentales enumerados anteriormente, ahora avanzamos para ofrecer una crítica objetiva de los métodos de diseño de casos de prueba. Teniendo en cuenta el primero de ellos, será imposible detectar absolutamente todos los defectos. Sin embargo, no podemos olvidar el segundo, el tercero y el cuarto, que nos proporcionan criterios para evaluar los métodos de diseño de casos de prueba:

1. ¿Cuánta información de aplicaciones es posible codificar en el proceso de diseño de casos de prueba?
2. ¿Cuántos defectos se convierten en observables?
3. Cobertura: ¿cuántos defectos pasan a ser observables por medio de este método, expresados como proporción del numero máximo teórico de defectos observables?
4. Número relativo de casos de prueba requerido para alcanzar un nivel de cobertura óptimo.

Basándonos en estos cuatro criterios, asignaremos a cada método de diseño de casos de prueba una puntuación (del 1 al 10, siendo 10 la mejor posible), que tomará en consideración los siguientes elementos:

- **Capacidad de codificación**: Se trata de la cantidad de información cuantitativa sobre la aplicación que es posible codificar e integrar en el método. (Según el enunciado 1, mencionado anteriormente).
- **Facilidad para la codificación**: ¿Cómo de fácil resulta codificar la información?
- **Aplicabilidad**: ¿Cuántos escenarios se pueden codificar sirviéndose de este método y actuando con sensatez?
- **Número de casos de prueba**: Número relativo de casos de prueba generados (1 – demasiados o insuficientes, 10 – óptimo). (Según el enunciado número 4, mencionado anteriormente).
- **Defectos detectables**: Número relativo de defectos que se pueden encontrar. (Según el enunciado 2, mencionado anteriormente).
- **Cobertura**: Cobertura funcional relativa que es posible conseguir. (Según el enunciado 3, mencionado anteriormente).

Fíjese en que todas las puntuaciones oscilan entre 1 y 10, siendo esta última la mejor posible.

---

**Sección 5**

## Defectos observables en la lógica

Antes de continuar y profundizar, es preciso definir algunas propiedades que caracterizan a los defectos observables en relación con los enunciados lógicos. La mayoría de los modelos formales de pruebas que usan información de aplicaciones dependen, de una forma fundamental, del análisis de enunciados lógicos. Por tanto, ese es un buen lugar donde comenzar la búsqueda de defectos.

Analicemos un enunciado muy sencillo: **SI A Y B, LUEGO C**

Se puede dividir en causas (A y B) y efectos (C). En términos de defectos, las dos causas pueden presentar tres estados: implementada correctamente (OK), atascada en el nivel 0 (0) y atascada en nivel 1 (1). Ahora bien, ¿qué sucede cuando consideramos los efectos asociados a C, a tenor de esta información?

Todos los posibles efectos asociados a C están codificados en la tabla que figura a continuación (todos los defectos se resaltan en rojo):

| A | B | C | 1 | 1 | OK | OK | 1 | 1 | 0 | 0 | A |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | OK | OK | 0 | 1 | 1 | 0 | 1 | 0 | B |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | **0** | |
| 0 | 1 | 1 | **0** | 1 | 1 | 1 | 1 | 1 | 1 | **0** | |
| 1 | 0 | 1 | 1 | 1 | **0** | 1 | 1 | 1 | 1 | **0** | |
| 1 | 1 | 0 | 0 | **1** | 0 | **1** | **1** | **1** | **1** | **1** | |

Como puede ver, las últimas tres variaciones funcionales (es decir, conjuntos de entradas verdadero/falso) son suficientes para descubrir todos los posibles defectos. La primera variación, sencillamente, no detecta más defectos. Así pues, es posible demostrar que todos los posibles efectos se pueden descubrir con esta metodología. Esta tabla indica cifras generales para un operador lógico con n entradas:

| Número de entradas | Número de variaciones funcionales necesarias | Número de combinaciones posibles | Número de defectos posibles |
|---|---|---|---|
| 2 | 3 | 4 | 8 |
| 3 | 4 | 8 | 26 |
| 4 | 5 | 16 | 80 |
| 5 | 6 | 32 | 242 |
| 6 | 7 | 64 | 728 |
| *n* | *n+1* | $2^n$ | $\sum_{x=1}^{n} \binom{n}{x} 2^x = 3^n - 1$ |

Como hemos visto antes, existen dos clases de defectos: los que se pueden observar y los que no. El factor distintivo no es que sus efectos sean imposibles de observar, sino que sus causas raíz sean inobversables. Esto es absolutamente esencial para asegurarnos de que llegamos a la respuesta correcta mediante el motivo apropiado. En este contexto, la observabilidad se puede concebir como la información mínima necesaria para identificar, reproducir o subsanar el defecto. Dicho de otra forma: ante la aparición de un defecto, debería ser posible señalar con exactitud dónde se ha originado. En muchos aspectos, un buen diseño de los casos de prueba intenta hacer precisamente esto a la inversa: proporcionar el número mínimo de casos de prueba para revelar la mayor cantidad posible de defectos observables. No olvide que, en nuestro ámbito, los defectos observables representan la entropía epistémica del software y se pueden anular y medir, mientras que los defectos que es imposible observar son sistémicos y no pueden someterse a pruebas aplicando el mismo modelo. Por lo tanto, el nivel de cobertura de las pruebas es una medida de la cantidad de entropía epistémica que es posible anular por medio del proceso de pruebas.

**Sección 6**

## Métodos combinatorios: rápidos y sucios (todos los pares, etc.)

Derivados de sencillas propuestas de recuento, los métodos combinatorios son la variante más sencilla del diseño de casos de prueba. Dada una lista de columnas y los valores posibles para cada una, sencillamente nos limitamos a derivar todas las posibles combinaciones de pares, tríos, etc. y rellenarlas rápidamente. La característica más atractiva que tienen estos métodos es que no descansan sobre los conocimientos de ninguna aplicación; sin embargo esta virtud también constituye su mayor debilidad. Por tanto, la entrada de información cuantitativa está restringida a los datos, sin la posibilidad de especificar ninguna información sobre sus relaciones. Siguiendo el principio fundamental expuesto anteriormente, la información cualitativa sobre el sistema que revelan las pruebas de este tipo también será relativamente baja. Dado que no se trazan relaciones con los puntos funcionales reales, de ninguna forma, no es posible determinar qué volumen de la lógica de las aplicaciones se está poniendo a prueba en realidad. En la mayoría de escenarios, la cantidad de pruebas realizadas adolece de una insuficiencia endémica, si bien algunos aspectos se someten a un exceso de pruebas sin necesidad, debido a las condiciones lógicas redundantes.

Los métodos combinatorios presentan una segunda debilidad, consecuencia de la anterior: a menudo se producen combinaciones de datos inválidas, que provocan la aparición de falsos positivos, en los que los fallos de las pruebas no los causan defectos reales, sino los propios datos. Esto se puede mitigar en cierto grado si se introducen limitaciones: así queda demostrado que, si se proporcionan más entradas cuantitativas, los datos cualitativos obtenidos son mucho más claros. En tercer lugar, los resultados esperados no se pueden considerar automáticamente. Una vez más, descansan sobre entradas de información cuantitativas. Los métodos superiores permiten codificar esta clase de información y por eso nuestro análisis desvela que los métodos combinatorios son una opción muy mediocre para diseñar casos de prueba.

Por tanto, los métodos combinatorios deberían limitarse exclusivamente para realizar pruebas de configuración y no funcionales. En otras palabras, para escenarios donde el único resultado que se espere sea "funciona".

He aquí las puntuaciones (del 1 al 10) para los métodos combinatorios:

| Información de entrada | | | Información de salida | | |
|---|---|---|---|---|---|
| Capacidad | Facilidad | Aplicabilidad | N.º de pruebas | Defectos detectables | Cobertura |
| 1 | 9 | 5 | 2 | 1 | 1 |

**Sección 7**

## Hiperespecialización: métodos de cobertura multiplicativa

Este método consiste en una adaptación altamente especializada de los métodos combinatorios. Solamente es útil allí donde el escenario sometido a pruebas depende únicamente del número de instancias de una entidad de datos concreta. Por ejemplo, supongamos que tenemos una base de datos de cuentas de clientes y una pantalla debe mostrar todas las cuentas correspondientes a un cliente determinado. Los escenarios de pruebas serían estos:

1. El cliente no tiene cuentas.
2. El cliente tiene solamente una cuenta.
3. El cliente tiene varias cuentas.

Fíjese en que esta técnica asume de forma implícita cierta lógica funcional, por ello, se introduce un determinado volumen de información de las aplicaciones en el diseño de los casos de prueba, a diferencia de otros métodos más simples. De hecho, yo mismo aplico esta técnica cada vez que me enfrento a relaciones padre-hijo dentro de una base de datos relacionales o a estructuras jerárquicas (como el caso del XML). En líneas más generales, esta técnica es fantástica para poner a prueba métodos de agregación sobre estructuras de datos complejas. Esto se debe a que algunas funciones de agregación presentan (o degeneran en) casos especiales, que deben procesarse por separado. Por ejemplo:

- Para obtener un promedio de un conjunto de números se requiere una operación especial si nos enfrentamos a un conjunto vacío. De lo contrario, nos topamos con el error de que uno se divida por cero.
- Para obtener la desviación estándar de un conjunto de números se requiere una operación especial tanto para el conjunto vacío como para un conjunto que contenga un único valor. Si no se procesan por separado, el primero provocará un resultado negativo y el segundo, una división por cero.

El resultado cualitativo de este método es la confirmación de que todos los conjuntos de entradas posibles funcionan correctamente, contando únicamente con su tamaño. Este es un caso especial de pruebas de clases de equivalencia, donde el rango de entradas se agrupa en clases desarticuladas con el fin de reducir el número total de escenarios de pruebas, al tiempo que se mantiene la integridad de la cobertura funcional.

He aquí las puntuaciones (del 1 al 10) para los métodos de cobertura multiplicativa:

| Información de entrada | | | Información de salida | | |
|---|---|---|---|---|---|
| Capacidad | Facilidad | Aplicabilidad | N.º de pruebas | Defectos detectables | Cobertura |
| 3 | 7 | 1 | 5 | 3 | 3 |

**Sección 8**

## Introducción a los modelos formales

Vamos a dedicar el resto de este artículo a las técnicas de modelado formales, que proporcionan la información cualitativa más precisa sobre el sistema. Como constata el principio expuesto más arriba, esto requiere el máximo volumen de conocimientos y experiencia, dado que es preciso disponer de un gran volumen de información cuantitativa para sacarle el mayor partido posible. Si nos atenemos a la cantidad de información que se puede codificar en el proceso de diseño de casos de prueba, estos métodos son únicos.

Antes de continuar, conviene que veamos una breve descripción de lo que son los modelos formales. En esencial, se trata de descripciones de precisión matemática de un requisito, que permiten realizar operaciones ulteriores que nos aportan información cualitativa sobre dicho requisito. Estas son las tres operaciones más importantes:

1. Comprobación de la coherencia: garantiza que la lógica del requisito sea coherente internamente. Así se eliminan ambigüedades debidas a las contradicciones.
2. Comprobación de la completitud: garantiza que la lógica del requisito sea completa. Así se eliminan ambigüedades debidas a omisiones.
3. Derivación de los resultados esperados: un cuerpo lógico coherente y completo aportará los resultados esperados para cualquier posible escenario. De este modo, posibilita derivar de forma automática los resultados esperados de los casos de pruebas. Desde la perspectiva de un responsable de pruebas, se trata del punto fuerte más destacado de los modelos formales.

Los propios formalismos resultan más útiles, sencillamente en virtud de las tres operaciones enumeradas anteriormente. Todas ellas proporcionan información cualitativa con un enorme nivel de detalle sobre el sistema que se pone a prueba. De hecho, si se aplicasen estos métodos en una fase más temprana del ciclo de vida del desarrollo, la información cualitativa se podría extraer de los propios requisitos, lo que garantizaría que el proyecto echa a caminar sobre unos cimientos robustos. Es más, la mayoría de las metodologías de especificación de requisitos son inertes e inútiles fuera de la fase de diseño (más allá de la intervención manual), mientras que los modelos formales pueden proporcionar información para todas las etapas del ciclo de vida del desarrollo. Sin entrar demasiado en detalles, estos formalismos constituyen los cimientos que han servido de apoyo para alcanzar la mayoría de los avances en matemática y computación aproximadamente del último siglo. Si son suficientemente buenos para eso, cabe pensar asimismo que lo serán para nosotros.

Los modelos formales se presentan en todo tipo de tamaños y formas, así que me centraré en los dos más pertinentes para las pruebas: la creación de diagramas de flujo y la creación de gráficas de efectos. Podríamos considerarlos como las formas canónicas; todo modelo formal es en cierto grado similar a alguno de estos dos.

**Sección 9**

## Visualización de sistemas y requisitos: diagramas de flujo

La creación de diagramas de flujo, en líneas generales, es una manera de especificar los requisitos que goza de una amplia comprensión. Presenta numerosas ventajas frente a los enfoques basados en grandes bloques de texto escrito, pero la más útil de ellas es la capacidad de derivar casos de prueba de forma sistemática a partir de los requisitos (lo que pueden lograr los algoritmos automatizados). Es un método más directo de lo que parece a primera vista y también es una perspectiva que a mí me parece difícil expresar.

Para derivar los casos de prueba, sencillamente se evalúan las posibles rutas que recorren el diagrama de flujo. Eso es todo, en esencia (en lenguaje formal, se denomina análisis homotópico gráfico). Su aspecto más ingenioso se revela cuando se aplican técnicas de optimización para reducir el número de casos de prueba, pero se conserva la cobertura funcional en virtud de la estructura lógica. En esencia, solamente hay que observar los bloques de decisiones individuales: si cada bloque (u operador) se prueba a fondo abarcando todas las vías directas de entrada y salida, se considera que el flujo se ha sometido a pruebas completas. Esto es algo que resulta extremadamente difícil (y frustrante) comunicar, más allá de la típica exclamación de que "¡las cuentas salen!".

Resumidamente, diremos que las cuentas salen porque algunas rutas son redundantes debido al mismo conjunto de condiciones locales (es decir, en el nivel de los operadores) que se están analizando, lo que sucede si intentamos contemplar otras condiciones que se someten a pruebas (lejanas, es decir, globales). Se trata, sencillamente, de que las variaciones funcionales redundantes se cancelan y despejan entre sí. Es una de esas cosas que resulta más fácil explicar en general que fijándose en casos específicos (como sucede con gran parte de la matemática abstracta).

Dado que los diagramas de flujo permiten especificar requisitos libres de ambigüedades, la cantidad de información cuantitativa que este método permite codificar es inmensa y multiplica la de los dos métodos anteriores. Una vez más, esto es válido para todos los métodos de modelado formales. Las diferencias radican en el grado. Además, gracias al concepto de realización de pruebas al nivel de operadores, la cantidad de defectos que es posible detectar también es muy elevada. De ahí que la cantidad de información cualitativa obtenida a través del modelo también sea muy grande.

He aquí las puntuaciones (del 1 al 10) para el método de diagramas de flujo:

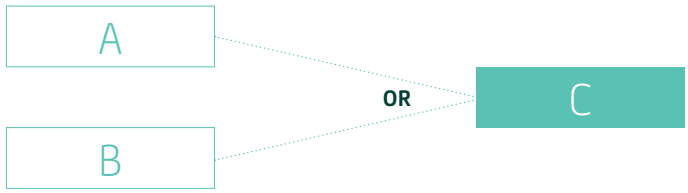
| Información de entrada | | | Información de salida | | |
|---|---|---|---|---|---|
| Capacidad | Facilidad | Aplicabilidad | N.º de pruebas | Defectos detectables | Cobertura |
| 9 | 9 | 9 | 9 | 9 | 9 |

**Sección 10**

# Especificación del contenido lógico esencial: gráficas de causa y efecto

Otro ejemplo de modelo formal; que en este caso consiste en modelar los enunciados lógicos como causas y efectos, especificando al detalle las relaciones entre estos elementos. Al igual que sucede con los diagramas de flujo, este método se ha analizado de forma específica debido a su capacidad para garantizar que los requisitos formulados mediante texto se especifiquen totalmente libres de ambigüedades.

Como con los diagramas de flujo, la idea consiste en que los bloques se enlazan, salvo que esta vez lo que se especifican son las relaciones causales. Asimismo, los bloques se ligan entre sí mediante operadores lógicos como Y, O y NO. Por ejemplo, el enunciado "si A o B, luego C" se puede codificar así:




Debe quedar claro que de esta manera es posible codificar la información de una forma extremadamente granular. En este aspecto, se trata de una de las mejores opciones disponibles. Sin embargo, también ha de quedar patente que este no es un método fácil en absoluto, por eso se considera uno de los más complicados.

No obstante, la capacidad para encontrar defectos es impresionante, mucho más que en el caso de los diagramas de flujo, dado que los defectos se pueden derivar de forma analítica a partir de una condición lógica. Si desea una explicación del proceso en profundidad, consulte la sección de "defectos observables a partir de la lógica". En particular, es posible demostrar que todos los defectos posibles se pueden descubrir por medio de esta metodología. Como resultado, esta metodología es, con mucho, la mejor si hablamos de descubrir defectos observables, aunque también es muy difícil en comparación con los diagramas de flujo. Este factor, por desgracia, ha obstaculizado su adopción. Su dificultad radica, concretamente, en intentar codificar requisitos específicos de pedidos, en oposición a los diagramas de flujo, que son ideales para esta finalidad aunque adolecen de la imposibilidad de codificar bien los requisitos independientes de pedidos.

He aquí las puntuaciones (del 1 al 10) para el método de gráficas de causa y efecto:

| Información de entrada | | | Información de salida | | |
|---|---|---|---|---|---|
| Capacidad | Facilidad | Aplicabilidad | N.º de pruebas | Defectos detectables | Cobertura |
| 10 | 5 | 8 | 10 | 10 | 10 |

**Sección 11**

# Comparación relativa

La tabla que figura a continuación contiene las puntuaciones generales de todos los métodos de diseño de casos de prueba vistos anteriormente (con puntuaciones del 1 al 10). Son las puntuaciones más objetivas posibles, juzgadas de acuerdo con los siguientes criterios debatidos:

- **Capacidad de codificación**: Se trata de la cantidad de información cuantitativa sobre la aplicación que es posible codificar e integrar en el método.
- **Facilidad para la codificación**: ¿Cómo de fácil resulta codificar la información?
- **Aplicabilidad**: ¿Cuántos escenarios se pueden codificar sirviéndose de este método y actuando con sensatez?
- **Número de casos de prueba**: Número relativo de casos de prueba generados (1 – demasiados o insuficientes, 10 – óptimo).
- **Defectos detectables**: Número de defectos relativo que se pueden encontrar.
- **Cobertura**: Cobertura funcional relativa que es posible conseguir.

NOTA: También se incluye la clase de modelos formales que se integran dentro de una categoría. Las puntuaciones se incluyen como rango, ya que algunos son más potentes, menos aplicables, etc. que otros.

| Método | Información de entrada | | | Información de salida | | |
|---|---|---|---|---|---|---|
| | Capacidad | Facilidad | Aplicabilidad | N.º de pruebas | Defectos detectables | Cobertura |
| Combinatorio | 1 | 9 | 5 | 2 | 1 | 1 |
| Multiplicativo | 3 | 7 | 1 | 5 | 3 | 3 |
| **Modelos formales (generales)** | **7-10** | **5-10** | **5-10** | **5-10** | **5-10** | **5-10** |
| Diagramas de flujo | 9 | 9 | 9 | 9 | 9 | 9 |
| Causa y efecto | 10 | 5 | 8 | 10 | 10 | 10 |

**Sección 12**

## La ventaja de CA Technologies

CA Technologies (NASDAQ: CA) proporciona soluciones de gestión de TI que ayudan a los clientes a gestionar y proteger entornos de TI complejos para dar soporte a servicios empresariales ágiles. Las organizaciones aprovechan el software y las soluciones de SaaS de CA Technologies para acelerar la innovación, transformar infraestructuras y proteger datos e identidades desde el centro de datos hasta la nube. En CA Technologies estamos comprometidos a asegurarnos de que nuestros clientes logren los resultados y el valor de negocio que esperan mediante el uso de nuestra tecnología. Para obtener más información sobre nuestros programas de éxito de clientes, visite **ca.com/es/customer-success.** Para obtener más información sobre CA Technologies, vaya a **ca.com/es**.

**Comuníquese con CA Technologies en ca.com/es**

CA Technologies (NASDAQ: CA) crea software que impulsa la transformación de las empresas y les permite aprovechar las oportunidades que brinda la economía de las aplicaciones. El software se encuentra en el corazón de cada empresa, sea cual sea su sector. Desde la planificación hasta la gestión y la seguridad, pasando por el desarrollo, CA trabaja con empresas de todo el mundo para cambiar la forma en que vivimos, realizamos transacciones y nos comunicamos, ya sea a través de la nube pública, la nube privada, plataformas móviles, entornos de mainframe o entornos distribuidos. Para obtener más información, visite **ca.com/es**.



CS200-154749