

詳説ラルフチャート

2009/11/25 (水)

秋山浩一

1. ラルフチャートの位置づけ

ソフトウェアテストを実施する時には、テストオブジェクトを識別し、それを様々な「視点」から扱いやすい粒度になるまで分解する必要がある。今回は、仕様を目的機能単位に分解し、それらを FV 表によって整理する方法について解説をおこなった。今回は、分解された FV 表の 1 行分、すなわち各々の目的機能に対してその要素と構造を明らかにするためのノートーションであるラルフチャートについて解説をおこなう。

ラルフチャートとは、もともと、ゼロックスの Ralph G. Faull が品質工学（タグチメソッド）による試験を行なうための因子の洗い出しに好んで使用していた表である。原型とも言うべきラルフチャートを図 1 に示す。

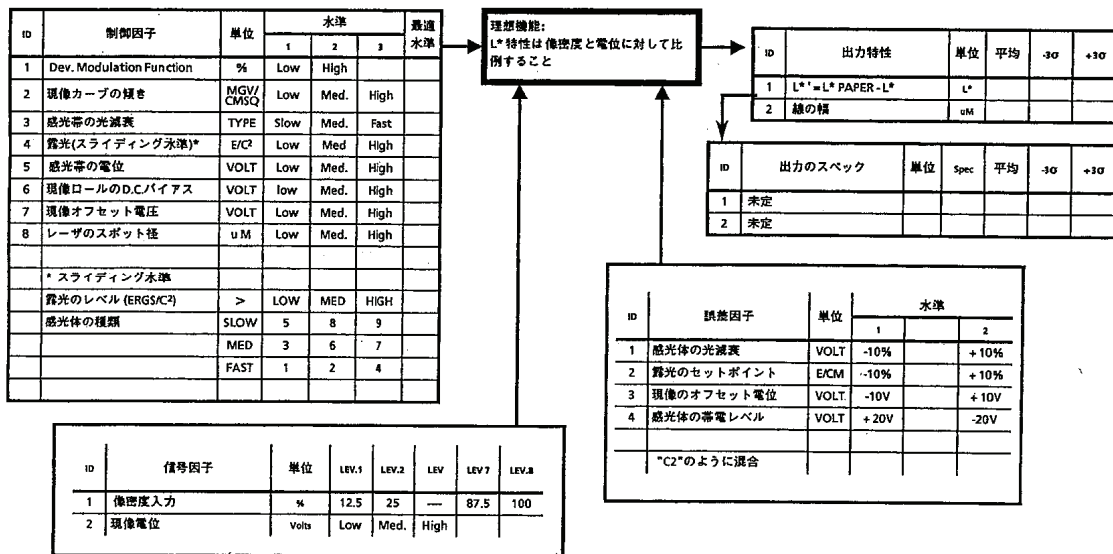


図 1 ラルフチャートの原型

(出展：品質工学 Vol. 3 No. 5 /Ralph G. Faull の論文より)

現在、品質工学の中で、「プロセス・ダイアグラム」、「P-ダイアグラム」、「システムチャート」と呼ばれるものは、このラルフチャートの派生系である。

ソフトウェアテストの技法のひとつである HAYST 法で使用しているラフチャートは、図 1 のラフチャートの改良版であり、クリーンルーム手法で”Box Structure Method”を提唱した Harlan Mills のコンセプトを取り込んだものになっている。

Harlan Mills は、”Box Structure Method”において、3 つのボックス構造モデルを提案している。それぞれ、”Black Box Specification (BB)”、”State Box Specification (SB)”、”Clear Box Specification (CB)”である。図 2 のように、機能をブラックボックスとして取り扱う BB モデル、状態変数の変化まで取り扱う SB モデル、アルゴリズムまで取り扱う CB モデルの 3 つのモデルである。Harlan Mills は、ソフトウェア開発時に、BB モデルを仕様化し、次に SB モデルで状態の仕様を考え、最後に CB モデルを用いて設計した後に、コーディングを実施することを推奨している。

ソフトウェアテストにおいても、特に「状態変数」に着目することは有効である。

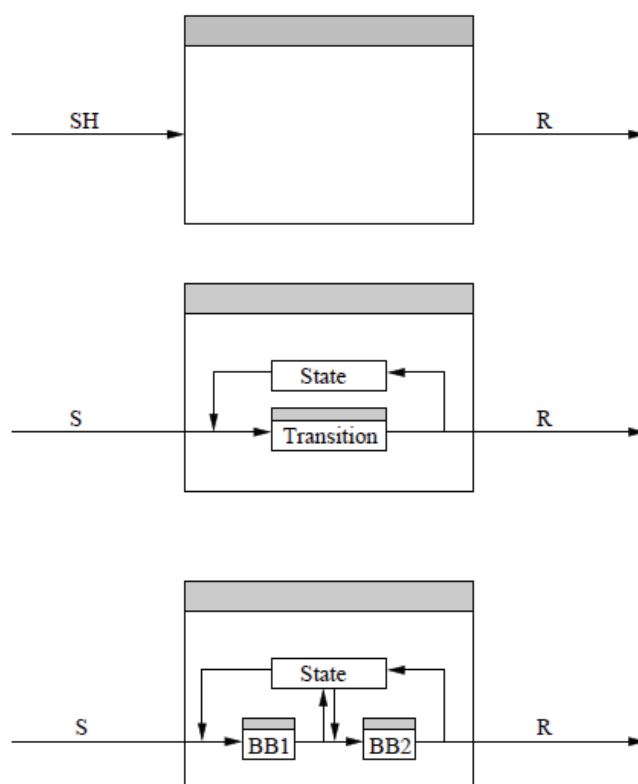


図 2 Harlan Mills のボックス構造モデル
(Industrial Formal Techniques より：上から BB、SB、CB モデル)

2. ラルフチャートの形式

HAYST 法で使用しているラルフチャート (“Ralph’s chart”) の、基本のフォーマットは、図 3 の通りである。

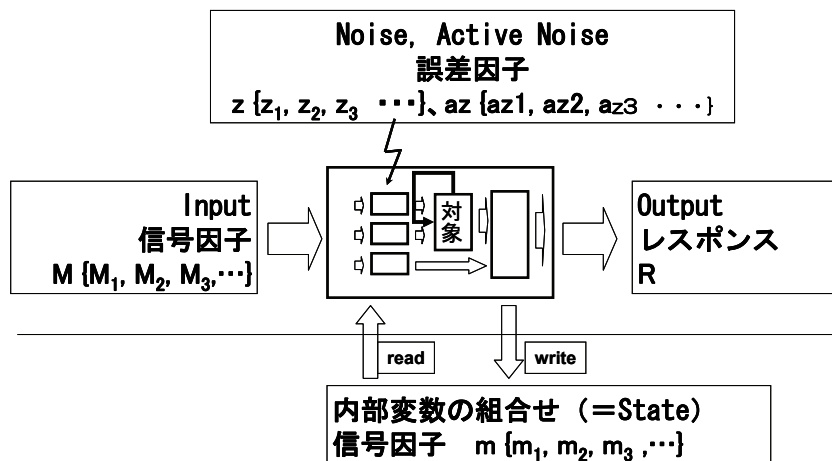


図 3 HAYST 法で使用しているラルフチャート

定義： ソフトウェアの操作 f は、以下の通りに定義される。

ソフトウェアの動作を、出力 R への関数として考えると、

$$R = f(\text{Input}, \text{State}, \text{Noise}, \text{Active Noise})$$

であらわされる。Input は、テスト対象システムへの入力、State はテスト対象システムの状態、Noise は入出力の関係を妨げる要因、Active Noise は悪意を持ったユーザによるシステム破壊要因（セキュリティの弱点を突いてシステムを破壊する行為など）を示す。

HAYST 法のテストモデルと、Harlan Mills の「ボックス構造化モデル」との相違点は、市場条件である Noise と Active Noise を明示的に書き込んでいる点にある。

従来から、ソフトウェアテストにおいて「意地悪テスト」と称し、負荷をかける、あるいは、少ないリソースでテストをすることが行われている。ところが、テスト対象のソフトウェア（部分）に対してどのような意地悪条件が効くのかについて十分な検討は行われてこなかった。本来、ソフトウェアにはそのパーツ毎に考慮しなければならない意地悪条件は異なるはずである。HAYST 法では、テスト設計の前段階で、ラルフチャートを使用して当該ソフトウェアが持っている弱点を狙ったノイズ、使用されるユーザ環境の違い、悪意を持ったユーザによる破壊行為を徹底的に洗い出してからテスト設計を実施する。

3. ラルフチャートの使い方

本項では、ラルフチャートの使い方について、解説する。

ラルフチャートでは、Input、 State、 Noise、 Active Noise を明らかにすることが目的である。FV 表を作成する際に、6W2H、すなわち開発の視点から、Why? (目的)、What? (仕様)、How? (設計) を再確認し、コンテキストの視点から、When? (いつ)、Where? (どこで)、Who? (誰が) を想定することで場を作り出し、How much? (セールスポイント)、Whom? (誰のために) で魅力的品質やお客様のお客様について考慮している。

したがって、FV 表が完成した時点で、ある程度、因子・水準はリストアップされているはずである。しかし、それをラルフチャートにして眺めて、レビューすることにより、多くの場合において、抜けている因子・水準を発見することができる。

以下、Input、 State、 Noise、 Active Noise の詳細について述べる。

3.1 Input

まず、FV 表により見出した目的機能に対する入力となる因子を考える。この因子のことを信号因子と呼ぶ。

信号因子は、目的機能を実現するためにお客様が入力するもの（＝お客様の意思そのもの）であるから、その結果はテストケースで言うところの「期待結果」にあたる。信号因子は単独の場合もあれば、複数の信号因子をセットしたあとに目的機能を動作させるトリガーとなる OK/完了ボタンを押して同時にシステムへ入力する場合もある。後者の場合は特に信号因子間同士の交互作用が発生していないことを組合せテストで確認する必要がある。

次に、Input となる信号因子の見つけ方について説明する。信号因子の多くは、What? (仕様) から発見する。すなわち、仕様書に書かれている「入力」フィールドをリストアップする。ただし、それだけでは、同時に使われる可能性が高い機能の検討が漏れるため、見つけた信号因子に対して Why? (目的) 展開を実施し、同じ目的を持つ別の信号因子を見つけることが大切である。

なお、信号因子の水準については、仕様書に十分に記載されていない場合があるため、How? (設計) に入り込み、場合によってはソースコードを読みながら境界値分析をして隠れ境界値（特異値）を見つける必要がある。

3.2 State

State は、目的機能が動作するときに、読み書きされる内部変数の組合せのことである。ところが一般に読み書きされる内部変数は多数存在する。したがって、その中でも特に重要な管理対象となる内部変数をリストアップすることが肝要である。

「重要な管理対象となる内部変数」を見つけるコツは、テストにおける「重要」の意味を考えることである。

テストにおける「重要」とは、「お客様にとって重要」と「開発者にとって重要」という2つの意味がある。つまり、「お客様自身が何らかの意図を持ってセットした変数」や、「使用のたびに予期せぬ値になることから組合せ問題を起こしやすい変数」を見つければよい。

このような、変数のほとんどは、実は、お客様自身が目的機能を使用する前に、セットしたもので、すなわち過去の信号因子である場合が多い。何故なら、お客様の意図をシステムに伝えるためには Input することが必要となるからである。ソフトウェアが変更する内部変数はパターンが限られるのに対して人間が変更する内部変数は予期せぬ値を持つことが多い。したがって、State の見つけ方は、その目的機能が参照する「過去に入力した信号因子」を見つけてやればよい。

3.3 Noise

Noise とは、Input の結果として働く処理を邪魔し、期待結果を得られなくするものである。Noise を受けてもきちんと機能することをロバストネス（頑健性、堅牢性）といい、「機能性が高い」（意地悪条件を受けても機能がきちんと働く性質が高い）と表現することもある。品質工学では、ノイズのことを誤差因子とも呼ぶ。

ハードウェアの世界において、Noise には、外乱、内乱、部品ばらつきが3つがあると言われている。外乱とは、市場環境の多様性を指す。つまり様々な顧客環境下できちんと動作するかということである。内乱は、商品の劣化（経年劣化）に対して長く安定して動作すること。部品ばらつきは、多少の部品の性能にばらつきがあったとしてもそれをカバーしあい、全体として問題なく動作することを表している。

ソフトウェアの世界においては、主に外乱を取り扱う。ソフトウェアに劣化や製品ごとの部品ばらつきは考えにくいからである。ソフトウェアに対する外乱は、ハードウェアと同じく市場環境の多様性であるが、もう少し細かく見ると、コンテキストの違い、すなわち、

When? (いつ)、Where? (どこで)、Who? (誰が) を想定することにあたる。

したがって、誤差因子を発見する方法は、コンテキストを想定することと同じである。ただし、単に、「時・場所・人」を想定するだけではだめで、それぞれに対していじわる条件を考える必要がある。具体的には、「時・場所・人」から発見した誤差因子に対して、いじわるな形容詞や副詞を連結させたものを水準として採用するとよい。

つまり、「時」であれば「すばやく、ゆっくりと、長く、短く、早い時期に、遅い時期に、...」といった水準をテストする。「場所」であれば、「負荷が高い環境、負荷が低い環境、古い機材、最新のソフトウェア、...」、「人」であれば、「初心者、熟練者、学生、子供、男性、女性、日本人、米国人、...」といった具合である。

3.4 Active Noise

最後の Active Noise は、システムのクラッキングなどの意図的な犯罪行為のことである。ソフトウェアテストにおいては、システムの脆弱性を評価する必要がある。ただし、操作性を向上することと、セキュリティを向上することは相反する場合が多い（例えば、長く複雑なパスワードはセキュリティ強化につながるが、操作性は落ちる）。したがって、お客様が求める目的機能に対して十分なセキュリティ強度を持っているかどうかをテストすることが重要になる。ラルフチャート作成時に Active Noise を考慮するのはそのためである。筆者は、今後、ISO/IEC 9126 の品質特性における主特性のレベルでセキュリティに対する脆弱性が重要視されてくるものと考えている。

Active Noise の発見方法は、情報収集が中心となる。

JPCERT/CC	http://www.jpccert.or.jp/
Secunia	http://secunia.com/advisories/
US-CERT	http://www.us-cert.gov/
Microsoft	http://www.microsoft.com/Japan/security/default.msp
Sun	http://sunsolve.sun.com/show.do?target=patchpage
ORACLE	http://www.oracle.com/technology/deploy/security/alerts.htm
Security NEXT	http://www.security-next.com/cat_cat178.html

上記のようなサイトを定期的にウォッチし、どのような攻撃方法が存在するのかについて整理しその情報を共有化することが大切である。つまり、セキュリティの専門家を作る（もしくは、そのような第三者検証機関に評価を依頼する）必要がある。

4. ラルフチャートの狙いと効果

4.1 ラルフチャートの狙い

図4は、「割り勘システム」のラルフチャートである。ラルフチャートを書く際には、因子だけでなく、水準をサンプルとして載せるとよい。水準のサンプルを例示することで、システム（目的機能）の具体的なイメージが湧きやすくなる。

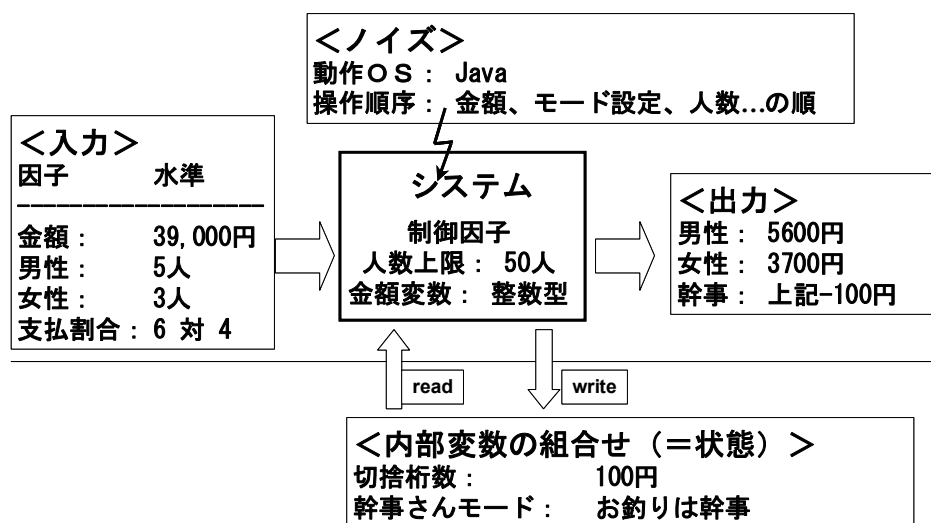


図4 ラルフチャートの実例

このように、ラルフチャートを書くことによって、「複数人の宴会の支払いを幹事が速やかに合理的に実施するために、金額や人数などから支払い金額を計算する」という目的機能に対する要因が明示される。中央にあるシステムの枠内にある制御因子とは、ソフトウェア開発者（もしくはメーカーの保守担当エンジニア）が決める因子であり利用者は変更することができない。

したがって、制御因子は、テスト時にテストするというよりも仕様のレビュー時に検討すべき項目である。ただし、テスト設計を実施する際にこれら制御因子の固定水準がどうなっているのか意識することは非常に重要である。

前回、FV表の解説において、FV表は、機能仕様書と1対1対応することで、テストカバレッジを確保していると述べた。逆に言うと、テスト要求分析を実施することで、テストする部分としない部分を明らかにすることがFV表の役割であった。

ラルフチャートでは、機能仕様書と、設計ドキュメントを参照しながら、FV表でテストす

ると決定した目的機能に対してテストする因子・水準を洗い出し、抜け落ちがないかレビューすることで、網羅性を確保する。

また、ラルフチャート作成と同時に、テスト空間の感覚を得ることも重要である。「割り勘システム」の例で言えば、入力で 4 因子、状態で 2 因子、ノイズで 2 因子が発見されている。したがって、それぞれの因子が 3 水準から成り立っていると仮定すると、 $3^8 = 6,561$ 項目が、ざっくりいった組合せテストの空間となる。

ラルフチャートを作成した後に、テスト設計していくことになるが、まずは、上記、テスト空間の規模感を持つことが大切である。テスト設計を行なうことで、例えば、直交表を使用すれば L_{27} で 27 項目に、All-pair を用いれば 16~19 項目程度のテスト項目数におさまるが、その際に、上記 6,561 項目という規模と比較してどれほどの規模のサンプリングなのかを意識するのである。すぐに、目に見える効果があらわれるというわけではないが、テストの問題を上流工程の改善につなげていくためにはこのテスト空間の規模感の変化を抑えていくことが重要である。

4.2 ラルフチャートの効果

このように、ラルフチャートを書くことは、テストの要素と構造、すなわち、テストアーキテクチャーを明らかにすることにつながる。

テストアーキテクチャーを明らかにすると、テスト自体の独立性や保守性を高めることができる。また、ラルフチャートを入れ子にし、Harlan Mills の CB モデルのようにその関係を明らかにすれば、テストの全体像を図示することができる。

つまり、ラルフチャートを書くことで、テスト網羅率をあげることに、テストアーキテクチャーを明らかにし、より効果的なテスト設計につなげることができる。

本稿がテスト設計に入る前の因子・水準の抽出の向上およびテストアーキテクチャー設計に役立てば幸いである。

以上