

## CFD 技法の評価と事例に基づくガイドの作成

The evaluations of the CFD test design technique and making guides based on its case study

### 分科会メンバー

主査：保田 勝通 [つくば国際大学]  
副主査：西 康晴 [電気通信大学] 高橋 寿一 [ソニー(株)]  
研究員：端野 宣行 [NTTコムウェア(株)] (リーダー)  
近藤 和彦 [東京エレクトロン ソフトウェア・テクノロジーズ(株)]  
斉藤 正純 [(株)アドイン研究所]  
小野寺哲哉 [日本ノーベル(株)]  
服部 祐二 [ブラザー工業(株)]

(敬称略)

## 1. 概要

---

近年、ソフトウェア開発の現場では、開発の短期化、前工程での遅延などのしわ寄せ等により合理的なテストを実施する必要性が増しているが、多くの開発者がテスト設計を、経験と勘により実施している場合が多い。この結果、テストケースが設計者個人の能力に依存し、テストケースに漏れや重複が発生し、非合理的なテスト設計が行われている状況が多く見られる。

本研究では、以上の問題を解決し最適なテストケースを設計するためにテスト技法としてCFD (Case Flow Diagram) 技法を選択し、評価及び試行を実施した。さらに、多くの開発現場で活用できるように、事例に基づいたガイドを作成した。

### Abstract

Recently, the rational and appropriate testings are more needed on the software developing projects because of requirement such as the shorter term of developments and the pressure by the delay at the previous development phase. But, a lot of engineers do design test cases, based on the intuitions or experiences of the project members. As a result, the designed test cases have lacks or duplications, because they do design the test cases, depended on their personal skills.

In order to solve these problems and to design the best optimized test cases, we tried and evaluated the CFD (Case Flow Diagram) test design technique. Further, we made the guides based on the real examples, aimed to be used on the software developing projects.

## 2. 背景

---

### 2.1 合理的なテストの必要性

---

近年、ソフトウェア開発において開発期間の短縮が求められ続けている。開発は常に納期との戦いであり、短い開発期間内に高品質なソフトウェアの開発・出荷を行う必要があ

る。本来、開発プロセスを最適化して品質を上げていくためには、開発の上流工程で品質を上げることが重要である。しかし一方、開発の短期化、前工程での遅延などのしわ寄せ等により下流工程であるテストが圧迫され、テストが不十分となる状況が見られる<sup>[1]\*1</sup>。

このような状況下においては、限られた時間内にテストケースの漏れや重複の無い合理的なテスト設計を実施することが重要となってくる。

しかし、それほど規模の大きくないソフトウェアでもテスト設計が難しい。その理由の1つとしてテスト条件の組み合わせの数が非常に多くなることが挙げられる。例えば、文字列を分析するソフトウェアという簡単な場合を考えてみる。8文字の英数字ファイル名のチェックを考えた場合、このファイル名の組み合わせは英数字の種類36の8乗=約3兆になり1秒に10,000の組み合わせをテストしても(通常はこのようなことはしない。開発者により常識的経験的に、またはテスト技法により削減処理される)全ての組み合わせをテストするには9.5年かかる<sup>[2]</sup>。このような簡単なソフトウェアでさえも、考えられるすべての組み合わせをテストすることはできない。従って、テストケースを削減する方法について考える必要があり、同時に合理的なものである必要がある。

## 2.2 テスト実施者の現状(アンケート結果から)

合理的なテスト設計が必要であるが、実際にはどのようにテスト設計が実施されているのか確認するため実態を調査した。具体的には、本グループのメンバーが自社の開発者に「テストに関するアンケート」と題して、以下の項目のアンケートを実施した。(付録1:アンケート結果参照)

1. 知っているテスト技法
2. 使っているテスト技法
3. 単体テストをどのように行っているか
4. 品質の良いプログラムを作る工夫

尚、アンケートに協力してくれた開発者は計32名、開発経験年数は1年~15年と幅広く、かつ6年以上のベテランが全体の6割以上を占めている。(図1参照)

開発者が品質向上に対して何に関心を持っているのかについて、質問4「品質の良いプログラムを書く工夫」から図2の結果が得られた。

「レビュー」の実施、「過去不具合対策」、「情報共有」、「コーディング規約遵守」などが上位を占めている。

これらの結果を見る限り、品質を上げるには上流工程での取り組み、即ち不具合を作り込まない工夫が効果大であるという考えが比較的浸透していると言える。一方、テストによる品質の確保は5位の第三者テストと6位のテストにおける品質向上だけであり、テストへの関心がやや薄いこともわかる。

また、テスト技法についての知識と使用状況につ

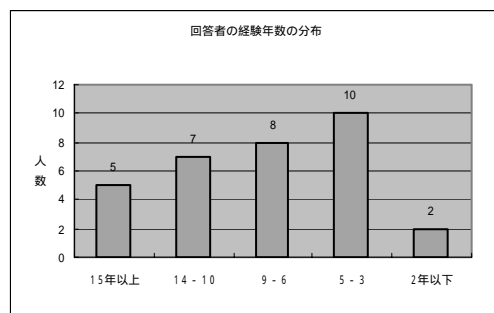


図1 アンケート協力者の経験年数

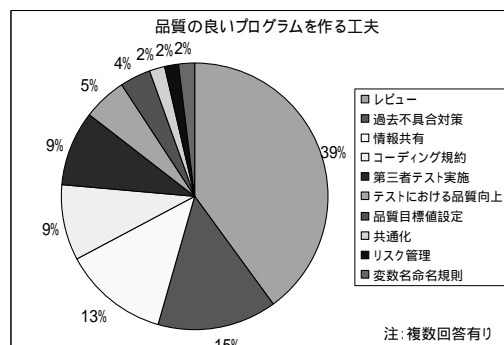


図2 品質の良いプログラムを作る工夫

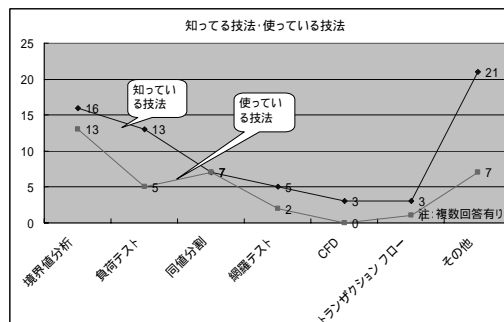


図3 知っている技法/使っている技法

\*1 雑誌「日経コンピュータ」による「2003年情報化実態調査」によると「システムの品質問題がどこで起きているか」という質問に対する回答で、要件定義の問題に続く第2位にテストの不十分が挙げられている。

いて、質問1.「知っているテスト技法」及び質問2.「使っているテスト技法」から、図3の結果が得られた。前述のように回答者にいわゆる「ベテラン」開発者も含まれているにもかかわらず、テスト技法に関する知識が少なく、テスト技法を使用している開発者は更に少ないことがわかる。

以上のことから、開発者は、合理的なテスト設計が求められている一方、テスト工程に関心が薄く、かつテスト技法によらない「経験」と「勘」による属人的方法によりテスト設計を実施していることがわかった。

そこで今回、合理的かつ属人性を排除したテスト設計技法について研究テーマとすることとし、これらの問題を解決するための技法として CFD 技法を研究のテーマとした。

### 3. CFD 技法の優位性

---

#### 3.1 現状の問題と従来技法の限界

---

##### 3.1.1 現状の問題

---

テストが軽視されテスト技法が浸透していない現状において、テストケースを抽出する場合に、「経験」と「勘」で実施する方法を取る場合が多い。この方法には次のような問題がある。

- ・ 網羅性を保証できない。即ち抜けがある可能性がある。
- ・ 重複が発生する可能性がある。
- ・ 優先すべき重要なテストケースが分からない。
- ・ テストケースの抽出根拠となる第三者が理解しやすい資料が無い為、テストケースのレビューを正確に行う事が出来ない。
- ・ 属人的で個人の技量が問われる為、一定水準以上の品質を確保する事が難しい。

これらに問題を解決する為には、「経験」と「勘」ではなく何らかの技法に従いテストケースを抽出する必要である。

##### 3.1.2 従来技法での問題解決とその限界

---

代表的なテスト技法として、原因結果グラフがある。原因結果グラフは同値分割された複数の原因（入力値）と複数の結果（出力値）の要素間の因果関係を論理演算子（AND、OR、NOT、EOR、等）と線で結合したグラフで表し、そのグラフからテストケースを抽出するためのデシジョンテーブルを作成するものである。（付録2「原因結果グラフによるテストケース抽出」を参照）

テスト対象の仕様からグラフを作成しテストケースを抽出する過程において、次の効果が期待できる。

- ・ 原因を同値分割する事により、似たようなテストケースを排除できる。
- ・ 結果に影響を与えない無関係なテストケースを排除する事が出来る。
- ・ 論理的にテストケースを抽出する事から、重複を排除しかつ網羅性を確保できる。

このように原因結果グラフは「経験」と「勘」でテストケースを抽出する際の問題を解決する手法の一つであり、一定の効果はある。

しかし、以下のような問題も残る。

- ・ 原因が多くなり原因間の関係が複雑になると、テストケースが多くなり削減効果が低くなる。
- ・ テストケースが多くなるとその優先度を考慮する必要があるが、優先度は付けら

れない。

- ・ 種々の論理演算子を用いる為、グラフ表現が複雑で理解しづらい。
- ・ それ故、習得しテスト設計するのに高いスキルを必要とする。

原因結果グラフは、ソフトウェア工学の教科書類ではテスト設計の基本としてよく紹介されている方法であるが、ヒアリング結果を見ても(付録1参照)「知っている技法」としても挙げられないほど浸透していない。以上のような問題がその背景となっていると思われる。

### 3.2 CFD 技法での問題解決

前節で述べたように原因結果グラフには問題点が残る。これを克服するものと考えられるテスト技法として、CFD技法に着目することとした。CFD技法は、原因結果グラフの発展型とも言うべき技法で、同値分割、デジションテーブルの手法を組み合わせたものである。以下に説明する。

#### 3.2.1 CFDでのテストケース抽出方法

まず、CFD技法を用いてテストケースを抽出するまでの基本的な流れを説明する。

仕様から原因を抽出し、同値分割する。

仕様から結果を抽出し、有効系と無効系に分類する。ここで、有効系とは仕様に定められた処理を行う(続行する)結果を指し、無効系とはそれ以外のエラー処理等で定められた処理が行われない結果を指す。

各原因と結果の関係を有効系と無効系とを明確に分け流れ図で表現する。これがCFD(Case Flow Diagram)である。(付録3を参照)

CFDをデジションテーブルに展開し、テストケースを抽出する。

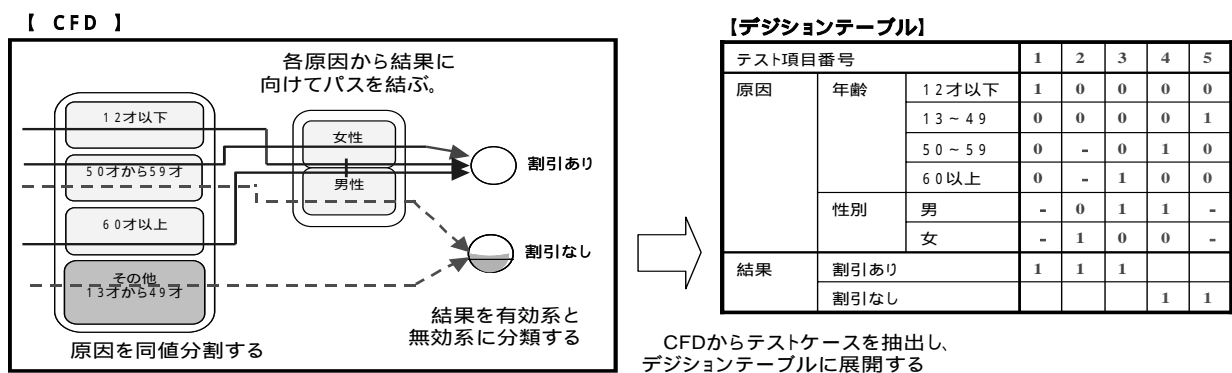


図4 CFD技法によるテストケース抽出

#### 3.2.2 CFD技法でのテストケース削減原理

CFDでのテスト項目削減は、「同値分割」「有効系の優先」「原因間の相互依存の排除」「階層化」によって実現される。

「同値分割」により、原因結果グラフと同様に類似のテストケースを排除する。

「有効系の優先」とは、より重要である有効系に対しては優先的にテストケースを抽出し、無効系に対しては優先度を落とすことで、テストケースを削減することである。

「原因間の相互依存の排除」では、依存関係の無い原因間のテストケースを排除することにより削減する。<sup>[2]</sup>(付録4を参照。)

そして「階層化」では、機能を階層で分割し、上位階層で結合する際に有効系の組み合わせを優先し、無効系同士の組み合わせを排除することにより、テストケースを削減する。

### 3.2.3 CFD技法の優位性

---

本節で述べてきたように、CFD技法は「3.1.現状の問題と従来技法の限界」で述べた問題を解決できる有効な手法である。その優位性を以下にまとめる。

- ・ CFD技法によりテストケースが削減できる、もしくは優先順位付けができる。
- ・ 原因結果グラフによる表現（付録2参照）とCFD技法による表現（付録3参照）を比較すれば明らかなように、CFDは可視性のある簡単な図表現でありテスト設計者にとって理解しやすい。合せて、第三者からも理解し易く、テストケースのレビューも容易である。
- ・ CFD作成の過程で、仕様や機能を机上シミュレートするのと同じ効果が得られることから、仕様の誤りを発見する事もできる。

CFD技法の優位性を確認できたことから、本グループではCFD技法を研究のテーマとして適切であると判断した。

## 4. モデルによる CFD 技法の試行

---

### 4.1 CFD技法の基礎知識の習得

---

本グループにおいてCFD技法を研究するにあたり、まず初めに我々がCFD技法の基礎知識を習得する事が必要である。そのため、市販の書籍<sup>[2]</sup>等での学習を試みた。しかし、CFD技法についての情報があまり掲載されておらず不足していたため、市販の書籍等だけでは理解する事が困難であった。そこで、CFD技法の生みの親である松尾谷先生がご自身のセミナー<sup>\*2</sup>のために作成されているテキスト<sup>[3]</sup>を、本研究のために特別に提供して頂いた。これを用いて、各自自習を進めるとともに、CFD技法の採用に取り組んでいる企業の方々<sup>\*3</sup>に講義していただくことにより、理解を深める事ができた。

### 4.2 試行対象となるモデルの選定

---

しかし、以上の過程でCFD技法の基礎知識は習得できたものの、CFD技法を実際にどのように適用すればよいのか具体的イメージが湧いてこない状態であった。そこで、CFD技法を実際のシステムをモデルとして試行することとした。

具体的には、我々が所属する企業で顧客との秘守義務、企業内の機密保持等の制約に抵触しないようなモデルを探した。なお、試行対象となるモデルの条件は、実際の業務アプリケーションに近いものであること、我々が共通認識を持つ事ができる内容及び規模であること、である。

その結果、某企業で設計技法に関する研修に用いたモデルである「レンタルビデオ予約システム」を選定することができた。共通認識が持ち易い身近な題材という面でも、適切なモデルであると言える。メンバの間で共通認識が持つことは、SPC研究会活動の運営において極めて重要である。

### 4.3 モデル機能「レンタルビデオの督促機能」での試行

---

「レンタルビデオ予約システム」の中の1機能である「レンタルビデオの督促機能」を対象としCFD技法を試行した（モデル機能の仕様及び試行結果については付録5参照）。

---

\*2 「ソフトウェア品質技術実践講座デバッグ工学とテスト技法コース」として日科技連にて開催

\*3 (株)インテック 前田直毅氏、TIS(株) 鈴木三紀夫氏

この試行により実際の設計書から CFD を作成する手順がより具体的に理解できた。

以上をまとめると、市販の書籍及びセミナーのテキストでの学習、講習会の実施等により、CFD 技法の基礎知識を習得できた。さらに、モデル業務アプリケーションを対象に CFD 技法を試行する事により、実際の設計書から CFD 技法を用いてテストケースを作成する手順について、イメージが湧かなかつたものがより具体的に理解できた。

## 5. 成果物

---

前節までで述べたように、本研究会の活動を通じて CFD 技法の特徴の理解、CFD 技法の内容の理解、実際の適用方法の習得、等ができた。しかし、我々の役割は各自の知識やスキルを向上させることももちろんであるが、本研究会の活動で得られたことを所属する各企業の中で展開することにある。

そこで我々は、そのために必要なものを本研究会の活動の成果物として作成することとした。

### 5.1 成果物の選定

---

成果物の候補として、研究会メンバーの中で議論し、以下の候補を得た。

- 1 - : 技法の解説書
- 1 - : 事例に基づいた実践的解説書
- 1 - : CFD 技法に適合した記述様式

1 - については、市販されている書籍、Web 等から得られるものでは、技法そのものの理解は不可能であったという今回の経験から挙げられた。

1 - については、活動の過程で得た情報から技法の表面的な理解は深まったものの、実際の場面でどう使うかが具体的に理解できなかつたという経験から挙げられた。

1 - については、社内で展開するには標準化が必要であり、記述様式の作成が不可欠との考えから挙げられた。

一方、実際に CFD 技法の適用を検討、あるいは試行されている企業の方々に、「社内展開で必要性を感じたものは何か」について、ヒアリングを実施した。その結果、以下の意見を頂いた。

- 2 - : 同値分割等テストの一般的基礎理論自体を解説するもの
- 2 - : キーワード集
- 2 - : 実践的な内容が必要

これらの候補から何を本グループの成果物とするべきか検討した。

1 - 及び 2 - については、必要ではあるが本研究会のこれまでの活動の中で既に個人レベルでは習得できたため、各企業の中で整備可能と判断し優先度を落とした。1 - の内容は、実践を積んだ上での課題であり、我々研究会メンバーが取り組むには時期尚早と考えた。2 - については、市販の書籍で充足できる部分が多いので、取り組みの優先度を落とした。

結論として、1 - 、2 - で代表される、「事例に基づいた実践的内容を整理したものをまとめること」を目標とし、「事例に基づく CFD 技法適用ガイド」を本グループの成果物として作成することとした。(付録 6 参照)

### 5.2 成果物の特徴

---

「事例に基づく CFD 技法適用ガイド」の特徴について、説明する。

### 5.2.1 事例の選定

---

前章で述べた CFD 技法試行の対象とした「レンタルビデオ予約システム」を「事例に基づく CFD 技法適用ガイド」を作成する際の対象事例とした。

### 5.2.2 盛り込んだ内容

---

本グループが入手できた書籍やセミナーテキスト等の既存の資料類は、同値分割や流れ図の記法等、技法の構成要素を説明するものである。そして、ソフトウェアの仕様の一部を取り出してその部分的仕様に対して CFD 技法の構成要素的技法を当てはめる形で説明されている。我々は、簡易なものではあるが、「レンタルビデオ予約システム」という具体的なソフトウェアの事例を対象として設定し、実際の設計書の中からテストケースを設計するという場面を想定して CFD 技法の適用事例としてまとめた。

また、既存の資料類では、CFD 技法と実際のテストフェーズとの関係が明らかにされたものが無かった。

我々は、単体テスト、結合テストとの関係を具体的に対応付けた手順を内容として盛り込んだ。

上記に示した内容とともに、既存の資料類では陽に示されていないが、我々が CFD 技法を習得する中で得られた内容を、成果物の中に盛り込んだ。例えば、CFD 技法によるテストケース削減メカニズム等の説明がその顕著な例である。

## 6. 結論

---

### 6.1 本年度の活動から得られたこと

---

ソフトウェアのテストは、テスト設計者個人に依存した属人的なものになっている。特に、技法に関する意識が希薄であり、テストケースの合理性や妥当性があまり考慮されていない現状がある。

これを克服するためのテスト技法として、CFD 技法は、従来技法を踏襲している、シンプルであり図の可読性が優れている、テスト項目の削減について従来手法より踏み込んでいる、等の観点から、他の技法に比して優位性を有しており、企業内等の組織レベルで展開をする価値がある。

企業内等の組織レベルで展開するには、公開された情報が不足している。特に、事例をベースにしたものが不可欠であるにもかかわらず、公開された情報は存在しない。我々研究会メンバは、今回の研究会活動でこの不可欠部分を補うための「事例に基づく CFD 技法適用ガイド」をまとめることができた。これを用いて、各々の所属企業内で展開活動に着手することが可能となった。

### 6.2 今後について

---

各々の所属企業内において、CFD 技法を展開するにあたり、以下を実施する必要がある。

活動当初に実施したアンケート結果から、CFD 技法の構成要素となっている既存の基礎的テスト技法について、開発者に定着させる必要がある。

今回の研究会活動の中では、事例の作成を優先し、CFD 技法そのものを解説するのは作成しなかったが、これを各企業で用意する必要がある。

自社の開発製品特性、開発者のスキル、開発プロセスに適合させるための、試行/評価/カスタマイズを実施する必要がある。

前項と並行に、  
、  
及び今回の成果物を活用した開発者へのトレーニングが必要である。この際、(株)インテック社における普及の取り組み施策「実践ワークショップ」が参考となると考える。<sup>[4][5]</sup>

## 7. 最後に

---

### 7.1 苦労した点

---

C F D 技法については、本報告中でも述べたが、通常的手段で得られる情報が非常に少なく、市販の書籍類、雑誌類から入手可能なものを収集しグループ内で学習したが、思うように理解が進まなかった。結果的に、様々な方々の特別なご協力を頂くことができたことにより C F D 技法を習得することができた。

また、研究会活動を進めるにあたっては、お互い扱うソフトウェア製品が異なり、勤務地も異なるメンバが、限られた時間の中で目標を共有し、成果物を分担して作成するのに困難が多かった。実際、グループメンバ 5 人の内 3 人がエンタプライズ系、2 人が組込系のソフトウェア製品の開発を手がける企業の所属であったし、複数メンバが業務により定例会含めた会合に参加できないこともあった。このような状況は我々のグループに特化した偶発的なものではなく、来年度以降の S P C 研究会においても続くと考えられる。しかし、各々背景が異なるメンバであるが故お互いを刺激し合える面もあり、「異業種混合」グループで活動すべきである。

### 7.2 謝辞

---

研究会メンバの C F D 技法理解のために情報提供下さった C F D 技法生みの親である松尾谷徹先生、企業内での試行経験を基に C F D 技法講習会の講師をお引受け頂く或は助言頂いた(株)インテック前田直毅氏、T I S (株)鈴木三紀夫氏、また、研究会定例会でご指導頂きましたテスト分科会の主査 / 副主査の先生方を始めとする方々、並びに各種調整依頼やお願い事等、我々の無理な要求に対応いただきました S P C 研究会事務局の方々に感謝いたします。

### < 参考文献 >

- [ 1 ] 「特集 プロジェクト成功率は 26.75%」, 日経コンピュータ n o 587, 2003
- [ 2 ] 松本正雄 / 小山田正史 / 松尾谷徹 (共著), 「ソフトウェア開発・検証技法」, 電子情報通信学会, 1997
- [ 3 ] 松尾谷徹, 「ソフトウェア品質技術実践講座デバッグ工学とテスト技法コース」セミナーテキスト, 日本科学技術連盟, 2004
- [ 4 ] 前田直毅, 「テスト技法「C F D」実践ワークショップの取り組み」, J a S S T 2004, 2004
- [ 5 ] 前田直毅, 「テスト設計技法 C F D の実プロジェクトへの適用と工夫」, 第 23 回ソフトウェア生産における品質管理シンポジウム 発表報文集, 日本科学技術連盟, 2004
- [ 6 ] 松尾谷徹, J a S S T 2005 チュートリアル「最先端のテスト技法 C F D」テキスト, 2005

# 付録1

## 「テストに関するアンケート」への回答(1/3)

開発経験 [年] (詳細ソート)	知っているテスト技法	使ってるテスト技法	単体テストをどのように行っているか	品質の良いプログラムを作る工夫		補足	
				個人	グループ		
a氏	1	ブラックボックステスト ホワイトボックステスト 境界値分析	ホワイトボックステスト 境界値分析	デバッグを用いて、実行しながら値が正しいかとかパスが正しいかをチェックする。 境界値分析を用いて、いくつかの入力を用意し、出力が正しいかをチェックする。	機能毎に関数を作成して、入出力の関係が分かるようにする。 何回も行われる処理は関数化して、メンテナンスをしやすくする。 エラー処理も含めて様々なケースを想定してプログラムを作成し、さらに様々なケースでテストを行なう。	ソースコードレベルのレビューや仕様書レベルのレビューなどを行なう。	レーザープリンタのWindowsプリンタドライバ設計者
b氏	2	境界値分析 制御バステスト	境界値分析 制御バステスト	テスト仕様書は自分で作成 デバッグ使用		レビュー(インスペクション)	半導体製造装置間で共通に使用する、アプリやドライバ開発
c氏	3	境界値テスト 機能パラメータ網羅テスト トランザクションテスト	境界値テスト トランザクションテスト	SECSIMのシミュレータを用いて、顧客のホストのメッセージを作成しテストを行った	できるだけ既存関数を用いて(改造業務が多かったため)、あまり同様の関数を作成しないように心がけた	顧客毎に通信仕様異なるため、できるだけ共通化できる部分は共通化するようにした	半導体業界の装置群コントローラ開発
d氏	3	操作テスト 負荷テスト リソースバステスト	操作テスト 負荷テスト	実機シミュレータを用いて、機能が正常に動作するかを確認した	新言語を使用している開発がメインだったため、できるだけソースコードレビューを行うように心掛けた	装置シミュレータのテストだけではなく、実機を用いて負荷テスト等の実証試験を行った	半導体業界の装置群コントローラ開発
e氏	3	ホワイトボックス ブラックボックス 境界値分析	ない	自分の感(あぶなそうところを重点的に)早い段階にお客さんに使ってもらう	単体テストOKで初めて作成済みのモジュールを組み込む、それまでは一緒にしない、他の人が分かるようにする(トリッキーな手法は使わない)誰でも修正可能にする、コメントを書く。	ない	版を配って、多くの人に使ってもらう。(オープンソース的な考え)テストとコードの質とは関係している。(コードを書いた開発者しかテストできない部分もある。)
f氏	3	ブラックボックス ホワイトボックス 境界値分析	ブラックボックス	テスト仕様書を作成し、テスト仕様書のレビューを行う、その上でテストを行う。	とにかく丁寧なソースを書く コメント又はきっちり書く ソースコードに修正が入ったときは、なぜ修正したのかまで記入する 汎用的な関数はモジュール化する 機能でモジュール化する	汎用的な機能はモジュール化して共有する 時々コードレビューを行う	
g氏	4	ブラックボックス ホワイトボックス トップダウン ボトムアップ	ブラックボックス	ユーザーインターフェースなので、実際に設定を実行、確認する	わかりやすいプログラムを作成する コメントを記述し、誰が見てもわかりやすく作成する ソース上のブラックボックスを無くす わからないことは、あいまいにせず識者に伺う	仕様書、検査書のレビューを実施し、情報を共有する ソースコードレビューを実施する	レーザープリンタのビデオコントローラ設計者 (プリンタ内蔵Web: パーコード関係が主、C言語で開発)
h氏	4	同値分割 境界値分析 CFD 状態遷移バス データフローバス	同値分割 境界値分析 データフローバス	テスト仕様書は未作成 関数単位でファイルにデータ出力し結果チェック デバッグ-を使って、関数の中をテスト	分かりやすいプログラムを書く 関数単位でファイルにデータ出力し結果チェック 直値の代入は行わない テストをしっかりと行う 技術テクニクがあれば品質も高くなる	第三者テストの実施 過去不具合の分析 レビューの実施 情報共有	半導体製造装置の通信系ソフトウェア開発
i氏	4.5	CFD	使用しているテスト技法はありません。	デバッグを使用して機能毎に関数の入出力等が正常に動作しているかをチェックする。 境界値も含めた具体的な設定値をいくつか代入し、正常に動作するかをチェックする。	分かりにくい処理には必ずコメントを付ける。 直接値を代入せずに、わかり易い名前でマクロを定義し使用する。 過去のソースで必要のないと判断したものは削除していき、見やすいプログラムとなるようにする。	必要に応じて機能毎にレビューを行う。 (外部仕様レビュー、内部仕様レビュー、ソースコードレビューなど)	レーザープリンタのビデオコントローラ設計者 (操作/パネル関係が主、C言語で開発)
j氏	5	ブラックボックス ホワイトボックス 負荷テスト	なし	テスト仕様書は未作成 デバッグを使って、関数の中をテスト 実機動作でテスト	分かりやすいプログラムを書く プログラムの中にコメントを入れる	過去不具合の対策に漏れがないよう、対策の必要なもの一覧表を作成	レーザープリンタのビデオコントローラ設計者 (フィーダー関係が主、オブジェクト指向言語で開発)
k氏	5	ブラックボックス ホワイトボックス 同値分割 境界値分析	同値分割 境界値分析	テスト仕様書は自分で作成 デバッグ使用 ある程度結合させて実行させる	単体テストを行う事、	各フェーズ毎のレビュー 過去不具合を検索できるようにして、同じトラブル出さない	半導体業界の装置群コントローラ開発
l氏	5	ブラックボックス ホワイトボックス 同値分割 トランザクションフローテスト 境界値分析	同値分割 境界値分析 トランザクションフローテスト	テスト仕様書は未作成 関数単位でデバッグ使って入力と出力をテスト	分かりやすいプログラムを書く ネットを避ける 直値の代入は行わない テストをしっかりと行う	第三者テストの実施 過去不具合の分析 レビューの実施 情報共有	半導体製造装置の通信系ソフトウェア開発
m氏	6	単体テスト 結合テスト システムテスト 負荷テスト ブラックボックステスト ホワイトボックステスト	単体テスト、 結合テスト、 システムテスト	テスト仕様書にそって行う。	見やすいプログラムを書く、簡単なロジックにする、インデントを上手に使う、処理時間を考慮する。 使いやすい機能が、要求に沿った機能になっているのを感じながらプログラムをする。本当は、設計段階で考慮することで、プログラム先行の108の場合はここで注意するしかない、これは駄目な例ですね。	各フェーズ毎のレビューをなんとか実施している。	
n氏	7	ブラックボックステスト ホワイトボックステスト ストレステスト 単体テスト 結合テスト(これらが含まれるかは？わかりません)	単体テスト	デバッグシリアルで、Printfで表示させるようにしておいて、簡単な境界値テストを行う。	なるべく見やすいようにスタイルを統一 たくさん行数を増やさない コメントになるべく入れる コンパイル時のWarningを取る。	開発環境WGの中で、QACの導入を進めている。 また、MISRA-Cやコーディングスタイルを今後教育していく。	レーザープリンタのビデオコントローラ設計者 (エンタレクション関係が主、C言語で開発)
o氏	7	制御バステスト 同値分割 境界値分析 制御バステスト ユースケース	同値分割 境界値分析 制御バステスト データフローテスト	テスト仕様書は作成していない、デバッグ-を使って、関数の中をテスト	複雑度を下げる努力 直値の代入は行わない 読みやすくする。 テストをしっかりと行う	第三者テストの実施 過去不具合の分析 レビューの実施 情報共有	半導体製造装置の通信系ソフトウェア開発

# 付録1

## 「テストに関するアンケート」への回答(2/3)

開発経験 [年] (詳細シート)	知っているテスト技法	使ってるテスト技法	単体テストをどのように行っているか	品質の良いプログラムを作る工夫		補足	
				個人	グループ		
p氏	7	単体テスト 結合テスト ブラックボックステスト ホワイトボックステスト 負荷テスト	単体テスト 結合テスト ブラックボックステスト 負荷テスト	テスト仕様書を作成してテストを行う。(もちろんレビューの済んでいるテスト仕様書を元にテストを行う。)	誰(自分も含む)が見てもわかりやすいプログラムを書く。あまり深いテストはしない。変数名は、意味のある名前にする。	テスト仕様書を作成後にレビューを行う。テスト時には、プログラマーとは別のメンバーがテストを行う。	
q氏	8	ブラックボックステスト ホワイトボックステスト マトリクス表	ブラックボックステスト ホワイトボックステスト マトリクス表	テストプログラムからの単体モジュール呼び出し、及びパラメータをランダムに振った単体モジュール呼び出し耐久テスト。 デバッグを用いたステップ実行。 シリアルポート出力によるデバッグプリント。	わかりやすい表現を用いること。 フェイルセーフを考慮すること。 コメントに"意味"ではなく、"意図"を記述すること。 慣習化できる構造、自動化できる構造を考へること。	コンパイルスイッチを乱立しない 共通変数・共通ラベルの命名について合意を取る。 機種間での互換性を保つ 問題点管理票データベース及び「本日発生M票一覧メール」を毎日自動的に流すこと	レーザープリンタのエンジン・ファームウェア設計者 (エンジン制御が主、C言語で開発)
r氏	8	ホワイトボックス ブラックボックス 境界値分析 カバレッジ(C0,C1) (xUnit, テストファースト技法か?)	境界値分析	モジュールを作成したら、それを使うドライバユニットを作成して、意図したとおり動作するかテストする。デバッグ中にテストは行わない(分離する)	シンプルに作る。各モジュールの先頭には必ずエラーチェックを埋め込む。命名規則を統一する。(抽象的な名前をさける = 変数を見ただけでなんのデータかが分かるようにする。)自分の流儀にこだわらない。(モジュールに修正、追加する場合は、そのモジュールを作成した人の意図に合わせる)簡単でも良いので、Readmeを必ず残す。	PLが機能、結合テストを行い、仕様と一致しているか確認する。	
s氏	8	ホワイトボックス ブラックボックス 境界値分析 カバレッジ	ない。勘、技法は知っているが、最終的には自分の勘。	モジュール単位で、テスト用ドライバを使ってテストを行う。機能テストは、何をテストするかリストアップ(仕様を元に勘で項目を列挙する)して行う。	モジュールを小さくする。コメントをしっかり書く。(関数にはもちろん、特に注意すべき箇所も)モジュール間結合度を低くする。命名規則を統一する。	コーディング規約、CVSを使っている。ソース、バージョン管理	どこでエラーの原因となるのかを特定する為の仕組み(Exception, ErrorCode, Log)を早い段階で作成し、組み込みは、テストでエラーとなった場合の対処がすぐ出来る。テストにより、エラーを検出するのはもちろん必要だが、問題は、早くその原因を特定し、すくに対処できる体制(仕組み)を取る事。 Unixは可能な限り小さいモジュールをたくさんつくり、一つの機能を実現する為にシェルでそのモジュールを呼び出す形をとっている。単体テストもしいない、不具合があった場合の対処もしやすい。
t氏	8	単体テスト、 結合テスト、 総合テスト、 負荷テスト、 ベンチマークテスト、 受入テスト	単体テスト、 結合テスト、 総合テスト	テストデータを作成し、実行させて、結果を確認する	繰り返しテストする	テストデータは他人に作成してもらおう	
u氏	10	ホワイトボックス法 (判定条件・条件網羅) ブラックボックス法 境界値分析 同値分割	ホワイトボックス法 ブラックボックス法 境界値分析 同値分割	システムテスト時にある単位(関数・サブシステムレベル)に着目して行っている。 単体テストと言うよりは、 デバッグと言う方が適当かもしれない。。。	開発に着手する前に以下の事を念頭に置いて行っている。 1.保守性・再利用性を考慮 2.改造仕様に適しているか(必要以上の改造にしていないか) 3.改造法は適当か(複雑にしていないか) 4.ロジックに誤りがないか(複雑にしていないか) 5.過去のトラブル事例を考慮(同じ過ちを繰り返していないか) 6.リスクはないか? (負荷・耐久性も考慮)	開発に着手する前に以下の事を念頭に置いて行っている。 1.保守性・再利用性を考慮 2.改造仕様に適しているか 3.改造法は適当か 4.ロジックに誤りがないか 5.過去のトラブル事例を考慮 6.リスクはないか? グループ内にて変更要求/問題点処理の管理を行っており皆で意見を出し合い必要なレビューも実施 妥当と判断された場合(承認)のみ開発に着手出来る仕組みになっている。	
v氏	10	境界値分析 同値分割 ブラックボックス ホワイトボックス	境界値テストが主? 個人によって差がある	個人任せ?	10行~20行単位で何を目的とした処理なのかコメントを書くようにしている	レビューの開催 & 指摘	
w氏	10	同値分析 境界値分析 機能網羅テスト、 ユースケーステスト、 状態遷移テスト CFDテスト、 ブラックボックス テスト ホワイトボックステスト	同値分析 境界値分析、 機能網羅テスト、 状態遷移テスト、 ブラックボックステスト	プログラム仕様書に基づいて、テストケースを作成、ドライバなどの作成をしてテストする。	テストケース作成時には、(コーディング時に無意識的に理解してしまった仕様の勘違いを排除するよう仕様のひとつひとつを再確認する。またテスト時にバグがあれば、その場しのぎの修正を避け、プログラム機能に対してより妥当な修正であることを確認して修正する。	テストケースの網羅性の確認方法、実施環境の統一化、テスト結果の確認方法など、グループ内で方法を統一する。	
x氏	10	ブラックボックステスト システムテスト ホワイトボックステスト 受入テスト 負荷テスト 結合テスト	単体テスト 結合テスト システムテスト	事前準備として物理設計書より、設計されたインターフェース、ロジックのパスを洗い出しながらテスト項目書を作成する。次に作成したテスト項目を検証できるドライバ、テストデータを作成する。事前準備ができたら、単体テストを実行して、不具合を修正し、テストを繰り返す。コードがテストに合格するまでテストを繰り返し、すべてのテスト項目をパスしたとき、コードを「完成」として、単体テストを終了する。	作成したプログラムをライブラリ化しておき、同じようなロジックのプログラムを作成するときを参考に、ライブラリ化したソースは新しく作成したプログラムと置き換え、常に分かりやすいやさや効率、メンテナンス性を少しずつ上げて(チューニング)より良いサンプル(パターン)を作るようにする。	陥りやすいコーディングミス(例えば、ネーミング規則の統一、メモリを大量に使う恐れのあるコーディングやバグになる可能性の高いコーディング等)を指摘するコーディングルールを利用して、グループで統一の取れたコーディングが行えるようになる。	
y氏	13	ブラックボックス ホワイトボックス 負荷テスト	ブラックボックス ホワイトボックス	デバッグ用のプログラムを作成し、挙動が変わるすべての正常値と異常値を入力とし、出力結果をデバッグプリントすることで単体モジュールテストを実施しています。	機能仕様や実装仕様の検討に時間をかける(いきなりコーディングに入ることはない) 経路上で仕様検討する。大きな機構は有識者を集めて必ずレビューを実施する	レビューの実施を促す スケジュールを引くときにの仕様検討のフェーズを入れるようにする	レーザープリンタのビデオコントローラ設計者 (全般的に担当、C言語で開発)

# 付録1

## 「テストに関するアンケート」への回答(3/3)

開発経験 [年] (詳細シート)	知っているテスト技法	使ってるテスト技法	単体テストをどのように行っているか	品質の良いプログラムを作る工夫		補足	
				個人	グループ		
I氏	13	境界値分析 境界値分析 ブラックボックス ホワイトボックス	境界値分析	テスト仕様書は未作成 デバッガを使用して、関数の入力を境界値に振って出力を確認し、必要に応じて、テストツール等を作成。	「昔は容量の制限があったため、わかりやすさよりも処理スピードとプログラム量をなるべく小さくすることを重視していた。最近では、プログラム量についてはあまり重視せず、誰がみてもわかりやすく(単純で、さらに処理速度を重視している。ソフト機能単位で組み込み、切り離しが可能なようする。	プログラムの書き方の取り決めが一般にないが、またかなり個人差があるようだ。 過去不具合を個人個人で持ち合っていたレビュー実施。(すべての抽出にはいたっていないが...)	レーザープリンタのビデオコントローラ設計者 (全般的に担当、C言語で開発)
A氏	15	同値分割 境界値分析 トランザクションフロー 網羅テスト	同値分割 境界値分析 網羅テスト	テスト仕様書は未作成 関数単位でファイルにデータ出力し結果チェック デバッガを使って、関数の中をテスト 簡易ツール作成してテスト	設計書を充実させる。 ネットを避ける 直値の代わりは行わない テストをしっかりと行う 標準ライブラリ等の利用	第三者テストの実施 過去不具合の分析 レビューの実施 情報共有	半導体製造装置の通信系ソフトウェア開発、リーダー
B氏	15	境界値分析 ブラックボックス ホワイトボックス	常に使っている技法は無いが、状況に応じて参照する。	ソフト開発というよりもカスタマイズ(客先仕様)による変更が多い為、変更した箇所に関する機能をリストアップし重点的にテストする。装置の制御がメインとなる為、シミュレータでの通信のタイミングに注意してテストを行う。	他装置との通信がメインなので、他装置の仕様変更等に柔軟に対応できる作りを心がける コーディングよりも設計に時間をかける モジュール化を心がける 判りやすいコメント	仕様書 / 設計書作成時にレビュー実施 ソフト変更後、ソースコードレビュー実施 コーディング規約作成(現在作成中)	
C氏	15	全体にV字モデルに沿ったテストを実施する。 単体テスト・ホワイトボックステスト ブラックボックステスト:主に人出力の確認 結合テスト システム全体のトランザクションなどの単位で、インタフェースを介したモジュール間の連携がうまくいっているかのテストを実施する。原則的に、まずは下位モジュールの法から積み上げて結果を確認していく。 総合テスト業務の流れにしたがってプログラムを実行させて、業務全体の流れを想定した通りにサポートしているかをテストする。 回帰テスト修正が発生したときの影響確認テスト ストレステスト 大量データまたは大量クライアントからの同時アクセスなどでシステムに大きな負荷をかけた際の動作テスト 受け入れテスト ユーザによる受入確認テスト	単体テスト ブラックボックステスト ホワイトボックステスト 結合テスト 結合テスト 回帰テスト ストレステスト 受け入れテスト	ホワイトボックステスト ツールがある場合には、ツールが利用できる場合はツールを利用して実行。ツールがない場合には、プリントコマンドなどを入れて、手作業で網羅率を確認。ブラックボックステストモジュールの設計書から入力出力を確認。想定される出力および異常系の出力が得られるように、いくつかのパターンを想定して入力を設定してテストを実施。両者の場合に共通インタフェースの定義から想定できるテストケースを作成し、それに則してテストを実施する。上位、下位のモジュールができていないときには、ドライバ/スタブを用意して実施する。	テストケースを作成する場合には、できるだけ網羅性の高いケースを作成してテストする。特に、異常系、準正常系は考慮ミスで問題が発見されることが多いと思われるため、これらをテストできるようにテストケースを準備する。入力データとその予想結果を明確にし、予想結果の証拠(画面ダンプなど)をしっかりと保存する。テスト以外に、プログラムの品質という観点では、プログラミング(言語だけでなく、構造化の方法なども含む)技術を向上させるように、学習をするというようにしている。	システムの仕様書、設計書を全員が共通で理解できるようにする。各人が持っているノウハウなどを積極的にグループ内で利用できるよう、コミュニケーションを密にする。初心者が参加している場合には、ベテランを指導者として組むようにするなど、低スキル者のスキル向上をはかる。品質に関する目標値などを数字で明確にして、それを達成するようにする。	
D氏	15	技法と言う名称で呼んでよいかかわりませんが、種類としては 単体テスト、 結合テスト、 システムテスト、 アクセプタンステスト ストレステスト、 システムテスト、 アクセプタンステスト、 ストレステスト、 パフォーマンステスト、 ホワイトボックステスト、 ブラックボックステスト、	単体テスト、 結合テスト、 システムテスト、 アクセプタンステスト ストレステスト、 ホワイトボックステスト	テストツールがある場合には、テストツールを使い実施テストツールが無い場合には、ドライバー用のプログラムとスタブ用のプログラムを使いテストステートメントを挿入して確認しながら実施テストステートメントは、テスト終了後コメントアウトにするようにして消さないようにしているまた必ず、テスト仕様書を作成して、仕様書のテストケースのシナリオに沿って実施	テストという観点では、いままでデータセントリックなシステムの開発が主だったので、テスト仕様書作成で、利用データとテスト実施結果の予想結果をきちんと記述すること、テスト結果や途中の変数のスナップショットをエビデンスとして、画面のコピーや、データベースのダンプをとって確認するようにしている。また、テストケース作成の時に、正常系、システムの異常系だけでなく、準正常系として、ビジネスルール上の異常な場合を確実に考え実施するようにしている。	テストという観点では、テスト仕様書のレビューとテスト結果のチェック(エビデンスの中身の確認)を行うようにしている。できれば、テスト実施者及びテスト仕様書作成者と、プロム作成者や設計者とは、別の要員が担当するようにしている。あと、品質指標値を出してもらいにれに満足しているかを評価することを指向しているが、品質指標値の作成方法や、どのようなあたりを持って十分とするかの考え方が組織的に整理できていないのが課題。	
E氏	15	ブラックボックステスト(設計書にそって設計書の機能確認テスト)ホワイトボックステスト(プログラムロジックの全ステップの実行テスト)トップダウンテスト(上位モジュールから下位モジュールへの連結動作確認テスト)ボトムアップテスト(下位モジュールから上位モジュールへの連結動作確認テスト)回帰テスト(プログラム変更時の影響確認テスト)ストレステスト(大量データやHW、SWが正常動作するか)単体テスト(個々のモジュールが設計書にそって動作するか)機能テスト(結合テスト(モジュール間のインターフェイス確認テスト)システムテスト(本番環境を想定しシステム全体が要求仕様に基づいた動作や性能を発揮しているかの確認テスト)ビッグバンテスト(まとめてシステム全体の動作をさせ一気にテストする確認テスト)	ブラックボックステスト ホワイトボックステスト トップダウンテスト、 ボトムアップテスト、 回帰テスト、 ストレステスト、 単体テスト、 結合テスト、 システムテスト	入力インターフェイスから想定されるテストケースを作成する。テストケースには、入力値や入力条件、予測値や注意点を記述していく。仕様書にもつづき仕様書に記載されている機能が、既に作成したテストケースで、すべてのケースをカバーできるかチェックする。カバーできていない場合は、テストケースの追加をする。出力インターフェイスにそって編集確認をすべきすべての項目を洗い出す。機能仕様から考えられる出力インターフェイスの編集パターンを洗い出しエクセルなどの表にまとめてテストケースを検証する。テスト後にチェックする際には、全出力項目の編集確認やテストケース毎の編集確認チェックをする。	プログラム機能上、重要なロジックとなる部分について、できる限り想定できるテストケースを洗い出して、重要なロジック部分を重点的にテストする。但し、これには機能要求を熟知してキモとなる部分を把握する必要があります。	あたり前のことであろうが以下のような点を考慮するようにしています。 グループ内での知識の共有、方式の徹底をする。 グループ内で共通した開発手順を徹底すること。 先行して進んでいる開発メンバーが得た様々な情報は、進行が遅れている開発メンバーや後継で開発がスタートした開発メンバーに公開できる体制を作る。	

## 付録2

# 原因結果グラフによるテストケース抽出

以下の仕様(付録3と同じもの)から原因結果グラフによりテストケースを抽出する例を示す。  
仕様

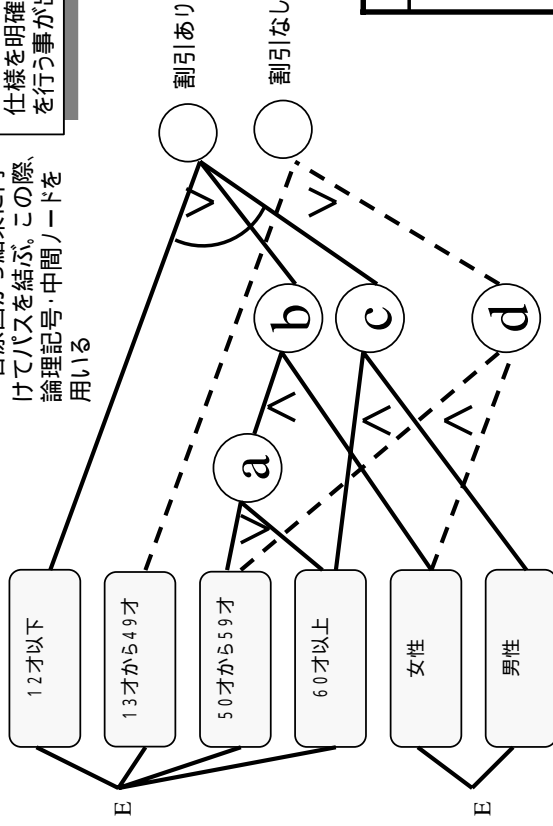
12才以下か、50才以上の女性か、60才以上の男性なら割引を行う。(そうでなければ割引しない)

### 原因結果グラフ

各原因から結果に向けてパスを結ぶ。この際、論理記号・中間ノードを用いる

### 設計の可視性ポイント

仕様を明確にでき、論理的なテスト設計を行う事が出来る。



原因を同値分割する

### 削減・網羅性ポイント

同値分割による、似たようなテストケースの削除  
同値分割された原因の種類を明確にする事によりテスト漏れを回避

**削減・網羅性ポイント**  
確実にテストパスを追う事により重複・漏れを回避

グラフの流れからテストケースを洗い出し、デジモンテーブルに展開する

### デジモンテーブル

テスト項目番号	1	2	3	4	5	6
原因						
年齢	12才以下	1	0	0	0	0
	13~49	0	0	0	0	1
	50~59	0	1	0	0	0
	60以上	0	0	1	1	0
性別	男	-	0	0	1	-
	女	-	1	1	0	-
結果	割引あり	1	1	1	1	
	割引なし					1

### 削減箇所

結果につながる原因の中で、因果関係が無いところが削除になった

# 付録3

## CFD技法によるテストケース抽出

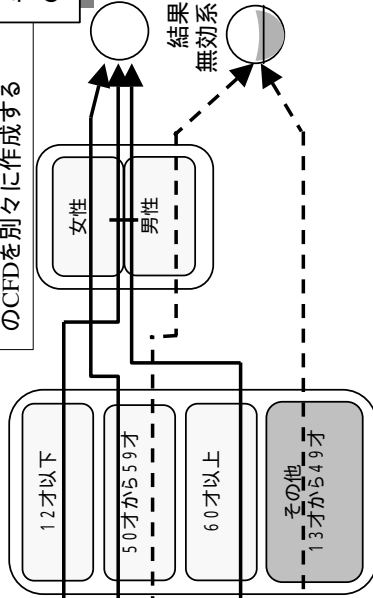
以下の仕様(付録2と同じもの)からCFD技法によりテストケースを抽出する例を示す。

### 仕様

12才以下か、50才以上の女性か、60才以上の男性なら割引を行う。(そうでなければ割引しない)

#### 【 CFD 】

各原因から結果に向けてパスを結ぶ。この際、有効系と無効系の流れを明確に分け、場合によっては有効系と無効系のCFDを別々に作成する



原因を同値分割する

**削減・網羅性ポイント**  
同値分割による、似たようなテストケースの削除  
同値分割された原因の種類を明確にする事によりテスト漏れを回避

**設計の可視性ポイント**  
仕様を明確にでき、論理的なテスト設計を行う事が出来る。  
有効系・無効系を分ける事によってテストの優先度・他機能の投入に必要な結果が明らかになる

+

**削減ポイント**  
無効系に対しては、他機能との結合試験の組み合わせを行わない事により、優先度の低い項目を削除する  
有効系に対しては、最小経路でのテストケース抽出を行う

**削減・網羅性ポイント**  
確実にCFDの流れを追う事により重複・漏れを回避

**【デジシオンテーブル】**  
CFDからテストケースを洗い出し、デジシオンテーブルに展開する

テスト項目番号	1	2	3	4	5	
原因	12才以下	1	0	0	0	0
	13~49	0	0	0	0	1
	50~59	0	-	0	1	0
	60以上	0	-	1	0	0
性別	男	-	0	1	1	-
	女	-	1	0	0	-
結果	割引あり	1	1	1	1	1
	割引なし					

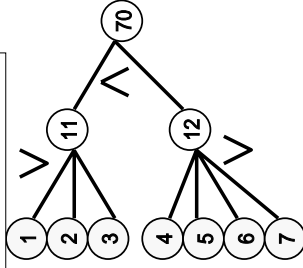
**削減箇所**  
有効系に対して、最小経路でのテスト項目抽出が行われた

**削減箇所**  
結果につながる原因の中で、因果関係が無いところが削除になった

原因間の相互依存の排除によるテストケース削減

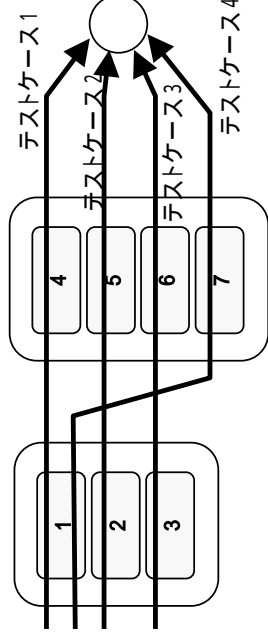
原因結果グラフとCFD技法とのテストケース抽出方法の違いを示す。

原因結果グラフ



原因	テストケース											
	1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	1	1								
2					1	1	1	1				
3								1	1	1	1	1
4	1				1			1				
5		1				1				1		
6							1				1	
7								1				1
有効結果	1	1	1	1	1	1	1	1	1	1	1	1

CFD技法



原因	テストケース			
	1	2	3	4
1	1			
2		1		
3			1	
4	1			
5		1		
6			1	
7				1
有効結果	1	1	1	1

原因結果グラフと比較する事により、CFD技法の原因間の相互依存の排除によるテストケース削減原理を説明する。

原因結果グラフでは、原因と原因との関係から12通りのテスト項目を作成する。

CFD技法では、原因と原因とを独立と見なし、最大の分岐数である4通りのテスト項目のみを作成する。

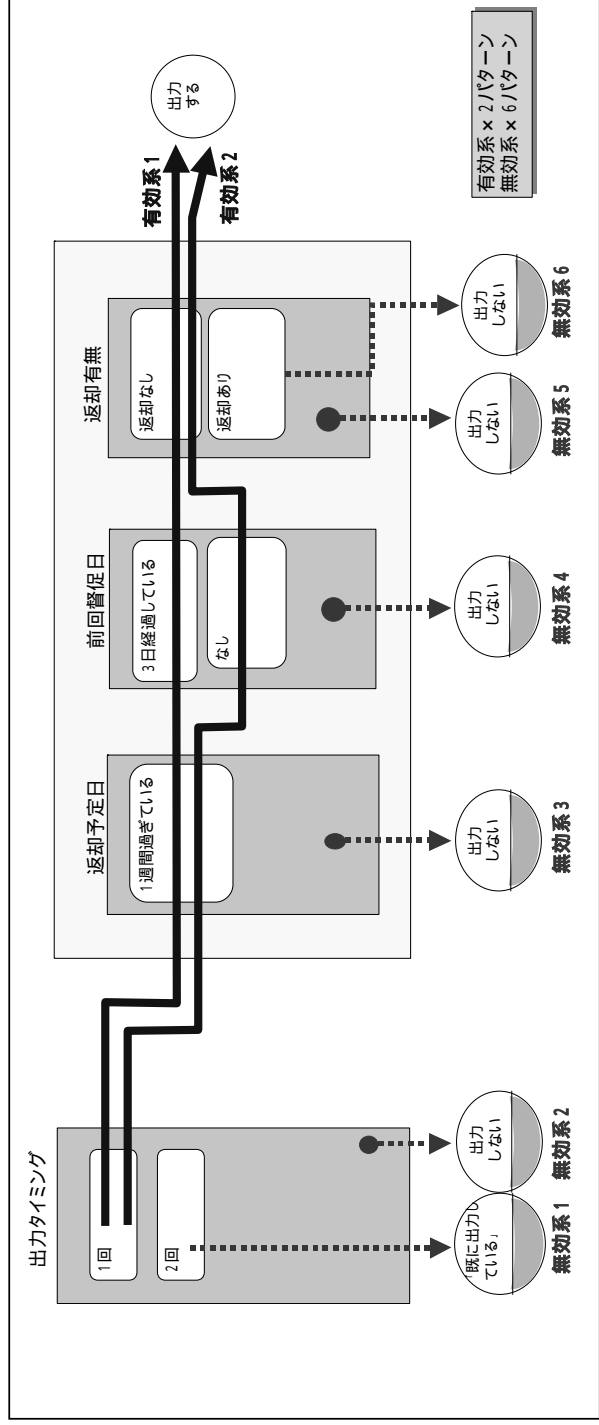
つまり、この2つのデジジョンテーブルの違いは、原因間の相互依存のテストを含めるか否かの違いである。

「レンタルビデオ予約システム」の「督促」業務の仕様(の一部)から、テストケースをCFD技法で抽出する事例を示す。

設計書

システム名	機能名	督促
レンタルビデオ予約システム	機能名	督促
起動種別	条件：1日に一度	操作者 従業員
内容 詳細	<p>システムから督促リストを出力し、それに基づいて延滞している会員に対して督促の電話をかける。システムで支援するのは督促リストの出力である。</p> <ul style="list-style-type: none"> <li>・1日に1度督促リストを出力する。</li> <li>・同日に2回以上この処理をしようとしたときには、「既に出力している」ことを知らせ、処理をさせない。</li> <li>・督促リストの出力対象は以下の条件のどちらかを満たすものである。 <ul style="list-style-type: none"> <li>- 返却予定日を1週間過ぎていて、かつ、前回督促日から3日経過している、かつ、まだ返却されていない、実体「貸出明細」中のインスタンス。</li> <li>- 返却予定日を1週間過ぎていて、かつ、前回督促日が無く、かつ、まだ返却されていない、実体「貸出明細」中のインスタンス。</li> </ul> </li> <li>・督促リストの出力形式は以下の通り。出力すべき情報内容に関しては下記の情報定義を参照。 <ul style="list-style-type: none"> <li>- 会員別にソートされている。ソートは会員番号で昇順である。</li> <li>- 会員毎に、返していない映画のタイトルのIDが表示される。映画のタイトルはソフト品番でソートされている。</li> <li>- 会員毎に連絡したか否かのチェックをつけるようにする。</li> </ul> </li> <li>・督促リストの出力をシステムに指示する際には、指示する従業員のIDを入力し、督促リストにはその従業員名を出力する。</li> <li>・システムが行なうのは督促リストの出力までである。それ以降の電話連絡、連絡したかの消しこみは出力リスト(紙)上で行なう。</li> </ul>	

CFD



# 「事例に基づくCFD技法適用ガイド」のイメージ

「事例に基づくCFD技法適用ガイド」の概要を示す。

## 【全体ページ構成】

