WHITE PAPER | 2015 年 9 月

テストの批評

ca technologies

目次

リンゴとナシ	**3**
客観性の必要性	3
ビジネス・チャンス	3
目的	3
テストの理念	**4**
情報のトランスフォーメーション	4
情報の測定可能性と不確実性	4
メタ数学とメタ開発	**6**
テスト・ケース設計技法の批評	**7**
ロジック内の観測可能な不具合	**7**
間に合わせの手段 - 組み合わせ技法	**8**
過度な専門化：反復カバー技法	**9**
形式モデル - 概要	**10**
要件とシステムの視覚化 - フローチャート	**11**
本格的なロジック仕様 - 原因結果グラフ	**12**
相対比較	**13**
CA Technologies のメリット	**14**

セクション 1

リンゴとナシ

客観性の必要性

さまざまなテスト・ケース設計技法の比較分析の必要性が生じ、このホワイトペーパーが執筆されました。分析を進めるにつれて、使用されているすべてのテスト・ケース設計技法に適用できる客観的かつ定量的な基準がないことが、すぐに明らかになりました。主観的かつ定性的な処理は数多く存在しますが、そのほとんどはベンダなどのマーケティングとしての役割を果たしています。このような処理は、その優劣を証明できないことに問題があります。基本原理から証明されることがなく、すべてが具体例とデータに基づいて行われます。つまり、これらの比較は帰納的に行われるのです。システムの進化という観点から見ると、帰納的分析にできることには限度があります。ある時点で、進捗の演繹的な証明が必要になります。

ビジネス・チャンス

このような演繹的基準に到達するためには、主観的分析を許容する基本的フレームワークから生じたすべての主観と分析結果を破棄する必要がありました。このステップは、哲学的にも数学的にも根本的な突破口となりました。基本原理（またはいわゆる公理的フレームワーク）に基づくフレームワークを構築することにより、データに基づく事後の証明が不要になったため、両方の分野が急速に前進できました。事後検証は現在広く認められている方法であるため、テスト担当者は事後検証の愚かしさに気付くのに最適な立場にあります。アイデアを最初からテストする代わりに実装中および実装後にテストすることが許容されていますが、これは土台の強度を検証せずにビルの強度をテストするようなものです。

目的

このホワイトペーパーの目的は、客観的フレームワークを確立することです。このフレームワークは次のような関連する 3 つの中心的概念に分けられます。

1. 測定可能と測定不能に分けられる汎用言語としての情報。 ソフトウェア開発ライフサイクル（SDLC）のすべての段階が、 それぞれ形態が異なる情報であると見なされます。
2. 情報エントロピーとしてモデル化された不確実性 ： 観測不可能な不具合とテスト不可能なソフトウェアがこのカテゴリに属します。
3. 情報のトランスフォーメーション：アイデアから設計へ、そして設計から実装へ。SDLC は多様な形態の情報の集まりであると考えられるため、間にある段階は情報のトランスフォーメーションとしてモデル化されます。ほとんどの処理では、これらはチューリング・マシンと見なされます。

分析を進めるにつれて、使用されているすべてのテスト・ケース設計技法に適用できる客観的かつ定量的な基準がないことが、すぐに明らかになりました。

これらのアイデアのほとんどは量子力学の分野に端を発するものであり、単にきわめて微細なレベルでの情報のモデル化であると考えることができます。このホワイトペーパー全体を通じて観測の尺度という概念が広範に使用されますが、実際に変化するのは詳細のみであり、異なる尺度で見てもシステムはやはりシステムだからです。たとえば、ビジネス・アナリストの間では、システムの要件を複数の異なる尺度で構築することが一般的です。まず高レベルで、次に少し低いレベルで構築しますが、正確な実装の詳細は最低レベルで提供されます。別の一般的な例として、開発者の間では、一体化して機能するようにコネクタで接続した小さなシステムが集合として大規模なシステムを実装することもあります。要件も実装も情報であり、その両方とも異なる尺度で見ることができるため、情報の処理に尺度を導入することは理にかなっています。

セクション 2

テストの理念

客観性の必要性を立証したら、次のステップは客観的分析にとって必要な公理を実際に立証することです。前述のように、公理は情報の量子力学的概念に基づいています。抽象概念としての情報は、何かの説明または実行可能な指示の集合として考える必要があります。説明的な情報は非常に簡単に理解できます。何らかのエンティティまたはオブジェクトに関する情報は、そのエンティティまたはオブジェクトを説明する手段です。たとえば、あるオブジェクトに関して、その美的特性を説明するために形容詞を使用したり、そのオブジェクトがどのように機能するかを説明するために動詞を使用します。一方、アクティブな情報は、あるオブジェクトを構築する方法を説明する指示で構成されます。たとえば、家具の部品を組み立てる方法に関する指示は、アクティブな情報と考えられます。

情報のトランスフォーメーション

この 2 つの形態の区別はやや微妙ですが、ここで論じている情報処理にとって非常に重要です。要件と設計文書はシステムまたはソフトウェアを組み立てる方法に関する指示と考えられるため、ここではアクティブな情報についてのみ論じていくからです。システムとソフトウェアはそれ自体、データの処理方法に関する指示の集合として抽象化できるため、データと同様に情報と見なす必要があります。これは、ここで使用する最も基本的な概念の 1 つであるため、この概念を完全に理解することがきわめて重要です。これにより、最も重要な概念すなわち情報のトランスフォーメーションへと到達できます。つまり、これらはある形態の情報から別の形態の情報を出力するトランスフォーメーションなのです。その最も簡単な例は開発者によるトランスフォーメーションです。開発者は、要件または設計という形態の情報をソフトウェアやシステムへと変化させます。

したがって、SDLC 全体を、トランスフォーメーションによって連結されたさまざまな形態の情報のチェーンであると考えることができます。要件から最終製品に至るまでの間にあるすべてのものを、何らかの形態をとった情報と見なすことができます。

情報の測定可能性と不確実性

しかし、ここまでは、品質という概念を考慮してきませんでした。品質については、2 つの関連性のある概念、情報の測定可能性と情報に関連した不確実性を想起する必要があります。つまり、情報の測定可能性は、実際に観測できる情報の量の尺度です。情報は、我々がそれを捕捉できる能力とはまったく無関係に存在しており、測定不能な情報は基本的には我々の役に立ちません。ただし、少なくともこの測定不能な情報の量がどれほど多いかについて考えることは重要です。メタ情報（または二次情報）については別のセクションで取り上げますが、ここでは一次情報について考えていきます。不確実性は、情報の測定可能性に関連した概念です。ここでは、これを情報エントロピーと呼びます。その定義は量子力学における解釈をそのまま取り入れたものであり、異なる情報出力へとつながります。一次と二次の区別の概念化を助けるために、データとメタデータの区別を理解する必要があります。メタデータはデータのプロパティを説明するために存在しますが、ここで言うデータとメタデータはそれぞれ一次情報と二次情報の一例です。メタ情報については、「メタ数学とメタ開発」のセクションで詳しく定義します。

これは実際には、要件の曖昧さの一般化です。曖昧さは指示の不確実性につながり、複数の解釈があると出力（またはそれらの要件の実装）が複数生じる可能性があります。テスト担当者と開発者が同じ要件の 2 つの異なる解釈をそれぞれ選択すると、ほぼ確実にテスト・ケースが実装を反映しなくなります。

理念について論じる前に明らかにしておく必要がある定義セットがもう 1 つあります。すなわち、次の 2 種類のエントロピーを区別する必要があります。

- 知識エントロピー - これは省略という意味で隠された情報であり、情報がそこにないために失われた自由度です。現実の世界では、その結果、トランスフォーメーション・プロセス中に情報を「作り出す」必要が生じます。たとえば開発者は、ときどき要件が曖昧すぎたり不完全な場合に、そのギャップを埋める必要に迫られることがあります。これは、原理上は測定可能な量です。実際には、測定には大きな労力が必要です。開発者が推測するにしても、これらは常に生成コードとして現れます。これは、トランスフォーメーションが可逆的であることが何を意味するかを示す一例です。
- システム・エントロピー - これはシステムに内在するエントロピーであり、測定できません。これは、量子力学におけるハイゼンベルグの不確定性原理に相当します。現実の世界では、その原因は指示の集合 / 公理体系が一貫性と完全性を両方とも備えることはできないという事実にあります。そのため、これは測定不能な量となりますが、相対的集合サイズという概念を使用すれば、少なくとも存在するシステム・エントロピーの量を正確に数量化することはできます。測定不能な集合の集まりは、直観には反しますが、測定可能だからです。

テストの基本理念は、ここまでの論考で確立した情報理論フレームワークの直接的な結果である 6 つの原理であり、次のように要約できます。

1. すべてをテストすることはできません。システム内には常にシステム・エントロピーが存在します。これは、公理体系のゲーデルの不完全性（またはチューリングの論証不能性）の直接的な結果です。
2. 何をテストできるかを知ることは常に可能です。知識エントロピーは、十分な情報があればなくすことができるため、測定可能になります。
3. 観測可能な不具合は、テスト計画の測定可能なエントロピーを表します。不具合（またはその不在）はソフトウェアの品質指標であるため、不具合を不確実性として扱うことは妥当です。特に、最良のテスト計画では、ほとんどの不具合が観測可能になります。これらの指標を組み合わせて 1 つの尺度にするために、通常は不具合に重大度が付加され、尺度は加重平均となります。ただし、任意の加重により、回避するべき主観性がある程度加わるため、重大度の付加は慎重に行う必要があります。
4. カバー率はテスト計画 / 戦略の忠実性の尺度です。つまり、テスト計画 / 戦略が要件を正確に反映していれば、観測可能な起こり得る不具合がすべて観測できるはずです。テスト計画 / 戦略に不備がある場合は、別の方法なら観測可能であったはずの数多くの不具合が観測不可能になります。
5. エントロピーが失われることはありません。これは、量子力学から直接来ています。現実の世界では、SDLC のいずれかのステップでエントロピーが生じた場合、たとえその後のすべての段階が 100% 忠実であったとしても、そのエントロピーを取り除くことは不可能であることを意味します。特に、ソフトウェア・システムの品質は、要件およびテスト戦略の品質に正比例します。ただし、これは機能するソフトウェアができないという意味ではありません。完璧なソフトウェアができないという意味です。
6. 単一のモデルですべての不具合を見つけることはできません。第 1 の原理により、すべてをテストすることはできませんが、複数のモデルを使用すればさまざまな不具合の集合を観測可能にできます。ただし、すべての不具合を見つけるには無数のモデルが必要になります。これはゲーデルの不完全性定理の直接的な結果です。

セクション 3

メタ数学とメタ開発

情報に関するこれまでの論考では、一次情報つまり純粋な情報についてのみ述べてきました。しかし、メタ情報または二次情報と呼ばれる「情報に関する情報」も存在します。その一例として、集計統計があります。たとえば、ある集団について、必ずしも個々人の身長を知らなくても、平均身長（および標準偏差）から多くの情報を探り出せます。

ここで、メタ情報がなぜ有用なのかを明らかにするために、ラムズフェルド・マトリクス（米国の元国防長官にちなんで名付けられました）に従って情報を次のように分類します。

- 既知の既知
- 未知の既知
- 既知の未知
- 未知の未知

これはグリッドとして表すことができます。ここまで取り上げてきた概念がすべてその中に含まれています。

	既知	未知
既知	情報	知識エントロピー 観測可能な不具合
未知	知識エントロピー 観測可能な不具合	システム・エントロピー

少なくともメタ情報は、たとえ情報がない場合でも、知らないことのサイズを測定するのに非常に役立ちます。メタ数学の原理が開発された目的は基本的に、数学的フレームワークの分析を通じて何が証明可能で何が証明不能かを明らかにすることです。数学的命題の大部分は、無限の濃度の存在に関する連続体仮説のように証明不能ですが、それによって数学コミュニティに問題が生じることはありません。証明可能な命題に集中すればよいからです。これと同じことがテストにも当てはまります。同様の方法で、テスト可能なもの（つまり既知の既知）に集中し、それ以外のものについてはメタ情報で済ませることができます。未知の未知（つまりシステム・エントロピー）は測定できないため、問題となります。しかし、他の 2 つの分類（既知の未知と未知の既知）は少なくとも測定可能であり、統計を導き出すことができるため、リスク分析にとって非常に有用です。

この比喩の核となる概念は（仮に「メタ開発」と呼ぶことにします）、テスト可能なソフトウェア（つまり既知の既知）に集中するべきであり、使用可能で観測可能な情報を最大化するためにすべての労力を注ぎ込むべきであって、それ以外のものに対してはメタ情報で十分であるという考えです。残念ながら、現在のテスト･ケース設計技法では、この批評で考察するように、使用可能な情報が最大限に活用されていません。チューリング･マシンの観点から、「ほとんどの」アルゴリズムはテスト不可能ですが（停止性問題によって明らかになったように）、少なくとも機能カバー率が最大 100% のテスト可能な構成が無限数存在します。この文脈において、ここでは無限集合を比較するために相対サイズという測度論的概念を使用します。テスト不可能なアルゴリズムの集合がアルゴリズムの総数のほとんど（つまり非可算無限数）を形成している一方で、テスト可能なアルゴリズムの集合は無視できるほど小さな部分集合を形成します（可算無限数：測度論において「測度 0 の集合」または「空集合」と見なされ、全体と比較すると事実上の無限小となります）。

セクション 4

テスト・ケース設計技法の批評

ここからは、前述の基本理念に基づいて、テスト・ケース設計技法を客観的に批評していきます。第 1 の理念を踏まえると、すべての不具合を見つけることは不可能です。しかし、第 2、第 3 および第 4 の理念から、テスト･ケース設計技法を評価するための次のような基準が導き出されます。

1. どれほどの量のアプリケーション情報をテスト・ケース設計プロセスにコード化できるか。
2. いくつの不具合を観測可能にできるか。
3. カバー率：観測可能な不具合の理論上の最大数との比率で、この技法でいくつの不具合が観測可能になるか。
4. 最適なカバー範囲に到達するために必要なテスト・ケースの相対数。

この 4 つの基準に基づいて、各テスト・ケース設計技法の以下の項目を評価し、10 点満点でスコアを付けます。

- **コード化できる量** – 技法にコード化できるアプリケーションに関する定量的な情報の量。(1 番の基準より導出)。
- **コード化の容易さ** – 情報のコード化がどの程度容易か。
- **適用性** – その技法を使用していくつのシナリオを合理的にコード化できるか。
- **テスト・ケースの数** – 生成されるテスト・ケースの相対数（1 – 少なすぎる / 多すぎる、10 – 最適)。(4 番の基準)。
- **検出可能な不具合** – 見つけられる不具合の相対数。(2 番の基準より導出)。
- **カバー率** – 達成可能な相対的機能カバー率。(3 番の基準より導出)。

スコアはすべて 1 から 10 までの点数であり、10 が「最高」です。

セクション 5

ロジック内の観測可能な不具合

先に進める前に、まず論理ステートメントに関連した観測可能な不具合のプロパティをいくつか決める必要があります。アプリケーション情報を使用するテストの最も正式なモデルは、基本的に論理ステートメントの分析に依存するため、不具合を探すならまず論理ステートメントから始めるのが適切です。

ここで、非常に単純な次の文について考えてみましょう。**IF A AND B THEN C**

この文は、原因（A and B）と結果（C）に分けられます。これを不具合の場合に当てはめると、原因の状態として、正しく実装された（OK)、0 で行き詰まった（0)、1 で行き詰まった（1）の 3 つが考えられます。しかし、この情報を踏まえて C に関連した不具合について考えるとき、何が起きるでしょうか。

C に関連した生じ得るすべての不具合を次の表に示します（すべての不具合が赤で示されています）。

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**	

これを見てわかるように、最後の 3 つの機能的変異（true/false の入力の集合）は、すべての生じ得る不具合を発見するのに十分です。最初の変異のみでは、それ以上の不具合は検出されません。したがって、生じ得るすべての不具合をこの技法で発見できると証明することが可能であり、n 個の入力を与えられた論理演算子について、この表からおおまかな数字を把握できます。

入力数	必要な機能的変異数	考えられる組み合わせ数	考えれる不具合数
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	$\sum_{x=1}^{n} \binom{n}{x} 2^x = 3^n - 1$

前述のように、不具合には観測可能なものと観測不可能なものの 2 種類があります。これらを区別する要因は、その結果が観測不可能であることではなく、根本原因が観測不可能であることです。これは、正しい理由に対して正しい答を確実に得るために極めて重要なことです。この文脈における可観測性は、不具合の識別、再現、または修正に必要な最小限の情報であると考えることができます。つまり、不具合がどこで発生したかを正確に特定できる必要があります。多くの点で、優れたテスト・ケース設計はこれを逆方向から実現すること、つまり、最小限の数のテスト・ケースを提供して観測可能なほとんどの不具合を検出することを目指します。前述のように、現実の世界では、観測可能な不具合はソフトウェアの知識エントロピーを表し、測定したりなくすことが可能ですが、観測不可能な不具合はシステム・エントロピーであるため、同じモデルを使用してテストすることはできません。したがって、テストのカバー率は、テスト・プロセスによってなくすことのできる知識エントロピーの量の尺度です。

セクション 6

間に合わせの手段 - 組み合わせ技法（全ペア・テストなど）

単純なカウンティング問題に由来する組み合わせ技法は、テスト・ケース設計の最も単純な形です。つまり、列のリストと各列の可能値を与えられ、2 つ組、3 つ組などのすべての可能な組み合わせがすばやく導出されて値が自動入力されます。これらの技法が持つ最も魅力的なプロパティは、いかなるアプリケーションの知識にも依存しないことです。ただし、これは最も致命的な弱点でもあります。したがって、定量的情報入力はデータのみに限られ、データ間の関係についてはいかなる入力もありません。前述の基本理念により、このようなテストによって明らかになるシステムに関する定量的情報も比較的少量です。これらは実際の機能ポイントにはマップされないため、アプリケーション・ロジックのどれほどの部分が実際にテストされるかを判断できません。ほとんどのシナリオでは、特有のテスト不足が起こりますが、重複する論理条件が原因で不必要にテスト過剰となる側面もあります。

組み合わせ技法には、上記の必然的帰結である第 2 の弱点があります。すなわち、無効なデータの組み合わせが頻繁に生成される結果として、実際の不具合ではなくデータが原因でテストが失敗する誤検出が発生します。これは、制約を導入することである程度軽減できます。そうすれば、提供される定量的入力が増えるほど、生成される定性的データがはるかに明確になります。第 3 に、予想される結果を自動的に因子分解することはできません。これらも定量的入力に依存するからです。優れた技法ではこのような情報をコード化できるため、組み合わせ技法はテスト・ケース設計にとってまったくお粗末な選択肢であると分析できます。

したがって、組み合わせ技法の使用は、機能以外の構成テストのみに限定するべきです。つまり、予想される結果が「機能する」のみであるシナリオに限定して使用するべきです。

組み合わせ技法のスコア（10 点満点）:

入力情報			出力情報		
量	容易さ	適応性	テスト・ケースの数	検出可能な不具合	カバー率
1	9	5	2	1	1

セクション 7

過度な専門化：反復カバー技法

組み合わせ技法の高度に専門化された応用であるこの技法は、テスト対象のシナリオが特定のデータ·エンティティのインスタンス数にのみ依存する場合に限って有用です。たとえば、顧客アカウント・データベースがあり、各顧客のすべてのアカウントが表示される画面があるとします。この場合、次のようなシナリオがテストされることになります。

1. 顧客がアカウントを持っていない
2. 顧客が単一のアカウントを持っている
3. 顧客が複数のアカウントを持っている

この技法は、ある種の機能ロジックを暗黙的に想定しているため、より単純な組み合わせ技法とは異なり、ある程度のアプリケーション情報がテスト・ケース設計に入力されます。実際、私はリレーショナル・データベースや階層構造（XML など）の中の親子関係を扱うときに、いつもこの技法を使用しています。一般的に、この技法は複雑なデータ構造に対して集約方法をテストするときに非常に役立ちます。これは、一部の集約機能には別個に処理する必要がある特殊な（または劣化した）ケースがあるという事実によるものです。次に例を示します。

- 数値の集合の平均をとる場合は、空集合のケースを特別に処理する必要があります。さもないと、ゼロ除算バグが発生します。
- 数値の集合の標準偏差をとる場合は、空集合と単一値集合の両方を特別に処理する必要があります。これらを別個に処理しないと、前者は負の結果につながり、後者はゼロ除算につながります。

このような技法の定性的な値は、考え得る入力の集合がすべて機能することの確認です（集合のサイズのみを与えられた場合）。これは、等価クラス・テストの特殊なケースであり、機能カバー率の整合性を保持しつつテスト・シナリオの総数を減らすために、一連の入力が互いに素のクラスにグループ分けされます。

反復カバー技法のスコア（10 点満点）：

入力情報			出力情報		
量	容易さ	適応性	テスト・ケースの数	検出可能な不具合	カバー率
3	7	1	5	3	3

セクション 8

形式モデル - 概要

この記事の残りの部分では、システムについて最も正確な定性的情報を提供する形式モデル技法について説明します。前述の理念に述べられているように、最大限の結果を得るためには大量の定量的情報が必要であるため、これは最も専門知識を必要とする技法です。これらの技法は、テスト・ケース設計プロセスにコード化できる情報の量という点でユニークです。

先に進む前に、形式モデルが何を意味するかについて簡単に説明します。基本的に、形式モデルは数学的に正確な要件の記述であり、それに対して、その記述に関する定性的情報を得るための操作を実行できます。最も重要な操作は次の 3 つです。

1. 一貫性チェック：要件のロジックが内部的に一貫していることを確認します。これにより、矛盾による曖昧さが排除されます。
2. 完全性チェック：要件のロジックが完全であることを確認します。これにより、欠落による曖昧さが排除されます。
3. 予想される結果の導出：一貫した完全なロジックにより、考え得るあらゆるシナリオについて予想どおりの結果が得られ、テスト・ケースの予想される結果が自動的に導出されるようになります。これは、テスト担当者の観点から見て形式モデルの最大の強みです。

形式主義それ自体が、前述の 3 つの操作の利点によって、形式主義をより「有用」にしています。この 3 つの操作はすべて、テスト対象のシステムについてきわめて詳細な定性的情報を提供します。実際、形式主義を早期に開発ライフサイクルに導入すれば、要件自体から定性的情報を抽出でき、強固な基盤の元にプロジェクトを開始できます。さらに、ほとんどの要件仕様技法は不活性的であり、設計段階以外（手動による介入以外）では役に立ちませんが、形式モデルは開発ライフサイクルのすべての段階に情報を提供できます。詳しくは説明しませんが、このような形式主義は、前世紀頃に達成されたほとんどの数学的および計算論的な進歩の基盤となっています。そのとき十分に有用であったのなら、我々にとっても十分有用であると考えるのは理にかなっています。

形式モデルにはあらゆる形とサイズがありますが、ここではテストに最も関係が深い 2 つ、フローチャートと原因結果グラフについて説明します。これらは、正準な形式と考えることができます。形式モデルはすべて、この 2 つにある程度似かよっています。

セクション 9

要件とシステムの視覚化 - フローチャート

全般的に、フローチャートは広く理解されている要件仕様の形です。フローチャートは多くの点で「テキストの壁」アプローチより優れていますが、何より有用なのは、要件からテスト・ケースを体系的に導出できる能力です（自動アルゴリズムによって達成できます）。これは一見、思ったより簡単ですが、伝えるのが難しい考え方です。

テスト・ケースを導出するには、単にフローチャート内の可能なパスを評価します。基本的に、必要なのはそれだけです（以前はグラフ・ホモトピー分析と呼ばれていました）。これが役に立つのは、テスト・ケースの数を減らすために最適化技法が適用されるときですが、論理構造の利点により機能カバー率が保持されます。本質的に、考慮する必要があるのは、個々の意思決定ブロックのみです。各ブロック（または演算子）がそこに出入りする直接的パスについて完全にテストされたら、流れ全体が完全にテストされたと見なされます。これは何よりも、「数学的」説明なしでは伝えることが非常に困難な（そしてストレスがたまる）ことです。

要するに、テストされる同じローカル条件の集合が原因で（つまり演算子レベルで）一部のパスが重複するため、これが成立するのです。このような重複は、他の（遠くの、つまりグローバルな）テスト対象を因子分解しようとしたときに発生しますが、これは、重複する機能的変異が互いに「相殺する」という単純な事実を示しています。これは、特定の事例について説明するより一般論として説明する方が簡単である事柄の 1 つです（理論数学の大部分と同様に）。

フローチャートによって明確な要件を指定できるため、コード化できる定量的情報の量が増大し、前述の 2 つの技法より桁違いに多くなります。これも、すべての形式モデルに当てはまることであり、異なるのはその程度だけです。さらに、演算子レベルのテストという概念により、検出できる不具合の量も大幅に増大するため、モデルから得られる定性的情報の量もそれに応じて膨大になります。

フローチャートのスコア（10 点満点）:

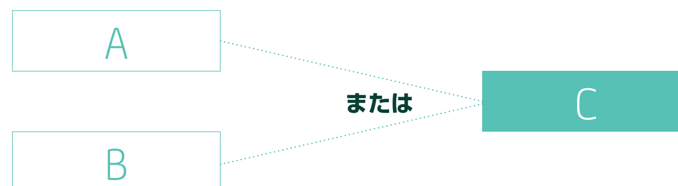
入力情報			出力情報		
量	容易さ	適応性	テスト・ケースの数	検出可能な不具合	カバー率
9	9	9	9	9	9

セクション 10

本格的なロジック仕様 - 原因結果グラフ

形式モデルのもう 1 つの例ですが、この場合は論理ステートメントが原因と結果としてモデル化され、それらの間の関係が完全に指定されます。この技法は、フローチャートと同様に、文章による要件が明確に指定できる機能を備えているため、特に分析されてきました。

ブロックがつなぎ合わされているという考え方はフローチャートと同じですが、指定されるのが因果関係である点が異なります。さらに、ブロックは AND、OR、NOT などの論理演算子を使用してつながれます。たとえば、「if A or B then C」という文は、次のようにコード化できます。



この方法では情報を非常にきめ細かくコード化できることは明らかであり、この点において最善であると見なされています。ただし、この技法が簡単ではないことも明らかであり、最も難しいと見なされています。

しかし、不具合を論理条件から分析的に導出できるため、不具合を見つける能力は圧倒的であり、フローチャートより優れています。そのプロセスの詳細については、「ロジック内の観測可能な不具合」に関するセクションを参照してください。特に、この技法によってすべての考え得る不具合を発見できると証明することが可能です。結果として、観測可能な不具合の検出という観点からは、この技法が最高に優れていますが、フローチャートと比べて非常に難しいため、残念なことにその採用が妨げられてきました。特に、その難しさは順序に固有の要件のコード化を試みることにあります（その目的に適したフローチャートとは対照的ですが、フローチャートは順序に関係のない要件を適切にコード化することはできません）。

原因結果グラフのスコア（10 点満点）:

入力情報			出力情報		
量	容易さ	適応性	テスト・ケースの数	検出可能な不具合	カバー率
10	5	8	10	10	10

セクション 11

相対比較

次の表は、これまで説明してきたすべてのテスト・ケース設計技法の全般的なスコアの比較です。スコアはすべて、前出のセクションで説明した次の基準に従って、できる限り客観的に 1 から 10 までの数値で示されます。

- **コード化できる量** – 技法にコード化できるアプリケーションに関する定量的な情報の量。
- **コード化の容易さ** – 情報のコード化がどの程度容易か。
- **適用性** – その技法を使用していくつのシナリオを合理的にコード化できるか。
- **テスト・ケースの数** – 生成されるテスト・ケースの相対数（1 – 少なすぎる / 多すぎる、10 – 最適）。
- **検出可能な不具合** – 見つけられる不具合の相対数。
- **カバー率** – 達成可能な相対的機能カバー率。

注 : 1 つのカテゴリに属する形式モデルのクラスも含まれています。スコアは範囲として示されていますが、これは一部のモデルは他のモデルより能力が高かったり適用性が低かったりするためです。

技法	入力情報			出力情報		
	量	容易さ	適応性	テスト・ケースの数	検出可能な不具合	カバー率
組み合わせ	1	9	5	2	1	1
反復	3	7	1	5	3	3
形式モデル（概要）	**7-10**	**5-10**	**5-10**	**5-10**	**5-10**	**5-10**
フローチャート	9	9	9	9	9	9
原因結果	10	5	8	10	10	10

セクション 12

CA Technologies のメリット

CA Technologies（NASDAQ:CA）は、複雑な IT 環境の管理と保護に役立つ IT 管理ソリューションを提供し、アジャイル開発のビジネス・サービスを支援します。CA Technologies のソフトウェアと SaaS ソリューションを活用することで、データセンタからクラウドに至るまで革新を加速し、インフラストラクチャを変革し、データとアイデンティティを保護できます。CA Technologies はそのテクノロジにより、お客様が必要な成果と期待どおりのビジネス・バリューを実現できるようにします。お客様を成功に導くプログラムの詳細については、**ca.com/jp/customer-success** をご覧ください。CA Technologies の詳細については、**ca.com/jp** を参照してください。

ca.com/jp/でCA Technologiesにアクセスしてください

CA Technologies（NASDAQ:CA）は、企業の変革を推進するソフトウェアを作成し、アプリケーション･エコノミーにおいて企業がビジネス・チャンスを獲得できるよう支援します。ソフトウェアはあらゆる業界であらゆるビジネスの中核を担っています。プランニングから開発、管理、セキュリティまで、CA は世界中の企業と協力し、モバイル、プライベート・クラウドやパブリック・クラウド、分散環境、メインフレーム環境にわたって、人々の生活やビジネス、コミュニケーションの方法に変化をもたらしています。詳細については **ca.com/jp** をご覧ください。

Copyright © 2015 CA, Inc. All rights reserved. 本書に記載のすべての商標は、該当する各社に帰属します。この文書は保証を含むものではなく、情報の提供のみを目的としています。機能に関する記述は、本書に記載された顧客に固有のものである可能性があり、実際の製品性能は異なる場合があります。

CS200-154749