WHITE PAPER | SEPTEMBER 2015

Eine Kritik des Testing

Inhaltsverzeichnis

Abschnitt 1

Äpfel und Birnen

Die Notwendigkeit der Objektivität

Anlass dieser Arbeit war die Notwendigkeit einer vergleichenden Analyse unterschiedlicher Designmethoden für Testfälle. Während der Zusammenstellung der Analyse stellte ich schnell fest, dass es keine objektiven und quantitativen Kriterien gibt, die auf alle heute verwendeten Testfall-Designmethoden angewendet werden können. Sicherlich gibt es zahlreiche subjektive und qualitative Vorgehensweisen, von denen die meisten als Marketing für den einen oder anderen Anbieter dienen. Das Problem mit diesen Vorgehensweisen ist, dass ihre vergleichbaren Vorteile nicht nachgewiesen werden können – alles wird durch Einzelfälle und Daten begründet, statt von Grundprinzipien aus bewiesen zu werden. Dies bedeutet, dass derartige Vergleiche im Nachhinein durchgeführt werden. Hinsichtlich der Weiterentwicklung von Systemen hat eine im Nachhinein durchgeführte Analyse ihre Grenzen: Irgendwann müssen Fortschritte vorab bewiesen werden.

Chance

Um Kriterien für eine solche Vorabanalyse zu entwickeln, wurde es notwendig, jegliche Subjektivität aufzugeben und von einem grundlegenden Framework aus zu arbeiten, das objektive Analysen gestattete. Dieser Schritt war ein fundamentaler Durchbruch in der Philosophie und der Mathematik: Durch Erstellung eines Frameworks, das auf Grundprinzipien beruht (eines sogenannten axiomatischen Frameworks), konnten beide Disziplinen sich enorm schnell weiterentwickeln, da sie nicht mehr im Nachhinein Belege durch Daten erbringen mussten. Tester sollten sich bewusst sein, wie unsinnig eine Verifizierung im Nachhinein ist – denn diese ist immer noch die verbreitete Vorgehensweise. Statt Ideen von ihrer Entstehung an zu testen, wird es als akzeptabel betrachtet, während und nach der Implementierung zu testen – das ist so, als würde man die Stärke eines Gebäudes testen, ohne zu überprüfen, ob die Fundamente standfest sind!

Zweck

Der Zweck dieser Arbeit besteht darin, ein solches objektives Framework einzurichten. Dieses wird in drei miteinander in Beziehung stehende Kernkonzepte aufgeteilt:

1. Informationen als die universelle Sprache, aufgeteilt in messbare und nicht messbare Informationen: Alle Phasen des Softwareentwicklungszyklus (SDLC) werden als Informationen in unterschiedlichen Formen betrachtet.
2. Unsicherheit, die als Informationsentropie modelliert wird: In diese Kategorie fallen nicht beobachtbare Fehler und nicht testbare Software.
3. Transformationen von Informationen: von der Idee zum Design und vom Design zur Implementierung. Da der SDLC als eine Abfolge unterschiedlicher Formen von Informationen angesehen wird, werden die Zwischenphasen als Informationstransformationen modelliert. In einem Großteil des Artikels betrachten wir sie als Turing-Maschinen.

Während der Zusammenstellung der Analyse wurde festgestellt, dass es keine objektiven und quantitativen Kriterien gibt, die auf alle heute verwendeten Testfall-Designmethoden angewendet werden können.

Die meisten dieser Ideen stammen aus dem Bereich der Quantenmechanik, die einfach als ein Modeling von Informationen auf sehr differenziertem Level angesehen werden kann. In diesem gesamten Artikel wird das Konzept des Beobachtungsmaßstabs umfassend genutzt, denn es ist eine einfache Tatsache, dass ein System, mit unterschiedlichen Maßstäben betrachtet, weiterhin ein System ist – nur die spezifischen Aspekte sind tatsächlich anders. Beispielsweise ist es unter Business-Analysten verbreitet, Anforderungen für ein System mit unterschiedlichen Maßstäben zu erstellen: auf hohem Level, gefolgt von einem etwas niedrigeren Level, während genaue Details zur Implementierung auf dem niedrigsten Level bereitgestellt werden. Ein weiteres Beispiel: Es ist unter Entwicklern auch üblich, ein großes System als eine Gruppe kleinerer Systeme zu implementieren, mit Konnektoren zwischen ihnen, die sicherstellen sollen, dass sie als ein einzelnes Ganzes funktionieren. Da es sich sowohl bei Anforderungen als auch bei Implementierungen um Informationen handelt und beide mit unterschiedlichen Maßstäben betrachtet werden können, ist es sinnvoll, den Maßstab beim Umgang mit Informationen zu berücksichtigen.

Abschnitt 2

Grundsätze des Testing

Nachdem wir nun festgestellt haben, dass Objektivität erforderlich ist, besteht der nächste Schritt darin, die notwendigen Axiome für unsere objektive Analyse tatsächlich zu etablieren. Wie oben veranschaulicht basieren die Axiome auf dem quantenmechanischen Konzept der Informationen. Informationen als abstraktes Konzept sollten als Mengen von Anweisungen betrachtet werden, die entweder etwas beschreiben oder als eine Handlungsgrundlage dienen können. Beschreibende Informationen sollten sehr einfach zu verstehen sein: Für eine Entität oder ein Objekt sind Informationen das Mittel, mit dem wir sie/es beschreiben. Für ein gegebenes Objekt verwenden wir beispielsweise Adjektive, um seine ästhetischen Merkmale zu beschreiben, und Adverbien, um zu beschreiben, wie es auf die Welt einwirkt. Aktive Informationen hingegen bestehen aus Anweisungen, die erläutern, wie ein gegebenes Objekt konstruiert wird. Beispielsweise wird eine Spezifikation, in der erläutert wird, wie ein Möbelstück zusammengebaut wird, als eine aktive Information betrachtet.

Informationstransformationen

Der Unterschied zwischen den beiden Formen ist eher subtil, aber er ist sehr wichtig für unsere Überlegungen, da wir nur aktive Informationen betrachten werden: Anforderungen und Designdokumente können als Anweisungen dazu betrachtet werden, wie ein System oder eine Softwarelösung konstruiert werden soll. Da Systeme und Software abstrakt selbst als Mengen von Anweisungen für die Bearbeitung von Daten betrachtet werden können, müssen wir sie, wie Daten, als Informationen betrachten. Dies ist eines der grundlegendsten Konzepte, die wir verwenden werden. Daher ist es sehr wichtig, es vollständig zu verstehen. Es führt zum zweiten unserer wichtigsten Konzepte: Informationstransformationen. Einfach gesagt sind dies Transformationen, die Informationen in einer Form akzeptieren und Informationen in einer anderen Form ausgeben. Das naheliegendste Beispiel ist das des Entwicklers: Ein Entwickler transformiert Informationen in Form einer Anforderung oder eines Designs in eine Softwarelösung oder ein System.

Der gesamte SDLC kann als Kette unterschiedlicher Informationen mit dazwischen befindlichen Transformationen angesehen werden. Alles von den Anforderungen bis zum fertigen Produkt kann als Informationen in irgendeiner Form angesehen werden.

Messbarkeit und Unsicherheit von Informationen

Wir haben bisher jedoch noch nicht das Konzept der Qualität betrachtet. Für dieses müssen wir zwei weitere, verwandte Konzepte betrachten: die Messbarkeit von Informationen und die Unsicherheit, die mit Informationen verbunden ist. Einfach gesagt bedeutet die Messbarkeit von Informationen, dass wir ermitteln können, wie viel Information wir tatsächlich wahrnehmen können. Dass die Information existiert, ist vollständig unabhängig von unserer Fähigkeit, sie zu erfassen, und nicht messbare Informationen sind für uns im Wesentlichen nutzlos – auch wenn es wichtig ist, dass wir immerhin eine gewisse Vorstellung davon haben, wie umfangreich diese nicht messbare Information tatsächlich ist. Wir werden in einem separaten Abschnitt über Metainformationen (oder Informationen zweiter Ordnung) sprechen; zunächst betrachten wir jedoch Informationen erster Ordnung. Unsicherheit ist ein Konzept im Zusammenhang mit der Messbarkeit von Informationen: Wir bezeichnen sie als die Informationsentropie, deren Definition wir direkt aus der Quantenmechanik entlehnen, die also zu veränderten Informationsausgaben führt. Um die Unterscheidung zwischen Informationen erster und zweiter Ordnung besser zu verstehen, betrachten Sie die Unterscheidung zwischen Daten und Metadaten: Die Aufgabe von Metadaten besteht darin, Eigenschaften von Daten zu beschreiben. Dies ist also ein Beispiel für Informationen erster bzw. zweiter Ordnung. Metainformationen werden im Abschnitt „Metamathematik und Metaentwicklung“ formal definiert.

Dies ist effektiv eine Verallgemeinerung von Uneindeutigkeiten in Anforderungen: Uneindeutigkeiten führen zu Unsicherheiten in der Anweisung, und mehrere Interpretationen lassen mehr als eine mögliche Ausgabe (oder Implementierung dieser Anforderungen) zu. Wenn der Tester und der Entwickler zwei unterschiedliche Interpretationen derselben Anforderung auswählen, ist es fast sicher, dass die Testfälle die Implementierung nicht widerspiegeln werden.

Wir benötigen noch eine letzte Gruppe von Definitionen, bevor wir die Grundsätze aufführen. Wir müssen nämlich zwischen zwei Arten von Entropie unterscheiden:

- Epistemisch – dies sind verborgene Informationen im Sinne von Auslassungen: verlorene Freiheitsgrade, die daher stammen, dass die Informationen einfach nicht da sind. In unserer Welt führt dies zu Informationen, die während des Transformationsprozesses einfach „ausgedacht" werden müssen. Beispielsweise muss ein Entwickler manchmal einfach die Lücken füllen, wenn die Anforderung zu vage oder unvollständig ist. Dies ist im Prinzip eine messbare Quantität. In der Praxis erfordert es beträchtlichen Arbeitsaufwand: Auch wenn Entwickler Annahmen treffen, sind diese stets im erstellten Code erkennbar. Dies ist ein Beispiel dafür, was der Begriff einer reversiblen Transformation bedeutet.
- Systemisch – dies ist inhärente, nicht messbare Entropie im System. Sie ist vergleichbar mit der Heisenbergschen Unschärferelation in der Quantenmechanik – in unserer Welt begründet sie sich in der Tatsache, dass Anweisungsmengen/Axiomensysteme niemals sowohl konsistent als auch vollständig sein können. Als solche ist dies eine nicht messbare Quantität, aber mithilfe des Konzepts der relativen Größe von Mengen ist es möglich, zumindest genau zu quantifizieren, wie viel systemische Entropie vorhanden ist – da die Menge nicht messbarer Mengen (auch wenn dies zunächst widersinnig erscheint) messbar ist.

Die wesentlichen Grundsätze des Testing sind sechs Prinzipien, die direkte Konsequenzen des informationstheoretischen Frameworks sind, das wir eingerichtet haben, und die wie folgt zusammengefasst werden können:

1. Nicht alles kann getestet werden. In jedem gegebenen System ist immer systemische Entropie vorhanden – dies ist eine direkte Folgerung aus Gödels Unvollständigkeitssätzen für Axiomensysteme (oder Turings Unentscheidbarkeit).
2. Es ist immer möglich, zu wissen, was getestet werden kann. Epistemische Entropie kann aufgehoben werden, wenn genug Informationen vorhanden sind, und sie kann auch gemessen werden.
3. Beobachtbare Fehler stellen die messbare Entropie eines Testplans dar. Da Fehler (oder ihre Abwesenheit) die Indikatoren für die Qualität einer Softwarelösung sind, ist es sinnvoll, sie als Unsicherheiten zu behandeln. Insbesondere sind die besten Testing-Pläne diejenigen, die die meisten Fehler beobachtbar machen. Um diese Indikatoren zu einem Maß zu verknüpfen, ist es üblich, Fehlern Schweregrade zuzuordnen. Damit wird die Messung zu einem gewichteten Durchschnittswert. Dies ist jedoch mit Vorsicht zu behandeln, da beliebig gewählte Gewichtungen eine gewisse Subjektivität einbringen, die vermieden werden sollte.
4. Die Abdeckung ist das Maß für die Genauigkeit des Testplans oder der Teststrategie. In anderen Worten: Wenn der Testplan oder die Teststrategie die Anforderungen präzise widerspiegelt, lässt er/sie uns jeden möglichen beobachtbaren Fehler auch tatsächlich beobachten. Eine schlechte Testplanung/-strategie macht viele Fehler, die andernfalls beobachtbar wären, zu nicht beobachtbaren Fehlern.
5. Entropie kann niemals verloren gehen. Dies stammt direkt aus der Quantenmechanik – in unserer Welt bedeutet es: Wenn irgendein Schritt im SDLC zu Entropie führt, kann diese niemals entfernt werden, auch wenn alle folgenden Phasen zu 100 % genau sind. Insbesondere ist die Qualität eines Softwaresystems direkt proportional zur Qualität der Anforderungen und der Testing-Strategie. Beachten Sie, dass dies nicht bedeutet, dass wir niemals funktionierende Software bekommen: Einfach gesagt bedeutet es, dass wir niemals perfekte Software bekommen.
6. Kein einzelnes Modell kann alle Fehler aufdecken. Aufgrund des ersten Prinzips kann nicht alles getestet werden, aber die Verwendung mehrerer Modelle bedeutet, dass unterschiedliche Mengen von Fehlern beobachtbar gemacht werden können. Um alle Fehler sichtbar zu machen, sind jedoch mindestens unendlich viele Modelle erforderlich – dies ist eine direkte Folgerung aus Gödels Unvollständigkeitssätzen.

Abschnitt 3

Metamathematik und Metaentwicklung

Bei unserer obigen Erörterung von Informationen haben wir uns auf Informationen erster Ordnung beschränkt, also auf reine Informationen. Es ist jedoch auch möglich, „Informationen über Informationen" zu besitzen, die als Metainformationen oder Informationen zweiter Ordnung bezeichnet werden können. Ein gutes Beispiel hierfür sind aggregierende Statistiken: Für eine Grundmenge von Menschen können viele Informationen aus dem Durchschnitt (und der Standardabweichung) der Körpergröße abgeleitet werden, auch wenn wir nicht unbedingt die Größe jeder einzelnen Person kennen.

Um zu sehen, warum Metainformationen nützlich sind, kategorisieren wir Informationen anhand einer sogenannten Rumsfeld-Matrix (benannt nach dem US-Verteidigungsminister), die die folgenden Elemente enthält:

- Bekanntes Wissen
- Unbekanntes Wissen
- Bekanntes Unwissen
- Unbekanntes Unwissen

Wir können dies als Raster darstellen – alle Konzepte, die wir bisher behandelt haben, wurden eingetragen:

	Wissen	Unwissen
Bekannt	Informationen	Epistemische Entropie Beobachtbare Fehler
Unbekannt	Epistemische Entropie Beobachtbare Fehler	Systemische Entropie

Metainformationen sind sehr nützlich, um zumindest den Umfang dessen einzuschätzen, was wir nicht wissen, auch wenn wir über die Informationen nicht verfügen. Dies ist im Wesentlichen die Aufgabe, für die die Metamathematik entwickelt wurde: über Analysen mathematischer Frameworks zu ermitteln, was beweisbar ist und was nicht. Auch wenn ein großer Teil der mathematischen Aussagen, wie die Kontinuumshypothese zum Vorhandensein bestimmter unendlicher Kardinalzahlen, nicht bewiesen werden kann, entstehen daraus keine Probleme für die Mathematiker, da es möglich ist, sich auf Aussagen zu konzentrieren, die beweisbar sind. Das Gleiche sollte für das Testing gelten: Mithilfe analoger Methoden ist es möglich, sich auf die Dinge zu konzentrieren, die getestet werden können (d. h. das bekannte Wissen), und sich für die übrigen mit Metainformationen zufriedenzugeben. Unbekanntes Unwissen (d. h. systemische Entropie) stellt ein Problem dar, da es nicht messbar ist. Die anderen beiden Abschnitte (bekanntes Unwissen und unbekanntes Wissen) können immerhin gemessen werden, und daher können Statistiken abgeleitet werden. Dies ist extrem nützlich für Risikoanalysen.

Die Hauptidee dieser Analogie, die ich provisorisch als „Metaentwicklung" bezeichne, ist der Gedanke, dass der Schwerpunkt auf testbarer Software (d. h. bekanntem Wissen) liegen sollte und dass alle Bemühungen darauf gerichtet werden sollten, die verfügbaren und beobachtbaren Informationen maximal zu nutzen – für den Rest genügen Metainformationen. Aktuelle Designmethoden für Testfälle machen leider nicht das Beste aus den verfügbaren Informationen, wie wir in unserer Kritik sehen werden. Auch wenn es wahr ist, dass „die meisten" Algorithmen, dargestellt als Turing-Maschinen, nicht testbar sind (wie das Halteproblem zeigt), gibt es unendlich viele Konfigurationen, die getestet werden können, mindestens bis zu dem Punkt 100 %-iger funktionaler Abdeckung. In diesem Kontext verwende ich das maßtheoretische Konzept der relativen Größe, um unendliche Mengen zu vergleichen – die Menge der nicht testbaren Algorithmen bildet die Mehrheit der Gesamtanzahl der Algorithmen (d. h. eine nicht abzählbar unendliche Anzahl), während die Menge der testbaren Algorithmen eine vernachlässigbare Teilmenge bildet (eine abzählbar unendliche Anzahl, die in der Maßtheorie als „Menge vom Maß null" oder „Nullmenge" betrachtet wird und die vergleichsweise unendlich klein ist).

Abschnitt 4

Eine Kritik von Testfall-Designmethoden

Anhand der oben beschriebenen wesentlichen Grundsätze nähern wir uns nun einer objektiven Kritik von Testfall-Designmethoden. Angesichts des ersten Grundsatzes wird es nicht möglich sein, alle Fehler zu entdecken. Aufgrund des zweiten, dritten und vierten Grundsatzes besitzen wir jedoch Kriterien, anhand derer wir Testfall-Designmethoden bewerten können:

1. Wie viele Anwendungsinformationen können in den Testfall-Designprozess integriert werden?
2. Wie viele Fehler werden beobachtbar gemacht?
3. Abdeckung: Wie viele Fehler werden durch diese Methode beobachtbar gemacht, gesehen als Anteil der theoretischen maximalen Anzahl beobachtbarer Fehler?
4. Relative Anzahl der Testfälle, die erforderlich ist, um optimale Abdeckung zu erreichen.

Basierend auf diesen vier Kriterien erhält jede Testfall-Designmethode eine Bewertung auf einer Skala von 1 bis 10. Bewertet werden dabei:

- **Integrationsfähigkeit**: Dies ist der Umfang quantitativer Information zu der Anwendung, der in diese Methode integriert werden kann (abgeleitet von Nr. 1 oben).
- **Leichtigkeit der Integration**: gibt an, wie leicht die Integration der Informationen ist.
- **Anwendbarkeit**: gibt an, wie viele Szenarien mit dieser Methode sinnvollerweise integriert werden können.
- **Anzahl Testfälle**: die relative Anzahl generierter Testfälle (1 – zu wenige/zu viele, 10 – optimal) (Nr. 4 oben).
- **Erkennbare Fehler**: die relative Anzahl der Fehler, die gefunden werden können (abgeleitet von Nr. 2 oben).
- **Abdeckung**: die relative funktionale Abdeckung, die erreicht werden kann (abgeleitet von Nr. 3 oben).

Beachten Sie, dass alle Bewertungen zwischen 1 und 10 liegen, wobei 10 „am besten“ ist.

Abschnitt 5

Beobachtbare Fehler in Logik

Bevor wir *in medias res* gehen, müssen wir zuerst einige Eigenschaften beobachtbarer Fehler in Relation zu logischen Aussagen notieren. Die meisten formalen Modelle für das Testing, die Anwendungsinformationen verwenden, hängen maßgeblich von der Analyse logischer Aussagen ab. Daher ist dies ein guter Ort, um mit der Suche nach Fehlern zu beginnen.

Betrachten wir einen sehr einfachen Satz: **WENN A UND B, DANN C**

Dieser kann in Ursachen (A und B) und Wirkungen (C) aufgeteilt werden. In Bezug auf Fehler kann jede der Ursachen drei Status aufweisen: korrekt implementiert (OK), dauerhaft 0 (0) und dauerhaft 1 (1). Aber was passiert, wenn wir angesichts dieser Informationen die Fehler betrachten, die mit C in Zusammenhang stehen?

Alle möglichen Fehler im Zusammenhang mit C sind in der folgenden Tabelle dargestellt (alle Fehler sind rot hervorgehoben):

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**	

Wie Sie sehen können, sind die letzten drei funktionalen Variationen (d. h. Menge von Wahr-/Falsch-Eingaben) ausreichend, um alle möglichen Fehler zu erkennen – die erste Variation erkennt einfach keine weiteren Fehler. Daher kann bewiesen werden, dass alle möglichen Fehler mit dieser Methode erkannt werden können – diese Tabelle enthält die allgemeinen Zahlen für einen logischen Operator mit n Eingaben:

Anzahl Eingaben	Anzahl erforderlicher funktionaler Variationen	Anzahl möglicher Kombinationen	Anzahl möglicher Fehler
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	$\sum_{x=1}^{n} \binom{n}{x} 2^x = 3^n - 1$

Wie oben angegeben gibt es zwei Klassen von Fehlern: beobachtbare und nicht beobachtbare. Was sie unterscheidet, ist nicht, dass ihre Auswirkungen nicht beobachtbar wären, sondern dass ihre eigentlichen Ursachen nicht beobachtbar sind. Dies ist sehr wichtig, um sicherzustellen, dass wir die richtige Antwort aus dem richtigen Grund bekommen. Beobachtbarkeit können wir uns in diesem Kontext vorstellen als die Informationen, die mindestens erforderlich sind, um den Fehler zu identifizieren, zu reproduzieren oder zu beheben – in anderen Worten: Für einen gegebenen Fehler sollte es möglich sein, genau zu orten, wo der Fehler aufgetreten ist. In vielen Hinsichten ist ein gutes Testfalldesign bemüht, dies in die umgekehrte Richtung zu tun: die minimale Anzahl an Testfällen bereitzustellen, um die meisten beobachtbaren Fehler aufzudecken. Erinnern Sie sich daran, dass beobachtbare Fehler in unserer Welt die epistemische Entropie der Software darstellen und dass diese aufgehoben und gemessen werden kann, während nicht beobachtbare Fehler systemisch sind und nicht mit demselben Modell getestet werden können. Die Testabdeckung ist daher ein Maß für den Umfang der epistemischen Entropie, die durch den Testing-Prozess aufgehoben werden kann.

Abschnitt 6

Quick and Dirty – kombinatorische Methoden (alle Paare usw.)

Kombinatorische Methoden werden von einfachen Zählargumenten abgeleitet und sind die einfachste Art des Testfalldesigns – einfach gesagt: Wenn eine Liste von Spalten und möglichen Werten für jede Spalte gegeben ist, werden alle möglichen Kombinationen von Paaren, Tripeln usw. abgeleitet und schnell mit Daten gefüllt. Die attraktivste Eigenschaft dieser Methoden besteht darin, dass sie keinerlei Anwendungswissen voraussetzen; dies ist jedoch auch ihre größte Schwäche. Daher werden als Eingabe quantitativer Informationen für sie nur Daten verwendet, aber keine Informationen zu den Beziehungen zwischen diesen Daten. Gemäß dem oben beschriebenen wesentlichen Grundsatz wird ein solches Testing auch nur relativ wenige qualitative Informationen zum System ans Licht bringen. Da keinerlei Zuordnung zu tatsächlichen funktionalen Punkten vorliegt, kann nicht ermittelt werden, wie viel von der Anwendungslogik tatsächlich getestet wird. In den meisten Szenarien wird endemisch zu wenig Testing durchgeführt, während andererseits aufgrund redundanter logischer Bedingungen einige Aspekte unnötigerweise übermäßig getestet werden.

Kombinatorische Methoden weisen eine zweite Schwäche auf, die als Korollar aus dem Obigen folgt: Häufig werden ungültige Datenkombinationen erzeugt, die zu falsch positiven Ergebnissen führen, da die Daten zum Misserfolg des Tests führen und nicht ein tatsächlicher Fehler. Dies kann etwas vermindert werden, indem Einschränkungen eingeführt werden: Dies zeigt, dass mehr quantitative Eingaben zur Erzeugung von klareren qualitativen Daten führen. Drittens können erwartete Ergebnisse nicht automatisch berücksichtigt werden – auch diese beruhen auf quantitativen Eingaben. Bessere Methoden ermöglichen die Integration solcher Informationen. Unsere Analyse zeigt also, dass kombinatorische Methoden für das Testfalldesign eine sehr schlechte Wahl sind.

Daher sollte die Verwendung kombinatorischer Methoden auf das nicht funktionale Testing und das Konfigurations-Testing beschränkt werden, also auf Szenarien, in denen das einzige erwartete Ergebnis lautet „es funktioniert".

Bewertung kombinatorischer Methoden auf einer Skala von 1 bis 10:

Eingabeinformationen			Ausgabeinformationen		
Fähigkeit	Leichtigkeit	Anwendbarkeit	Anzahl Testfälle	Erkennbare Fehler	Abdeckung
1	9	5	2	1	1

Abschnitt 7

Überspezialisiert: multiplikative Abdeckungsmethoden

Als hoch spezialisierte Adaptierung kombinatorischer Methoden ist diese Methode nur dann nützlich, wenn das zu testende Szenario ausschließlich von der Anzahl der Instanzen einer bestimmten Datenentität abhängt. Nehmen Sie beispielsweise an, dass wir über eine Datenbank mit Kundenaccounts verfügen und dass auf einem Bildschirm alle Accounts eines bestimmten Kunden angezeigt werden. Die zu testenden Szenarien sind dann:

1. Kunde besitzt keine Accounts
2. Kunde besitzt einen einzigen Account
3. Kunde besitzt mehrere Accounts

Beachten Sie, dass diese Technik implizit eine gewisse funktionale Logik voraussetzt. Daher gehen einige Anwendungsinformationen in das Testfalldesign ein, anders als bei den einfacheren Methoden. Ich verwende diese Technik jedes Mal, wenn ich mit Relationen zwischen übergeordneten und untergeordneten Objekten in einer relationalen Datenbank oder in hierarchischen Strukturen (wie XML) arbeite. Allgemein wird diese Technik häufig verwendet, wenn Aggregationsmethoden für komplexe Datenstrukturen getestet werden. Dies liegt daran, dass einige Aggregationsfunktionen Spezialfälle (oder degenerierte Fälle) aufweisen, die separat behandelt werden müssen. Beispiele:

- Wenn Sie den Durchschnitt einer Menge von Zahlen bilden möchten, müssen Sie den Fall der leeren Menge gesondert betrachten – andernfalls erhalten Sie einen Fehler, weil versucht wird, durch null zu teilen.
- Wenn Sie die Standardabweichung einer Menge von Zahlen berechnen, müssen Sie die leere Menge und eine einelementige Menge gesondert behandeln – andernfalls führt die erstere zu einem negativen Ergebnis und die letztere zu einer Division durch null.

Die qualitative Ausgabe einer solchen Methode ist die Bestätigung, dass alle möglichen Mengen von Eingaben funktionieren, und zwar rein anhand ihrer Größe. Dies ist ein Spezialfall von Äquivalenzklassen-Testing, bei dem das Spektrum der Eingaben in disjunkte Klassen gruppiert wird, um die Gesamtzahl der Testszenarien zu verkleinern und zugleich die Integrität der funktionalen Abdeckung zu wahren.

Bewertung multiplikativer Abdeckungsmethoden auf einer Skala von 1 bis 10:

Eingabeinformationen			Ausgabeinformationen		
Fähigkeit	Leichtigkeit	Anwendbarkeit	Anzahl Testfälle	Erkennbare Fehler	Abdeckung
3	7	1	5	3	3

Abschnitt 8

Formale Modelle – eine Einführung

Im Rest dieses Artikels sprechen wir über Techniken des formalen Modeling, die die genauesten qualitativen Informationen zum System bieten. Wie der obige Grundsatz besagt, sind hierfür die umfangreichsten Fachkenntnisse erforderlich, da für eine optimale Nutzung umfassende quantitative Informationen notwendig sind. Diese Methoden sind einzigartig hinsichtlich des Umfangs an Informationen, der in den Testfall-Designprozess integriert werden kann.

Bevor wir fortfahren, sollten wir kurz beschreiben, was mit formalen Modellen gemeint ist. Im Wesentlichen sind formale Modelle mathematisch präzise Beschreibungen einer Anforderung, wobei weitere Vorgänge durchgeführt werden können, die qualitative Informationen dazu bereitstellen. Die wichtigsten drei Vorgänge sind:

1. Konsistenzprüfungen: stellen sicher, dass die Logik der Anforderung intern konsistent ist. Dies eliminiert Uneindeutigkeiten durch Widersprüche.
2. Vollständigkeitsprüfungen: stellen sicher, dass die Logik der Anforderung vollständig ist. Dies eliminiert Uneindeutigkeiten durch Auslassungen.
3. Ableitung erwarteter Ergebnisse: konsistente und vollständige Logik führt zu den erwarteten Ergebnissen für jedes mögliche Szenario. Daher können erwartete Ergebnisse für Testfälle automatisch abgeleitet werden. Dies ist aus der Sicht der Tester die größte Stärke formaler Modelle.

Die Formalismen selbst bieten sich als „nützlicher“ an, einfach aufgrund der drei obigen Vorgänge – sie alle bieten unglaublich detaillierte qualitative Informationen über das zu testende System. Wenn sie früher in den Entwicklungszyklus eingeführt werden, können qualitative Informationen sogar aus den Anforderungen selbst extrahiert werden. So wird sichergestellt, dass das Projekt mit einer starken Grundlage beginnt. Außerdem sind die meisten Spezifikationsmethoden für Anforderungen außerhalb der Designphase (abgesehen von manuellen Eingriffen) inaktiv und nutzlos, während formale Modelle Informationen für alle Phasen des Entwicklungszyklus bereitstellen können. Ohne dies zu detailliert beschreiben zu wollen: Derartige Formalismen bilden den Grundstein, auf dem die meisten Fortschritte der Mathematik und der Informatik in den letzten circa hundert Jahren basieren. Was für sie nützlich ist, kann auch für uns nützlich sein.

Es gibt sehr viele unterschiedliche formale Modelle. Ich konzentriere mich auf die beiden, die für das Testing am relevantesten sind: Flowcharts und Ursache-Wirkungs-Diagramme. Diese können als die kanonischen Formen betrachtet werden – jedes formale Modell ähnelt in gewisser Weise einem von ihnen.

Abschnitt 9

Visualisierung von Anforderungen und Systemen – Flowcharts

Die Verwendung von Flowcharts ist im Großen und Ganzen eine verbreitete Form der Spezifikation von Anforderungen. Sie hat gegenüber Ansätzen mit langen Fließtexten viele Vorteile. Der nützlichste davon ist die Möglichkeit, Testfälle systematisch aus den Anforderungen abzuleiten (was automatisierte Algorithmen leisten können) Dies ist leichter, als es auf den ersten Blick aussieht, und es ist eine Denkweise, die nach meiner Erfahrung schwer mitzuteilen ist.

Um Testfälle abzuleiten, werten Sie einfach die möglichen Pfade durch das Flowchart aus. Das ist im Grunde schon alles (formal wird dies als Analyse der Homotopie von Graphen bezeichnet). Um dies intelligent zu nutzen, werden Optimierungstechniken angewendet, um die Anzahl der Testfälle zu reduzieren, wobei die funktionale Abdeckung dank der logischen Struktur beibehalten wird. Im Wesentlichen sind hierbei nur einzelne Entscheidungsblöcke zu betrachten: Wenn jeder Block (oder Operator) hinsichtlich aller direkten Ein- und Ausgänge vollständig getestet wurde, wird das gesamte Flowchart als vollständig getestet angesehen. Vor allem dies ist extrem schwierig (und frustrierend) mitzuteilen – ohne zu sagen: „Das funktioniert, weil es Mathematik ist!“

Kurz gesagt funktioniert es, weil einige Pfade redundant werden, wenn dieselbe Menge lokaler Bedingungen (d. h. auf dem Level der Operatoren) getestet wird. Dies passiert, wenn wir andere (weit entfernte, d. h. globale) Tests einbeziehen – es liegt einfach daran, dass redundante funktionale Variationen einander „aufheben“. Dies ist einer der Sachverhalte, die leichter allgemein zu erklären sind als für spezifische Instanzen (was für große Bereiche der abstrakten Mathematik gilt).

Da mit Flowcharts eindeutige Anforderungen spezifiziert werden können, können quantitative Informationen in riesigem Umfang integriert werden – mehrere Größenordnungen über denen in den beiden vorherigen Methoden. Dies wiederum gilt für alle Methoden des formalen Modeling – die Unterschiede sind nur graduell. Außerdem können dank des Konzepts des Testing auf Operatorlevel sehr viele Fehler gefunden werden. Daher können von dem Modell entsprechend sehr umfangreiche qualitative Informationen erhalten werden.

Bewertung von Flowcharts auf einer Skala von 1 bis 10:

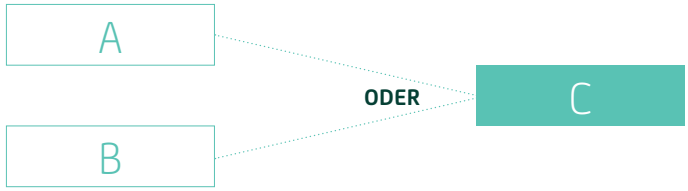
Eingabeinformationen			Ausgabeinformationen		
Fähigkeit	Leichtigkeit	Anwendbarkeit	Anzahl Testfälle	Erkennbare Fehler	Abdeckung
9	9	9	9	9	9

Abschnitt 10

Strenge Logikspezifikation – Ursache-Wirkungs-Diagramme

Dies ist ein weiteres Beispiel für ein formales Modell – dieses Mal werden logische Aussagen als Ursachen und Wirkungen modelliert, und die Beziehungen zwischen ihnen werden vollständig spezifiziert. Wie Flowcharts wurde dieses Modell speziell dank seiner Möglichkeiten analysiert, sicherzustellen, dass aus Anforderungen, die durch lange Fließtexte beschrieben sind, eindeutige Spezifikationen erstellt werden.

Wie bei Flowcharts werden Blöcke miteinander verbunden, aber hier werden Ursachenbeziehungen spezifiziert. Außerdem werden Blöcke mit logischen Operatoren wie AND, OR und NOT miteinander verknüpft. Beispielsweise kann der Satz „wenn A oder B, dann C“ wie folgt dargestellt werden:



Es sollte offensichtlich sein, dass Informationen auf diese Weise sehr differenziert dargestellt werden können, und diese Methode wird in dieser Hinsicht als eine der besten betrachtet. Es sollte jedoch ebenso offensichtlich sein, dass diese Methode ganz und gar nicht einfach ist. Sie wird als eine der schwierigsten angesehen.

Die Möglichkeit, Fehler zu finden, ist jedoch überwältigend, noch mehr als bei Flowcharts, da Fehler analytisch von einer logischen Bedingung abgeleitet werden können – eine detaillierte Erläuterung des Prozesses finden Sie im Abschnitt „Beobachtbare Fehler in Logik“. Insbesondere kann bewiesen werden, dass alle möglichen Fehler mit dieser Methode entdeckt werden können. Daher ist diese Methode bei Weitem die beste, um beobachtbare Fehler zu finden; sie ist jedoch im Vergleich zu Flowcharts sehr schwierig, was leider ihrer Akzeptanz schadet. Insbesondere liegt die Schwierigkeit in dem Versuch, Anforderungen im Hinblick auf Reihenfolgen zu integrieren (im Gegensatz zu Flowcharts, die für diesen Zweck ideal geeignet sind, auch wenn sie wiederum Anforderungen, die nicht von einer Reihenfolge abhängen, nicht gut integrieren können).

Bewertung von Ursache-Wirkungs-Diagrammen auf einer Skala von 1 bis 10:

Eingabeinformationen			Ausgabeinformationen		
Fähigkeit	Leichtigkeit	Anwendbarkeit	Anzahl Testfälle	Erkennbare Fehler	Abdeckung
10	5	8	10	10	10

Abschnitt 11

Relativer Vergleich

Die folgende Tabelle zeigt einen Vergleich der allgemeinen Bewertungen aller oben gezeigten Testfall-Designmethoden – alle Bewertungen liegen zwischen 1 und 10 und wurden so objektiv wie möglich anhand der folgenden oben erläuterten Kriterien bewertet:

- **Integrationsfähigkeit**: Dies ist der Umfang quantitativer Information zu der Anwendung, der in diese Methode integriert werden kann.
- **Leichtigkeit der Integration**: gibt an, wie leicht die Integration der Informationen ist.
- **Anwendbarkeit**: gibt an, wie viele Szenarien mit dieser Methode sinnvollerweise integriert werden können.
- **Anzahl Testfälle**: die relative Anzahl generierter Testfälle (1 – zu wenige/zu viele, 10 – optimal)
- **Erkennbare Fehler**: die relative Anzahl der Fehler, die gefunden werden können
- **Abdeckung**: die relative funktionale Abdeckung, die erreicht werden kann

HINWEIS: Enthalten sind auch die Klassen der formalen Modelle, die in einer gemeinsamen Kategorie enthalten sind – die Bewertungen sind als Spektrum angegeben, da einige hilfreicher, weniger anwendbar usw. sind als andere.

Methode	Eingabeinformationen			Ausgabeinformationen		
	Fähigkeit	Leichtigkeit	Anwendbarkeit	Anzahl Testfälle	Erkennbare Fehler	Abdeckung
Kombinatorische Methoden	1	9	5	2	1	1
Multiplikative Methoden	3	7	1	5	3	3
Formale Modelle (allgemein)	**7-10**	**5-10**	**5-10**	**5-10**	**5-10**	**5-10**
Flowcharts	9	9	9	9	9	9
Ursache-Wirkungs-Diagramme	10	5	8	10	10	10

Abschnitt 12

Die Vorteile von CA Technologies

CA Technologies (NASDAQ: CA) bietet Lösungen für das IT-Management, mit denen Kunden komplexe IT-Umgebungen managen und schützen können, um agile Business Services zu unterstützen. Unternehmen nutzen Software und Software-as-a-Service-Lösungen von CA Technologies, um Innovationen zu beschleunigen, ihre Infrastruktur zu transformieren und Daten und Identitäten vom Rechenzentrum bis zur Cloud zu schützen. Wir haben es uns zum Ziel gesetzt, dass durch Einsatz der Produkte von CA Technologies alle unsere Kunden die gewünschten Geschäftsergebnisse erreichen und den erwarteten geschäftlichen Nutzen erzielen. Weitere Informationen zu unseren Customer Success-Programmen finden Sie unter **ca.com/de/customer-success**. Weitere Informationen zu CA Technologies finden Sie unter **ca.com/de**.

Kontaktieren Sie CA Technologies unter ca.com/de.

CA Technologies (NASDAQ: CA) entwickelt Software, die Unternehmen bei der Umstellung auf die Application Economy unterstützt. Software steht in allen Branchen und in allen Unternehmen im Mittelpunkt. Ob Planung, Entwicklung, Management oder Security – CA Technologies arbeitet weltweit mit Unternehmen zusammen, um die Art, wie wir leben, Transaktionen abwickeln und kommunizieren, in mobilen, privaten und öffentlichen Cloud-Umgebungen oder in verteilten Systemen und Mainframe-Umgebungen neu zu gestalten. Weitere Informationen finden Sie unter **ca.com/de**.

CS200-154749