ソフトウェアテスト技法の習得と実務への適用

- 闇雲なテストからの脱却 -

An approach to learn and practice software testing techniques

- Departure from the test on personal experience -

主査　：奥村　有紀子　（有限会社デバッグ工学研究所）
副主査：秋山　浩一　（富士ゼロックス株式会社）
副主査：堀田　文明　（有限会社デバッグ工学研究所）
研究員：佐藤　佳延　（株式会社キヤノン IT ソリューションズ）
古庄　一磨　（キヤノンファインテック株式会社）
南雲　博人　（株式会社ニコンシステム）
豊福　暁久　（株式会社リンクレア）
青木　滋　（NTT データシステム技術株式会社）

研究概要

ソフトウェア開発に対する市場ニーズは、納期やコストだけではない。市場投入後のソフトウェアの重篤な不具合により、社会的に大きな問題に発展する場合や、その不具合に対する改修費用が莫大な金額になり企業経営に打撃を与える場合があるため、品質についても強く求められている。

しかしソフトウェア開発は、機械や電気または電子回路設計と比較すると開発自由度が高くスピードが求められる側面や、開発者自身でソフトウェアがどのように使用されてどのような振る舞いをするか、ソフトウェアテストで全て把握するのは現実的に困難な状況にあるためソフトウェアテストでの品質の確保は漠然としており難解なものになっている。

本論文ではソフトウェア品質の確保に取り組む足がかりとすべく、ソフトウェアテスト初心者が実践できる基礎的なソフトウェアテスト技法の習得に重点を置き、実際のソフトウェアテスト業務への適用を試みた。

Abstract

High quality is the serious request for software development from customers, in addition to the delivery time and cost. When the software expose a serious defect, it may cause major social issue. Repair costs for defects will be a huge amount of money and also may hurt corporate management. Therefore, for software development, quality must be much important.

In software development, a high degree of free hand is exist, and the development speed is required, however. For this reason, ensuring software quality tends to be a vague as a theme, and to become very difficult.

In this paper, we take up a test that plays an important role in ensuring the quality of software. And focuses on the discussion of learning a relatively common testing techniques that beginners can practice in software testing, and application to actual test operations.

1.　はじめに

テストによるソフトウェア品質の確保という難解なテーマに対し、メンバーそれぞれが持つ悩みや問題点を持ち寄りディスカッションした結果、論理的根拠を持たない独自の方法でのソフトウェアテストや、担当者のスキルや経験に依存しているためソフトウェアテストの品質にばらつきがある等、課題が浮き彫りとなった。なぜこのような事態になっているかさらに考察した結果、特にソフトウェアに実装された機能の組合せテストにおいて効率的なテストケースの抽出方法や、論理的根拠に基づいたパラメータの組合せ方法を知らないことが一因であると考えた。そこで、基礎的なソフトウェアテスト技法の習得に重

点を置いて実際のソフトウェアテスト業務へ適用を試みた。

2. 研究課題

まず、なぜソフトウェアテスト技法が必要かを整理する。アドホックテストやランダムテストのように仕様に対して不明確な目的で非計画的なテストを実施した場合、どこをどのように実施したか、どこまで確認したか把握しにくい。その結果として同じ範囲を何度もテストすることになる無駄や、テストで確認していない範囲が出てくるなどのムラが発生する。また、担当者によりテストの実施方法方が異なることでテスト品質の確保ができない等の課題がある。

2.1 ソフトウェアテスト技法の学習と課題

ソフトウェアテスト技法の適用メリットは次の 2 点である。

- ◆ 目的を明確にしたソフトウェアテストの計画を立案しやすい
- ◆ どのように、どこまでテストしたか把握しやすい

本分科会では、前半の 6 回＋臨時会 1 回を使って主査・副主査・鈴木三紀夫氏、池田暁氏から、講義/演習形式で、基礎的なソフトウェアテスト技法にはどのようなものがあり、どのように適用できるものなのかを重点的に習得した。

本論文では、習得の前後でソフトウェアテストへの意識の違いについて述べる。また、習得したソフトウェアテスト技法を実際のソフトウェアテスト業務へ適用し効果を考察する。なお、業務への適用に際しては、組合せテストの技法である、デシジョンテーブル、All-Pair 法を選択した。

3. 習得したソフトウェアテスト技法

3.1 ソフトウェアテスト技法

講義ではソフトウェアテスト技法の習得に重点を置いた。表 1 に学習したソフトウェアテスト技法と概要を示す。

表 1．習得したソフトウェアテスト技法と概要

技法	概要
同値分割法	単体テストにて、入力値を同値クラスに分割し代表値を用いる技法
境界値分析法	単体テストにて、入力値に同値クラスの境界値を用いる技法
CFD 法 デシジョンテーブル	結合テストにて、複雑な内部仕様を流れ図で表しデシジョンテーブルで入力と出力の組合せを示す技法
直交表 HAYST 法	組合せテストにて、実験計画法で用いられる直交表の性質を利用し 2 因子間の組合せを網羅したテストケースを設計する技法
All-Pair 法	組合せテストにて、任意の 2 因子間の全水準の組合わせが 100%となるテストケースを設計する技法
状態遷移テスト	結合テストにて、テストすべき状態とイベントを表にしテストケースを設計する技法

3.1.1 同値分割法

同値分割法とは、同値または同値クラスと呼ぶ入力値を同一特性の部分集合に分割し、出力が仕様の機能と動作条件に合致しているかテストする技法。

[同値分割法の適用メリット]

入力可能な値を一つ一つテストすると、テストケースが多くなりすぎるため、プログラムの条件に影響する入力を部分集合することによりテストケースを削減する。

[同値分割法習得後の意識の違い]

・当たり前の技法として認知されているためか、丁寧な説明を受けたのは初めてであり、

同値分割をあらためて考える時間が取れたのは良かった。同値分割の重要性を強く認識した。
・あらためて自社のテストケースを確認してみると、処理の結果が無効になる同値クラスでの評価項目がない場合も散見された。ソフトウェアの評価者に限らず開発者も含めて必ず身につけておきたい技法である。

3.1.2 境界値分析法

境界値分析法とは同値と同値の境界値を使い、仕様の出力が機能と動作条件に合致しているかテストする技法。同値分割法との違いは同値クラス内の「代表値」を使うか「境界値」を使うかである。

[境界値分析法の適用メリット]

この技法の適用時のポイントとして、処理の結果が無効となる同値クラスに関しても上限、下限の境界値を使ってテストで確認することである。しかし、無効となる同値クラスは仕様に定義されていない場合がある。この技法を適用すると無効となる同値クラスの仕様を明確にできる場合がある。

[境界値分析法習得後の意識の違い]

・同値分割同様に当たり前の技法であるが、前提となる同値分割を正しく行わないまま適切でない境界値でテストをするなど、境界値分析を正しく理解していなかった。

3.1.3 CFD 法とデシジョンテーブル

CFD法（Cause Flow Diagram）により、複雑な内部仕様を流れ図で表し、デシジョンテーブルで入力と刺激（原因）、及び、対応する出力と処理（結果）の組合せを示す。テストケースの設計に利用でき、仕様からプログラムの条件判断ロジックを展開できる。

[CFD 法とデシジョンテーブルの適用メリット]

CFD で原因を同値分割することでテストケースを削減できる。結果は有効系と呼ぶ仕様に定められた処理を行うものと、無効系と呼ぶエラー処理等の有効系以外に明確に分割することでテストの優先順位をつけることができる。CFD からテストケースを抽出しデシジョンテーブルに展開することで条件の組合せが複雑な仕様を可視化できる。

[CFD 法とデシジョンテーブル習得後の意識の違い]

・デシジョンテーブルを作成することによって、複雑な条件の組合せのテストケースが作成されるだけではなく、各処理間の関係も見えるので仕様として記述されていなかったあいまいな部分も発見できるようになった。

3.1.4 直交表と HAYST 法

直交表とは整数の配列表で､任意の 2 列を選択した場合どの列を取っても同じ組合せが同じ数だけある表のことである。例として 2 水準の直交表を表 2 に示す。

表 2. 2 水準の直交表の例

L4	因子 A	因子 B	因子 C
テストケース 1	0	0	0
テストケース 2	0	1	1
テストケース 3	1	0	1
テストケース 4	1	1	0

任意の 2 列の数字並びは（0,0）（0,1）（1,0）（1,1）の 4 通りあり、この 4 通りの並びがどの 2 列でも必ず同数回ある。HAYST 法では、直交表の性質を利用して 2 つの因子間の組合せを網羅したテストケースを設計する技法である。大規模ソフトウェア（入力として同時に与える可能性のある機能数が 300 程度の組合せを持つソフトウェアテスト）へ適用できることと、禁則回避処理を持つ点に特徴がある[1]。

[直交表と HAYST 法の適用メリット]

機能と機能間に仕様上で関係性がないとされている場合の組合せに有効である。

任意の 2 機能間、任意の 3 機能間の因子・水準を直交表に割り付けることにより、効率的にテストケースを生成できる。L256 直交表のような多因子多水準に対応している。
[直交表と HAYST 法習得後の意識の違い]
・因子と水準さえ決まっていればあとは技法のルールに従ってテストケースを作成すればよい。しかし、因子・水準の選定に関してはテスト要求分析やテストアーキテクチャ設計を注意して行わないと無駄なテストケースが出てしまうので注意が必要。

3.1.5 All-Pair 法

任意の 2 因子間の全水準の組合せが 100%となるテストケースを作成する方法。
HAYST 法との違いは任意の 2 因子間の水準組合せが「同数回」出現するのに対して、All-Pair 法では「同数回」という条件を外すことである。
[All-Pair 法の適用メリット]
任意の 2 因子間の水準の組合せに着目し、比較的簡単にテストする組合せを作成できる。組合せ作成ツール等もあり、実作業への適用が容易である。
[All-Pair 法習得後の意識の違い]
・組合せテストの技法であり、長所・短所が理解できた。なお、All-Pair 法はツールがあり、比較的容易にできるので適用したいと考えた。

3.1.6 状態遷移テスト

同じ入力であっても内部変数等の状態によって出力が違う場合が多い。
そのような場合に、状態とイベントをマトリクスに表現した表を作成しテストを実施する。
[状態遷移テストの適用メリット]
複数の状態が遷移する様子を表現する状態遷移図を作成することにより仕様の不備や誤りを検出することができる。
[状態遷移テスト習得後の意識の違い]
・状態遷移図を作成し、そこからテストケースを作成するというやり方は従来から行っていたが、各状態に遷移しているか確認する程度のもので、状態ごとのパスに注目した評価はしていない。今後は実施していく必要がある。

3.2 テスト分析技法

3.2.1 3 色ボールペン＆マインドマップを用いたテスト設計

[習得した内容]
仕様書から「仕様の整理」、「テスト観点の抽出」を行う際に、テストベース（テスト設計時のベースとなるドキュメント）となる仕様書を読み解く際 3 色ボールペンを使って仕様を整理し、テスト観点の抽出を行う際に色分けしたマインドマップを作成することで、見落としがちな仕様やテスト観点を視覚的にも気付く事を演習により体験した。
[習得後の意識の違い]
・思考が可視化され、新たな気付きを得るには良い技法である。しかし、マインドマップの書き方によっては、発想が広がり難いものになってしまうため、書き方や議論の進め方にも経験が必要。
・マインドマップという言葉は知っていたが、テストケースの作成に利用することができると初めて知った。ケースを網羅するということではなく、全体像を視覚的に把握できるため、関連している要素や観点に思い至りやすいという利点がある。

4. ソフトウェアテスト技法の適用

3 章までの内容を踏まえ、2 名のメンバーがソフトウェア技法の実務への適用を試みた。

4.1 デシジョンテーブル

4.1.1 背景と目的

メンバーの 1 人は、出荷前の第三者評価を行う品質保証部門に所属しており、組込み系のソフトウェア評価を担当している。

そこで問題となっているのは 2 つ以上の機能が絡み合う場合の評価方法が不明確な点である。機能の組合せに対する評価において、テスト技法は導入しておらず経験やスキルに基づいて行われており、統一されたやりかたになっていない。そのため評価担当者ごとのばらつきが避けられず、評価漏れの発生によりバグが後工程や市場で発見されるケースがあとを絶たない。

評価漏れは次の事から発生していると考えられる。単機能を評価する場合は、機能に関係する動作条件や入力値の組合せとその出力値を確認すればよい。しかし、複数機能を評価する場合は、機能を掛け合わせた場合の優先順位や協同動作条件等を明確に抽出し、協調動作時の処理や条件、出力結果を整理し評価しなければならない。ただし、機能が増えるほど、動作条件や優先順位等が複雑化する。さらに評価方法が人により異なるため、評価担当者ごとにばらつきがでてしまい、結果として、テストケースに漏れがでてしまう。

この問題の原因は、「複数機能の評価方法が不明確」であり、結果として担当者ごとのばらつきが発生してしまっていると考えた。そこで原因を解消すべく、『複数機能の評価方法の明確化』という課題を設定した。

この課題を解決するために、研究会にて学習した、複数の判定条件の組合せと、それに対応する判定結果をまとめることができる『デシジョンテーブル』[1]を適用した。

4.1.2　デシジョンテーブルの適用

今回の事例は、現在開発中のプロジェクト出荷前評価に適用したものである。テストベースは機能仕様とした。

一つの出力に対して影響を及ぼす 6 機能間の優先順位と協調動作についてデシジョンテーブルを適用した。

デシジョンテーブルの適用は以下の手順で行った。

(1)仕様の整理と条件の洗い出し

(2)デシジョンテーブルを使って全組合せの書き出し

(3)評価への適用

(1)仕様の整理と条件の洗い出し

デシジョンテーブル作成の前準備として、仕様の整理を行った。仕様書から、各 6 機能に関係する入力値、動作状態や条件、処理内容、目的等を表（例：表 3）にまとめ、仕様書に記載がない場合は設計者に問い合わせ、各機能がどの機能と協調動作し優先されるのかを整理（例：表 4）した。また、仕様整理段階で抜け漏れがあると、バグの見落としにつながるので、関係者にレビューを依頼した。最後に整理した表から動作結果に影響する条件を洗い出した。

作成した表は､デシジョンテーブル作成時､各条件から処理を記入する際にも使用する。

表 3　各機能の仕様まとめ例

機能	入力条件	環境1	環境2	処理内容	目的	備考
機能1	全適応	全適応	全適応	処理内容1	目的1	機能2と同時設定不可
機能2	入力1、入力3があった場合	全適応	全適応	処理内容2	目的2	機能1と同時設定不可
機能3	全適応	全適応	高湿の場合	処理内容3	目的3	特になし
機能4	入力1、入力2、入力4あった場合	全適応	全適応	処理内容4	目的4	特になし
機能5	入力1、入力3があった場合	高温の場合	全適応	処理内容5	目的5	特になし
機能6	全適応	全適応	全適応	処理内容6	目的6	特になし

※入力は4種類あり、
選択できるのは一つ

表 4　優先順位まとめ例

	機能1	機能2	機能3	機能4	機能5	機能6
機能1		○	△	○	×	○
機能2	×		△	△	×	○
機能3	△	△		△	○	△
機能4	×	△	△		×	△
機能5	○	○	×	○		○
機能6	×	×	△	△	×	

○：横が優先
×：縦軸が優先
△：重複可能

(2) デシジョンテーブルを使って全組合せの書き出し

前項から条件は、6 機能（うち機能 1 と機能 2 は排他）、入力 4 種（選択できるのは一つ

のみ)、環境 1（低温と高温)、環境 2（低湿と高湿）とした。

条件指定部に選択した条件を書き出し、各条件全組合せを入力した。(例：表 3、選択した条件に「Y」を表記）今回の条件の場合、機能 1 と機能 2 が排他処理のため 3 通り、機能 3～6 までが設定有無で各 2 通りずつ、入力が 4 通り、環境 1 が低温と高温で 2 通り、環境 2 が低湿と高湿で 2 通り、結果として 3*2*2*2*2*4*2*2=768 組合せとなった。

次に、組合せた条件に対応した結果を、前項で作成した仕様のまとめ表や優先順位表を用いて結果指定部に記入していく。

また、結果指定部は出力（数値等）を記載するのではなく、各条件下で最終的に動作する機能とした。(例：表 5、No1 の条件の場合、機能 1 と機能 3 が協調動作する。No2 の条件の場合、機能 1 が優先され動作する。)

結果として、他プロジェクトの同条件の機能協調動作評価のテストケースは、約 100 項目前後であったのに対し、今回の適用事例では項目数が 768 項目となった。

表 5　全組合せの書き出し例

条件指定部　　結果指定部

No	条件														処理							
	機能						入力				環境1		環境2		機能1	機能2	機能3	機能4	機能5	機能6	通常動作	設定不可
	機能1	機能2	機能3	機能4	機能5	機能6	入力1	入力2	入力3	入力4	低温	高温	低湿	高温								
1	Y		Y				Y					Y		Y	Y		Y					
2	Y			Y			Y					Y		Y	Y							
3	Y				Y		Y					Y		Y					Y			
4	Y					Y	Y					Y		Y	Y							
5	Y		Y	Y			Y					Y		Y	Y		Y					
6	Y			Y	Y		Y					Y		Y					Y			
7	Y				Y	Y	Y					Y		Y					Y			
8	Y		Y			Y	Y					Y		Y	Y		Y					
⋮																						
767			Y		Y	Y				Y	Y		Y									Y
768			Y	Y	Y	Y				Y	Y		Y									Y

(3) 評価への適用

前項で作成したデシジョンテーブルは処理結果の動作機能が散在しているので、評価をスムーズに進めヒューマンエラーを防止するために、処理結果が同じ内容の項目をまとめ整理した。(例：表 6)

表 6　項目の整理例

No	条件														処理						
	機能						入力				環境1		環境2		機能①	機能②	機能③	機能④	機能⑤	機能⑥	通常動作
	機能①	機能②	機能③	機能④	機能⑤	機能⑥	入力①	入力②	入力③	入力④	環境①	環境②	環境①	環境②							
1	Y			Y			Y				Y		Y		Y						
2	Y				Y		Y				Y		Y		Y						
3	Y					Y	Y				Y		Y		Y						
4	Y			Y	Y	Y	Y				Y		Y		Y						
5		Y		Y			Y				Y		Y			Y					
6		Y			Y		Y				Y		Y			Y					
7		Y				Y	Y				Y		Y			Y					
8		Y		Y	Y	Y	Y				Y		Y			Y					

処理と組合せが同じものをまとめる

4.1.3　結果と考察

複数機能の評価方法の明確化という課題に対して、今回デシジョンテーブルを適用し、

一定のルールに従い複数機能の組合せに対するテストケースが作成できた。また、仕様の整理段階で、仕様の情報が網羅的になっているか整理したことにより、仕様の抜けや不明確な部分にも対処した組合せのテストケースを作成することができた。

また、他プロジェクトの機能協同評価のテストケースは、100 項目前後であったのに対して、今回の適用事例では、768 項目と大幅にテストケースが増えた。他プロジェクトではテスト技法等適用しておらず、経験やスキルによって作成されており、内容は 6 機能中の 2 機能間の総当たりと、機能が最大選択された場合のテストケースしかないとが項目数の少ない原因である。しかし、今回の適用事例は、項目数は増大したものの、仕様の整理とレビューからデシジョンテーブルを適用し、以前と比べ漏れなくかつ分かりやすく表現できたと考える。

4.1.4 所感

今回デシジョンテーブルを作成して感じたことは、一度に広範囲の機能の組合せをデシジョンテーブル上に表現しようとすると、条件が多数でてきてしまい、項目数が増え考えにくくなることである。

そのため、デシジョンテーブルを作成する前の仕様整理により、関係が強い機能の組合せ範囲を適切に抽出することが重要だと実感した。

今後の課題としては 2 つ。1 つ目は仕様整理段階での担当者のばらつきを無くすため、仕様整理の実施とその方法についてルール化すること。2 つ目はデシジョンテーブルで作成したテストケースの合理的削減であり、今後検討する。

4.2. ALL-Pair 法

4.2.1 背景と目的

All-Pair 法を試行したメンバーは開発部門とは異なる管理部門に所属しており、その役割は、開発部門が作成したエンタープライズアプリケーションをユーザー視点でテストを行い、製品の品質を確認することである。最近は、開発部門と所属部門との間の役割見直しにより、開発部門に役立つテストの知識やテスト技法を習得し、社内展開する事が望まれている。また、経験豊富な開発者はテスト技法を用いて漏れの少ないテストを行っているが、新人などテストスキルが充分でない開発者は、バグ修正時に影響範囲の見極め誤りにより、機能間の不整合を発生させてしまっている。

この問題の対策として、管理部門としてはツールの利用や標準ガイド作成により、できるだけ属人性を排除する方法を検討している。

そこで、研究会で学習したテスト技法の中から、少ないテストケースで 2 項目間の組合せ網羅が確保できる「All-Pair 法」を試行してみる事とした。なお、All-Pair 法のツールとしては、研究会では PictMaster と Qumias について学習した。試行にあたっては、PICT よりも高速で動作するアルゴリズムを採用している Qumias を用いた[3]。

4.2.2 All-Pair 法の試行

All-Pair 法の試行は、ツール（Qumias）を用いて、下記手順で行った。

① 外部設計書に記載されている設定項目や項目内容をもとに、因子と水準を洗い出し、一覧にする

- 水準 2 個の因子：18 個
- 水準 3 個の因子：2 個
- 水準 4 個の因子：1 個

② 一覧にした因子・水準をツールに読み込む

③ ツールの GUI で網羅度の設定（全体/特定項目)、6 件の制約条件の設定、試行回数の設定を行う

・網羅度の設定：2 項目間の網羅（デフォルト値）

・制約条件の設定：「if A=1 then B=2 and C=3」といった制約条件 6 種類

・試行回数の設定：設定なし（デフォルト値）

④ 入力条件を元にテストケースを作成

4.2.3 結果

上記手順で作成した結果、16 件のテストケースとなった。また、試行回数の設定を変更して、最少テストケース数となる設定で再作成した場合は 14 件のテストケースとなった。

制約条件などの各設定がマニュアルを読まなくても直観的に出来たこともあり、本ツールを初めて使った筆者でも比較的簡単にテストケースを作成出来た。

4.2.4 所感

今回の試行で、ツールが直感的に操作可能なことを体験出来たのは、今後社内展開するうえで良かったが、技法そのものの習得は十分に行えていない。特に因子や水準は外部設計書ベースでの抽出であり、機能間の干渉度合などの関係性を分析していないため、効果的な抽出が出来たとは言い難い。今後の課題としては、FV 表やラルフチャート等を利用した、因子や水準の効果的な抽出方法や、「3 項目間以上の組合せテスト」を行う必要が有るか無いかを見極める方法など、さらなる技法の学習を行う必要がある[1]。また、他ツール（PictMaster）でも試行し、実業務に最適なツールを選定する必要がある。

5. 最後に

5.1 考察

本論文での活動の目的は、ソフトウェアテスト技法を習得し、実業務へ適用することであった。基礎的なソフトウェアテスト技法の取得については、講義を通じて、全参加メンバーが達成できた。さらに、なぜソフトウェアテスト技法が必要なのかを考えるよい機会となった。

一方で、本論文参加メンバーのうち 2 人が習得した技法を実業務へ適用してみると、例題を解くようにはいかなかった。テストの目的や状況に合せてソフトウェアテスト技法をどう適用するか、テスト結果を分析し、どう反映するか等の課題が見つかった。その原因を下記に挙げる。

- ◆ 実業務で扱う因子と水準の数が多く、適切な組合せでテストケースを生成するのに時間を要した。
- ◆ テスト対象の規模が思っていたよりも大きく、仕様から適切にデシジョンテーブルを作成することや、All-Pair 法を適用することが難しかった。

5.2 反省点

今回は基礎的なソフトウェアテスト技法を習得したがどのように実業務に展開していくのか、考えが足りていなかった。そのため習得した技法が、どの程度有効か証明する数値の取得には至らなかった。しかし、基礎的なソフトウェアテスト技法を組み合わせることにより現状より効率的で効果の高いテストができることは理解できたので大きな収穫と言える。

参考文献

[1] 秋山浩一ほか：“ソフトウェアテスト HAYST 法入門”，日科技連出版社（2007-7）
[2] リー・コープランド，宗雅彦（翻訳）：ソフトウェアのテスト技法，日経ＢＰ社,2005
[3] 特集ツール Qumias,<http://www.qbook.jp/qptool/special>,(2013/01/24 アクセス)