WHITE PAPER | SETTEMBRE 2015

Analisi del Testing

ca technologies

Sommario

Elementi non confrontabili 3

L'esigenza di obiettività 3

L'opportunità 3

L'obiettivo 3

Gli assiomi del testing 4

Le informazioni si trasformano 4

Misurabilità e incertezza associate alle informazioni 4

Meta-matematica e meta-sviluppo 6

Una critica dei metodi di progettazione dei casi di test 7

Difetti osservabili nella logica 7

Rapido e indolore: metodi combinatori 8

La sovraspecializzazione: i metodi di copertura moltiplicativi 9

Modelli formali: un'introduzione 10

Visualizzare requisiti e sistemi: i diagrammi di flusso 11

La specifica della logica hardcore: grafici di causa ed effetto 12

Confronto relativo 13

Il vantaggio di CA Technologies 14

Sezione 1

Elementi non confrontabili

L'esigenza di obiettività

Alla base di questo documento c'è la necessità di un'analisi comparativa delle diverse metodologie di progettazione dei casi di test. Durante la compilazione dell'analisi, mi sono rapidamente reso conto che non esistono criteri oggettivi e quantitativi applicabili a qualsiasi metodologia di progettazione dei casi di test in uso. Sono sicuramente disponibili una serie di trattamenti soggettivi e qualitativi, la maggior parte dei quali fungono da strategia di marketing per qualche vendor. Il problema di quei metodi è che non è possibile dimostrarne i relativi vantaggi: tutto avviene attraverso aneddoti e dati, anziché essere comprovato a partire da principi primi. Quei confronti, cioè, vengono eseguiti a posteriori. In termini di evoluzione dei sistemi, l'analisi a posteriori è valida solo fino a un certo punto, dopodiché diventa necessario provare i progressi ottenuti a priori.

L'opportunità

Per arrivare a un criterio a priori di questo tipo, si è reso necessario abbandonare ogni soggettività e lavorare a partire da un quadro di riferimento essenziale che consentisse l'analisi obiettiva. Questo passaggio è stato un progresso fondamentale in filosofia e in matematica: con la creazione di un quadro di riferimento basato su principi primi (detto anche quadro assiomatico), entrambe le discipline hanno compiuto enormi passi avanti, dato che la dimostrazione dei dati a posteriori non era più necessaria. I tester sono nella posizione migliore per realizzare quanto sia carente la verifica a posteriori, ovvero il metodo di lavoro attualmente accettato. Anziché testare le idee dall'inizio, è considerato accettabile farlo durante e dopo l'implementazione; un po' come collaudare la resistenza di un edificio senza verificare prima che le fondamenta siano solide.

Obiettivo

Lo scopo di questo documento è stabilire un quadro di riferimento obiettivo, che sarà suddiviso in tre concetti chiave collegati:

1. Le informazioni come linguaggio universale, suddivise in misurabili e non misurabili. Tutte le fasi del ciclo di vita di sviluppo del software (SDLC) sono considerate come informazioni in forme diverse.
2. L'incertezza modellata come entropia delle informazioni: i difetti non osservabili e il software non verificabile ricadono in questa categoria.
3. Le trasformazioni delle informazioni: dall'idea alla progettazione, e dalla progettazione all'implementazione. Dal momento che l'SDLC è considerato come una serie di informazioni in forme diverse, le tappe intermedie sono modellate in base alle trasformazioni delle informazioni stesse. Per gran parte del metodo, andremo a considerarle come macchine di Turing.

Durante la compilazione dell'analisi, è emerso che non esistono criteri oggettivi e quantitativi applicabili a qualsiasi metodologia in uso di progettazione dei casi di test.

La maggior parte di queste idee proviene dall'ambito della meccanica quantistica, che può essere considerata semplicemente come modellazione di informazioni a un livello altamente granulare. Nel corso di questo lavoro, il concetto di scala di osservazione sarà ampiamente utilizzato, perché è un fatto evidente che, anche visto a scale differenti, un sistema rimane un sistema; a modificarsi effettivamente sono soltanto le sue specifiche. Ad esempio, è pratica comune tra gli analisti di business costruire i requisiti per un sistema su scale diverse: ad alto livello, seguito da livello leggermente inferiore, mentre i dettagli precisi dell'implementazione vengono forniti al livello più basso. Per fare un altro esempio, è una pratica comune tra gli sviluppatori implementare un sistema di grandi dimensioni come un insieme di sistemi più piccoli, con connettori che li collegano per garantire che funzionino come un tutto unico. Dal momento che requisiti e implementazioni sono entrambi informazioni, e dal momento che entrambi possono essere visti su scale diverse, ha senso introdurre la scala nel nostro metodo di trattamento delle informazioni.

Sezione 2

Gli assiomi del testing

Una volta stabilita l'esigenza di oggettività, il passo successivo è definire effettivamente gli assiomi necessari per la nostra analisi obiettiva. Come illustrato sopra, gli assiomi si basano sul concetto di informazioni della meccanica quantistica. Le informazioni, come concetto astratto, devono essere pensate come un insieme di istruzioni che descrivono qualcosa, oppure sono attuabili. Le informazioni descrittive dovrebbero essere molto semplici da comprendere: data un'entità o un oggetto, l'informazione è il mezzo con cui lo descriviamo. Ad esempio, dato un oggetto, usiamo aggettivi per descriverne le caratteristiche estetiche e avverbi per descriverne il comportamento. Le informazioni attive, d'altro canto, sono costituite da istruzioni che spiegano come creare un determinato oggetto. Ad esempio, una specifica su come assemblare un mobile è considerata un esempio di informazione attiva.

Le informazioni si trasformano

La distinzione tra le due forme è piuttosto sottile, ma molto importante per il nostro metodo, dato che prenderemo in considerazione solo le informazioni attive, in cui i requisiti e i documenti di progettazione possono essere considerati come istruzioni di assemblaggio di un sistema o di un componente software. Dato che i sistemi e il software possono anch'essi essere astratti sotto forma di insieme di istruzioni su come manipolare i dati, anch'essi vanno considerati come informazioni, esattamente come i dati. Questo è uno dei concetti più fondamentali che useremo, quindi è molto importante che venga pienamente compreso. Da esso deriva il secondo dei concetti fondamentali: le informazioni si trasformano. In sintesi, si tratta di trasformazioni che, a partire dalle informazioni in una determinata forma, producono informazioni di output in un'altra. L'esempio più immediato è quello dello sviluppatore: uno sviluppatore trasforma informazioni sotto forma di requisito o progetto in un componente software o sistema.

Emerge così che l'intero SDLC può essere pensato come una catena di diverse forme di informazioni collegate da trasformazioni. Tutto, dai requisiti al prodotto finito, può essere considerato come informazione in una qualche forma.

Misurabilità e incertezza associate alle informazioni

Finora, tuttavia, non abbiamo considerato il concetto di qualità, per il quale dobbiamo richiamare altri due concetti correlati: la misurabilità delle informazioni e l'incertezza a esse associata. In poche parole, la misurabilità delle informazioni è una misura della quantità di informazioni che possiamo effettivamente osservare. L'esistenza delle informazioni è completamente indipendente dalla nostra capacità di acquisirle, e le informazioni non misurabili sono sostanzialmente inutili ai nostri fini, anche se è importante avere almeno un'idea della loro entità effettiva. Ci occuperemo di meta-informazioni (o informazioni di secondo grado) in una sezione separata; per ora, prenderemo in considerazione le informazioni di primo grado. L'incertezza è un concetto associato alla misurabilità delle informazioni: la definiremo come l'entropia delle informazioni, la cui definizione deriviamo direttamente dalla meccanica quantistica, che determina come output informazioni diverse. Per facilitare la concettualizzazione della distinzione tra primo grado e secondo grado, consideriamo quella tra dati e metadati: i metadati esistono per descrivere le proprietà dei dati; si tratta rispettivamente di esempi di informazioni di primo e secondo grado. Le meta-informazioni saranno definite correttamente nella sezione "Meta-matematica e meta-sviluppo".

Questa, in effetti, è una generalizzazione dell'ambiguità nei requisiti: le ambiguità portano a incertezze nell'istruzione, e interpretazioni molteplici rendono possibili più risultati (o implementazioni di quei requisiti). Se il tester e lo sviluppatore scelgono due interpretazioni diverse di uno stesso requisito, è quasi certo che i casi di test non rifletteranno l'implementazione.

Esiste un ultimo insieme di definizioni di cui abbiamo bisogno prima di procedere all'affermazione degli assiomi; dobbiamo cioè distinguere tra due tipi di entropia:

- Epistemica: si tratta delle informazioni nascoste, sotto forma di omissioni: livelli perduti di libertà dovuti alla carenza delle informazioni. Nel mondo reale, questo si traduce in informazioni che semplicemente devono essere "create" durante il processo di trasformazione. Ad esempio, uno sviluppatore a volte è costretto a colmare i vuoti quando il requisito è troppo vago o incompleto. Si tratta di una quantità misurabile, in linea di principio. In pratica, richiede una quantità notevole di lavoro: anche quando gli sviluppatori avanzano ipotesi, questi si manifestano sempre nel codice risultante. È un esempio di ciò che significa, per una trasformazione, essere reversibile.
- Sistemica: è l'entropia insita nel sistema, che non può essere misurata. È paragonabile al principio di indeterminazione di Heisenberg nella meccanica quantistica: nel nostro mondo, deriva dal fatto che gli insiemi di istruzioni/sistemi assiomatici non possono mai essere sia coerenti che completi. Si tratta quindi di una quantità non misurabile; ma utilizzando il concetto di dimensione impostata relativa, è possibile almeno quantificare esattamente il livello di entropia sistemica, poiché l'insieme di insiemi non misurabili è (contrariamente a quanto si potrebbe pensare) misurabile.

Gli assiomi fondamentali del testing sono sei principi che rappresentano conseguenze dirette del quadro di riferimento teorico delle informazioni che abbiamo definito, e si possono riassumere così:

1. Non tutto può essere testato. All'interno di un dato sistema esisterà sempre un certo grado di entropia sistemica; questa è una conseguenza diretta dell'incompletezza di Gödel dei sistemi di assiomi (o indecidibilità di Turing).
2. È sempre possibile sapere cosa può essere testato. L'entropia epistemica può essere annullata, dato un livello sufficiente di informazioni, e può anche essere misurata.
3. I difetti osservabili rappresentano l'entropia misurabile di un piano di test. Dal momento che i difetti (o la loro assenza) sono gli indicatori di qualità di un componente software, è ragionevole trattarli come incertezze. In particolare, i migliori piani di test sono quelli che rendono osservabile il maggiore numero di difetti. Per collegare insieme questi indicatori all'interno di una misurazione, è prassi usuale collegare la gravità ai difetti; la misura diventa allora una media ponderata, anche se è necessario fare attenzione, dato che ponderazioni arbitrarie introducono un certo grado di soggettività, che dovrebbe essere evitato.
4. La copertura è la misura della fedeltà del piano/strategia di test. In altre parole, se il piano/strategia di test riflette accuratamente i requisiti, allora renderà osservabile ogni possibile difetto che lo sia effettivamente. Un piano/strategia di test carente renderà inosservabili molti difetti altrimenti osservabili.
5. L'entropia non si distrugge. Questo deriva direttamente dalla meccanica quantistica: nel mondo reale significa che, se qualche passaggio dell'SDLC introduce entropia, questa non potrà mai essere rimossa, anche se tutte le fasi successive sono fedeli al 100%. In particolare, la qualità di un sistema software è direttamente proporzionale a quella dei suoi requisiti e della strategia di sperimentazione. Si noti che questo non significa che non otterremo mai un software funzionante ma, in termini semplici, che non otterremo un software perfetto.
6. Nessun modello singolo è in grado di individuare tutti i difetti. A causa del primo principio, non tutto può essere testato, ma l'utilizzo di più modelli implica che possono essere resi osservabili insiemi diversi di difetti. Tuttavia, l'esposizione di tutti i difetti richiede almeno un numero infinito di modelli; questa è una conseguenza diretta dei teoremi di incompletezza di Gödel.

Sezione 3

Meta-matematica e meta-sviluppo

Nella nostra precedente discussione sulle informazioni, ci siamo limitati alle informazioni di primo grado, cioè quelle pure. Tuttavia, è anche possibile avere "informazioni sulle informazioni", che possono essere definite meta-informazioni o informazioni di secondo grado. Un buon esempio è rappresentato dalle statistiche di aggregazione: data una popolazione, è possibile ricavare una quantità di informazioni dall'altezza media (e dalla deviazione standard), anche senza conoscere necessariamente l'altezza di ogni singolo individuo.

Per comprendere l'utilità delle meta-informazioni, possiamo categorizzare le informazioni in termini di quella che può essere chiamata una matrice di Rumsfeld (dal nome del segretario della Difesa statunitense), che ha i seguenti elementi:

- Elementi noti noti
- Elementi incogniti noti
- Elementi noti ignoti
- Elementi incogniti ignoti

Possiamo rappresentare tutto questo in una griglia, che raccoglie tutti i concetti che abbiamo affrontato fino a ora:

	Elementi noti	Elementi incogniti
Noto	Informazioni	Entropia epistemica Difetti osservabili
Sconosciuto	Entropia epistemica Difetti osservabili	Entropia sistemica

Le meta-informazioni sono molto utili per ottenere almeno un'idea dell'entità degli elementi che non conosciamo, anche se non disponiamo effettivamente di quelle informazioni. Questo è essenzialmente il motivo per cui è stata sviluppata la disciplina della meta-matematica: stabilire, attraverso l'analisi dei quadri di riferimento matematici, cosa è dimostrabile e cosa non lo è. Anche se un'ampia percentuale di proposizioni matematiche, come l'ipotesi del continuum per quanto riguarda l'esistenza di certi cardinali infiniti, non può essere dimostrata, questo non crea problemi alla comunità matematica in quanto è possibile concentrarsi sulle proposizioni che possono essere dimostrate. Lo stesso dovrebbe valere per i test: con metodi analoghi è possibile concentrarsi su ciò che può essere testato (ovvero gli elementi noti noti) e accontentarsi delle meta-informazioni per il resto. Gli elementi incogniti ignoti (cioè l'entropia sistemica) sono un problema, dal momento che non possono essere misurati. Ma le altre due categorie (elementi incogniti noti ed elementi noti ignoti) possono almeno essere misurate, e di conseguenza è possibile derivare statistiche al riguardo, il che è estremamente utile per l'analisi del rischio.

L'idea centrale di questa analogia, che provvisoriamente chiameremo "meta-sviluppo", è l'idea che il software testabile (cioè gli elementi incogniti noti) dovrebbe focalizzarsi, insieme a tutti i relativi sforzi, sulla massimizzazione delle informazioni disponibili e osservabili; per il resto, le meta-informazioni sono sufficienti. Le metodologie correnti di progettazione dei casi di test, purtroppo, non utilizzano al massimo le informazioni disponibili, come andremo a vedere nella nostra analisi. Se è vero che "la maggior parte" degli algoritmi, in termini di macchine di Turing, non è testabile (come mostrato dal problema della terminazione), rimane un numero infinito di configurazioni che possono essere testate, almeno fino a una copertura funzionale del 100%. In questo contesto, utilizzo il concetto di dimensione relativa della teoria della misura per confrontare insiemi infiniti: l'insieme degli algoritmi non testabili costituisce la maggioranza del numero totale di algoritmi (ovvero un numero infinito non numerabile), mentre l'insieme degli algoritmi testabili costituisce un sottoinsieme trascurabile (un numero infinito numerabile, che nella teoria della misura è considerato un "insieme di misura 0" o un "insieme nullo" il quale, a confronto, è a tutti gli effetti infinitamente piccolo).

Sezione 4

Una critica dei metodi di progettazione dei casi di test

Dati gli assiomi di cui sopra, passiamo ora a fornire una critica oggettiva dei metodi di progettazione dei casi di test. Tenendo presente il primo assioma, non sarà possibile individuare tutti i difetti. Tuttavia, dato il secondo, il terzo e il quarto assioma, disponiamo di criteri utilizzabili per classificare i metodi di progettazione dei casi di test:

1. Quante informazioni sull'applicazione possono essere codificate nel processo di progettazione del caso di test?
2. Quanti difetti vengono resi osservabili?
3. Copertura: quanti difetti vengono resi osservabili mediante questo metodo in proporzione al numero massimo teorico di difetti osservabili?
4. Numero relativo di casi di test necessari per ottenere una copertura ottimale.

Sulla base di questi quattro criteri, a ogni metodo di progettazione di casi di test sarà attribuito un punteggio da 1 a 10, per valutarne:

- **Capacità di codifica**: la quantità di informazioni quantitative sull'applicazione che possono essere codificate nel metodo. (Derivato dal punto 1 sopra).
- **Facilità di codifica**: quanto è facile codificare le informazioni.
- **Applicabilità**: quanti scenari possono essere codificati in modo sensibile mediante il metodo.
- **Numero di casi di test**: il numero relativo di casi di test generati (1 - troppo pochi/troppi, 10 - ottimale). (Punto 4 sopra).
- **Difetti rilevabili**: il numero relativo di difetti che è possibile individuare. (Derivato dal punto 2 sopra).
- **Copertura**: la copertura funzionale relativa che può essere ottenuta. (Derivato dal punto 3 sopra).

Si noti che tutti i punteggi sono compresi tra 1 e 10, dove 10 è il più elevato.

Sezione 5

Difetti osservabili nella logica

Prima di passare oltre, è necessario definire alcune proprietà dei difetti osservabili in relazione alle dichiarazioni logiche. La maggior parte dei modelli formali di test che utilizzano le informazioni sull'applicazione sono fondamentalmente dipendenti dall'analisi delle dichiarazioni logiche, che rappresentano quindi un buon punto di partenza per la ricerca dei difetti.

Consideriamo una frase molto semplice: **SE A E B ALLORA C**

Questa può essere suddivisa in cause (A e B) ed effetti (C). In termini di difetti, entrambe le cause possono avere tre stati: implementato correttamente (OK), bloccato a 0 (0) e bloccato a 1 (1). Ma cosa avviene quando consideriamo i difetti associati a C, date queste informazioni?

Tutti i possibili difetti associati a C sono codificati nella seguente tabella (tutti i difetti sono evidenziati in rosso):

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**	

Come si vede, le ultime tre varianti funzionali (cioè l'insieme di input vero/falso) sono sufficienti per individuare tutti i possibili difetti; la prima variante semplicemente non rileva ulteriori difetti. Così, è possibile dimostrare che tutti i possibili difetti possono essere individuati da questa metodologia; la tabella riporta i dati generali per un operatore logico con n input:

Numero di input	Numero di variazioni funzionali possibili	Numero di combinazioni possibili	Numero di potenziali difetti
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	$\sum_{x=1}^{n} \binom{n}{x} 2^x = 3^n - 1$

Come detto sopra, esistono due classi di difetti: osservabili e non osservabili. Il fattore distintivo non è che i loro effetti siano inosservabili, ma che la loro root cause sia inosservabile. Questo è assolutamente fondamentale al fine di essere certi di ottenere la risposta giusta per la ragione giusta. L'osservabilità in questo contesto può essere pensata come le informazioni minime necessarie per identificare, riprodurre o correggere il difetto; in altre parole, dato un difetto, dovrebbe essere possibile individuare esattamente dove si è verificato. Per molti aspetti, una buona progettazione di casi di test ha lo scopo di fare questo in senso inverso: fornire il numero minimo di casi di test per esporre il maggior numero di difetti osservabili. Ricordiamo che, nel mondo reale, i difetti osservabili rappresentano l'entropia epistemica del software e possono essere annullati e misurati, mentre i difetti inosservabili sono sistemici, e non possono essere testati utilizzando lo stesso modello. La copertura di test, quindi, è una misura del livello di entropia epistemica che può essere annullato dal processo di test.

Sezione 6

Rapido e indolore: metodi combinatori (tutte le coppie e così via)

Derivati da semplici argomenti di conteggio, i metodi combinatori sono la forma più semplice di progettazione di casi di test; in sintesi, dato un elenco di colonne e di possibili valori per ognuna, tutte le possibili combinazioni di coppie, terne e così via vengono derivate e rapidamente compilate. La proprietà più attraente di questi metodi è che non si basano su alcuna conoscenza dell'applicazione; tuttavia, questa è anche la loro carenza più grave. Di conseguenza, il relativo input di informazioni quantitative è limitato ai dati, senza alcun input relativo alle relazioni. In conseguenza dell'assioma di cui sopra, le informazioni qualitative sul sistema evidenziate da questo tipo di test saranno anch'esse relativamente ridotte. Dal momento che non hanno corrispondenza con i punti funzionali effettivi, nessuna determinazione può essere fatta sulle modalità reali di test di gran parte della logica dell'applicazione. Nella maggioranza degli scenari, il livello di test eseguito è endemicamente inferiore al necessario, anche se ci sono alcuni aspetti che vengono sovra-testati inutilmente, a causa di condizioni logiche ridondanti.

I metodi combinatori hanno un secondo punto debole, che è un corollario di quanto sopra: spesso vengono prodotte combinazioni di dati non valide, generando falsi positivi laddove sono i dati a causare un fallimento del test e non un difetto reale. Questo può essere in qualche modo mitigato mediante l'introduzione di vincoli: a dimostrare che, quando viene fornito più input quantitativo, i dati qualitativi generati sono molto più chiari. In terzo luogo, i risultati attesi non possono essere considerati in automatico; ancora una volta, si basano sull'input quantitativo. Metodi superiori consentono la codifica di tali informazioni, e quindi la nostra analisi dimostra che i metodi combinatori sono una scelta molto scadente per la progettazione dei casi di test.

Pertanto, l'uso di metodi combinatori dovrebbe essere limitato esclusivamente a test non funzionali e di configurazione: in altre parole, scenari in cui l'unico risultato atteso è "funziona".

Punteggi, da 1 a 10, per i metodi combinatori:

Informazioni di input			Informazioni di output		
Capacità	Facilità	Applicabilità	Numero di casi di test	Difetti rilevabili	Copertura
1	9	5	2	1	1

Sezione 7

La sovraspecializzazione: I metodi di copertura moltiplicativi

Adattamento altamente specializzato di metodi combinatori, questo metodo è utile solo se lo scenario oggetto di test dipende esclusivamente dal numero di istanze di una particolare entità di dati. Ad esempio, supponiamo di disporre di un database di account di clienti, con una schermata che mostra tutti gli account di un dato cliente. Gli scenari oggetto di test sarebbero:

1. Il cliente non ha alcun account
2. Il cliente ha un unico account
3. Il cliente ha più account

Si noti che questa tecnica presuppone implicitamente una logica funzionale, ergo alcune informazioni sull'applicazione vengono inserite nella progettazione del caso di test, a differenza di quanto avviene per i suoi omologhi più semplici. In realtà, utilizzo questa tecnica ogni volta che mi occupo di relazioni padre-figlio all'interno di un database relazionale o di strutture gerarchiche (come l'XML). Più in generale, questa tecnica trova applicazione durante i test di metodi di aggregazione su strutture di dati complesse. Questo è dovuto al fatto che alcune funzioni di aggregazione presentano casi particolari (o degenerati) che devono essere gestiti separatamente. Ad esempio:

- Generare una media di un insieme di numeri richiede un trattamento particolare per il caso di un insieme vuoto, pena l'ottenimento di un errore di divisione per zero.
- Considerare la deviazione standard di un insieme di numeri richiede un trattamento particolare sia per l'insieme vuoto sia per un insieme a valore singolo; se non vengono gestiti separatamente, il primo porterà a un risultato negativo, il secondo a una divisione per zero.

L'output qualitativo di questo metodo è la conferma che tutti i possibili insiemi di input funzionano, data unicamente la loro dimensione. Questo è un caso speciale di testing di classe di equivalenza, in cui l'intervallo di input è raggruppato in classi disgiunte al fine di ridurre il numero totale di scenari di test, pur mantenendo l'integrità della copertura funzionale.

Punteggi, da 1 a 10, per i metodi di copertura moltiplicativi:

Informazioni di input			Informazioni di output		
Capacità	Facilità	Applicabilità	Numero di casi di test	Difetti rilevabili	Copertura
3	7	1	5	3	3

Sezione 8

Modelli formali: un'introduzione

Il resto di questo articolo sarà dedicato alle tecniche di modellazione formale, che forniscono il livello più elevato di precisione delle informazioni qualitative sul sistema. Come affermato dall'assioma sopra, questo richiede le competenze più elevate, in quanto un'ampia quantità di informazioni quantitative è necessaria per sfruttarlo al massimo. Questi metodi sono unici in termini della quantità di informazioni che consentono di codificare nel processo di progettazione del caso di test.

Prima di procedere, è necessario descrivere brevemente ciò che si intende per modelli formali. In sostanza, i modelli formali sono descrizioni matematicamente precise di un requisito, utilizzabili per eseguire operations ulteriori che forniscono informazioni qualitative su di esso. Le tre operations più importanti sono:

1. Verifica della coerenza: assicura che la logica del requisito sia intrinsecamente coerente. Questo elimina le ambiguità dovute a contraddizioni.
2. Verifica della completezza: assicura che la logica del requisito sia completa. Questo elimina le ambiguità dovute a omissioni.
3. Derivazione dei risultati attesi: la logica coerente e completa fornirà i risultati attesi per ogni possibile scenario, il che consente la derivazione automatica dei risultati attesi per i casi di test. Questa è il vantaggio principale dei modelli formali dal punto di vista di un tester.

I formalismi stessi finiscono per essere più "utili", semplicemente in virtù delle tre operations citate sopra; quanto sopra fornisce informazioni qualitative incredibilmente dettagliate sul sistema oggetto di test. In effetti, se introdotte precocemente all'interno del ciclo di sviluppo, le informazioni qualitative possono essere estratte dai requisiti stessi, assicurando che il progetto poggi su una base solida. Inoltre, la maggior parte delle metodologie di specificazione dei requisiti è inerte e inutile al di fuori della fase di progettazione (fatto salvo un intervento manuale), laddove i modelli formali possono fornire informazioni in tutte le fasi del ciclo di sviluppo. Senza entrare troppo nel dettaglio, questi formalismi costituiscono il fondamento su cui si basa la maggior parte dei progressi matematici e computazionali dell'ultimo secolo circa. Se sono validi per quelle discipline, lo saranno ovviamente anche per la nostra.

I modelli formali sono disponibili in tutte le forme e dimensioni; mi concentrerò quindi sui due più pertinenti ai test: vale a dire, i diagrammi di flusso e i grafici di causa ed effetto. Questi possono essere pensati come forme canoniche: ogni modello formale è in qualche modo simile a uno di essi.

Sezione 9

Visualizzare requisiti e sistemi: i diagrammi di flusso

I diagrammi di flusso, in generale, sono una forma ampiamente comprensibile di specificazione dei requisiti; i loro vantaggi rispetto agli approcci basati su un "muro di testo" sono numerosi, ma nessuno è più evidente della possibilità di derivare casi di test dai requisiti in modo sistematico (risultato ottenibile dagli algoritmi automatizzati). Ho trovato questo approccio, più semplice di quanto possa sembrare a prima vista, difficile da trasmettere.

Per derivare casi di test, è sufficiente valutare i percorsi possibili all'interno del diagramma di flusso. Questo, in sostanza, è tutto (formalmente, si parla di analisi omotopica dei grafici). Il vantaggio diviene evidente all'applicazione di tecniche di ottimizzazione per ridurre il numero di casi di test, ma conservando copertura funzionale grazie alla struttura logica. In sostanza, basta considerare i singoli blocchi decisionali: se ogni blocco (o operatore) è testato completamente in relazione a tutti i percorsi diretti nei due sensi, tutto il flusso si può considerare completamente testato. Questo, più di ogni altra cosa è estremamente difficile (e frustrante) da comunicare, a meno di dichiarare che "la matematica funziona!".

In breve, funziona perché alcuni percorsi sono resi ridondanti a motivo dello stesso insieme di condizioni locali (cioè a livello di operatore) oggetto di test, il che avviene se proviamo a integrarne altre (lontane, cioè globali) oggetto di test; si tratta semplicemente di variazioni funzionali ridondanti che "si annullano" a vicenda. È una di quelle cose che è più facile spiegare in generale piuttosto che per casi specifici (come per gran parte della matematica astratta).

Poiché i diagrammi di flusso consentono la specificazione di requisiti univoci, la quantità di informazioni quantitative che può essere codificata è enorme, diversi ordini di grandezza maggiore rispetto ai due metodi precedenti. Questo, ancora una volta, vale per tutti i metodi di modellazione formali; le uniche differenze sono relative all'ordine di grandezza. Inoltre, a motivo del concetto di test a livello di operatore, la quantità di difetti che può essere rilevata è anch'essa molto ampia, e di conseguenza lo è la quantità di informazioni qualitative ottenute dal modello.

Punteggi, da 1 a 10, per i diagrammi di flusso:

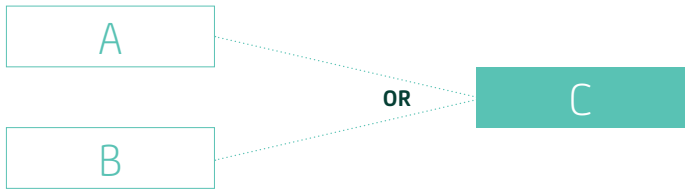
Informazioni di input			Informazioni di output		
Capacità	Facilità	Applicabilità	Numero di casi di test	Difetti rilevabili	Copertura
9	9	9	9	9	9

Sezione 10

Specifica della logica hardcore: grafici di causa ed effetto

Un altro esempio di modello formale: questa volta, le dichiarazioni logiche sono modellate come cause ed effetti, e il rapporto tra loro è completamente specificato. Come i diagrammi di flusso, questo metodo è stato specificamente analizzato grazie alla sua capacità di assicurare che i requisiti verbosi siano specificati in modo univoco.

L'idea è che, come in un diagramma di flusso, i blocchi sono collegati tra loro, ma questa volta sono le relazioni causali a essere specificate. Inoltre, i blocchi sono collegati tra loro mediante operatori logici quali AND, OR e NOT. Ad esempio, la dichiarazione "se A o B allora C" può essere codificata come segue:



Dovrebbe essere evidente che le informazioni possono essere codificate in modo estremamente granulare in questo modo; il metodo è considerato tra i migliori da questo punto di vista. Tuttavia, dovrebbe essere altrettanto evidente che si tratta di un metodo affatto semplice; e, infatti, è considerato tra i più complessi.

La sua capacità di individuare difetti è però impressionante, maggiore di quella dei digrammi di flusso, poiché i difetti possono essere derivati analiticamente da una condizione logica; consultare la sezione sui "difetti osservabili dalla logica" per una spiegazione approfondita del processo. In particolare, si può dimostrare che tutti i difetti possibili possono essere individuati mediante questa metodologia. Come risultato, questa metodologia è di gran lunga la migliore in termini di individuazione dei difetti osservabili, anche se risulta molto complessa rispetto ai diagrammi di flusso; e questo purtroppo ne ha ostacolato l'adozione. In particolare, la difficoltà consiste nel tentativo di codificare requisiti specifici per l'ordine (diversamente dai diagrammi di flusso, che sono ideali per questo scopo, anche se soffrono dell'incapacità di codificare correttamente i requisiti indipendenti dall'ordine).

Punteggi, da 1 a 10, per i grafici di causa ed effetto:

Informazioni di input			Informazioni di output		
Capacità	Facilità	Applicabilità	Numero di casi di test	Difetti rilevabili	Copertura
10	5	8	10	10	10

Sezione 11

Confronto relativo

La seguente tabella mette a confronto i punteggi generali di tutte le modalità di progettazione di casi di test illustrate sopra; tutti i punteggi sono compresi tra 1 e 10 e, pur essendo il metodo il più oggettivo possibile, vengono valutati in base ai seguenti criteri, discussi sopra:

- **Capacità di codifica**: la quantità di informazioni quantitative sull'applicazione che possono essere codificate nel metodo.
- **Facilità di codifica**: quanto è facile codificare le informazioni.
- **Applicabilità**: quanti scenari possono essere codificati in modo sensibile mediante il metodo.
- **Numero di casi di test**: il numero relativo di casi di test generati (1 - troppo pochi/troppi, 10 - ottimale).
- **Difetti rilevabili**: il numero relativo di difetti che è possibile individuare.
- **Copertura**: la copertura funzionale relativa che può essere ottenuta.

NOTA: È inclusa anche la classe di modelli formali compresi in una categoria; i punteggi sono inclusi come intervallo, in quanto alcuni sono più validi, meno applicabili e così via, rispetto ad altri.

Metodo	Informazioni di input			Informazioni di output		
	Capacità	Facilità	Applicabilità	Numero di casi di test	Difetti rilevabili	Copertura
Combinatorio	1	9	5	2	1	1
Moltiplicativo	3	7	1	5	3	3
Modelli formali (generale)	**7-10**	**5-10**	**5-10**	**5-10**	**5-10**	**5-10**
Diagramma di flusso	9	9	9	9	9	9
Causa ed effetto	10	5	8	10	10	10

Sezione 12

Il vantaggio di CA Technologies

CA Technologies (NASDAQ: CA) fornisce soluzioni per la gestione dell'IT in grado di aiutare i clienti a gestire e proteggere ambienti IT complessi a supporto di servizi di business agile. Le aziende sfruttano i software e le soluzioni SaaS di CA Technologies per accelerare l'innovazione, trasformare l'infrastruttura e proteggere dati e identità, dal data center al cloud. CA Technologies è impegnata a garantire che i suoi clienti ottengano i risultati attesi e il business value previsto attraverso l'utilizzo della sua tecnologia. Per ulteriori informazioni sui programmi a supporto dei nostri clienti, visita **ca.com/it/customer-success.** Per ulteriori informazioni su CA Technologies, visita **ca.com/it**.

Entra in contatto con CA Technologies all'indirizzo ca.com/it

CA Technologies (NASDAQ: CA) crea software che promuove l'innovazione all'interno delle aziende, consentendo loro di sfruttare le opportunità offerte dall'economia delle applicazioni. Il software rappresenta il cuore di qualsiasi business, in ogni settore. Dalla pianificazione allo sviluppo, fino alla gestione e alla sicurezza, CA Technologies lavora con le aziende di tutto il mondo per cambiare il nostro modo di vivere, interagire e comunicare, in ambienti mobile, cloud pubblici e privati, distribuiti e mainframe. Per ulteriori informazioni, visita il sito **ca.com/it**.

CS200-154749