効率的にソフトウェアの品質向上を実現するHAYST法®の適用

Application of the HAYST Method to Allow Efficient Quality Improvement of Software

要　旨

【キーワード】
ソフトウェアテスト、組合せテスト、直交表、ペアワイズ、実験計画法、タグチメソッド、品質、評価／予測

近年、ソフトウェアは、大規模・複雑化の一途をたどっている。そのために、ソフトウェアテストの割合は年々増加し、ソフトウェア開発費に占めるテスト費用の割合は50%を超えるようになってきた。ソフトウェアテストは、機能の使用条件を効率よく選択し、その組合せや順序を設計することがポイントである。米国ではソフトウェアテストを効率化するために、ここ数年“Combinatorial Testing”（組合せテスト）が注目を浴びている[1)]。実は、組合せテストは直交表のソフトウェア評価への活用という側面において日本で生まれ、タグチメソッドの創始者である田口玄一博士によっても研究が進められ、1980年代日本は世界のトップランナーであった[2)]。本稿で述べるHAYST法は1972年から継続的に田口玄一博士の指導を受けてきた富士ゼロックス独自のソフトウェア組合せテスト技法である。本技法はツール化され、富士ゼロックスのみならず、数十社への適用実績がある。

Abstract

【Keywords】
software testing, combinatorial testing, orthogonal array, pairwise, experimental design, Taguchi methods, quality, evaluation/estimation

Software today is becoming increasingly large-scale and complicated, thus causing the proportion of costs spent on software tests to increase each year, so that it now exceeds 50% of total software development costs. When designing software tests, it is critical to efficiently select the conditions of use of functions and the combinations and order in which they are to be tested. Recently, combinatorial testing methods have gained attention in the United States as a way to improve the efficiency of software tests.[1)] In fact, it was in Japan that these methods were first applied to software evaluation using an orthogonal array, and Dr. Genichi Taguchi, developer of the Taguchi methods, also researched them, making 1980's Japan a pioneer in the field.[2)] The HAYST method described in this paper is Fuji Xerox's own combinatorial software testing method which was created based on the continuous guidance of Dr. Taguchi since 1972. This method has been made into a tool which has been adopted by many other companies in addition to being used by Fuji Xerox.

執筆者

秋山　浩一（Kouichi Akiyamai）

ソリューション・サービス開発本部 ソリューションプロジェクトマネジメント部
（Solution Project Management, Solution Service Development Group）

1. はじめに

1980年代に始まった、LSIやマイコン技術の革新にともない、これまでハードウェアで実現していた機能が次々とソフトウェアに置き換わるようになっている。このハードウェアからソフトウェアへの流れは止まらない。なぜなら、ある機能をハードウェアで実現しようとした場合は、その部品を開発もしくは調達する必要があり、商品の原価を押し上げ利益を減少させる。ところが、もしその機能をソフトウェアで実現すると多少の記憶容量は必要となるかもしれないが、基本的には原価が増えないからである。

たとえば、複写機においても20年前の商品の「拡大・縮小コピー機能」はレンズを利用したものが主流であったが、現行機ではソフトウェア処理によってレンズは不要となりその分の原価が下がっている。市場導入後もレンズ故障が発生しないため、故障対応に要する運用コストも下がっている。

このような流れの中でソフトウェアは大規模化・複雑化している。富士ゼロックスの主力商品である複合機においても1990年代に100万行程度あったソフトウェア（プログラムのコード行数）が約20年後の現在は1,600万行になり、16倍も増加している。

ソフトウェアの規模が大きくなるにつれて携わる開発者の人数も増加する。そこで、多くの場合、ソフトウェアのモジュール分割を行うとともに、開発組織を分割している。

組織は情報伝達リスクを最小化することを目的として、商品のアーキテクチャーを反映した構造にすることが多い。

しかし、商品アーキテクチャーは、たとえ初期に、疎結合で凝集度が高いように定義しても開発を進めていくうちに例外対応が増えることで次第に歪みが生じ、市場導入時期には当初の商品アーキテクチャーは影も形も失い、雑然としたものに劣化してしまうことが多い。

実際に1990年代における富士ゼロックスのソフトウェア開発の悩みは、「1. 全体を理解している技術者がごくわずかしかいない」、「2. そのため、設計レビューが十分に行えない」、「3. 開発プロセスの改善を目的としてCMM®（能力成熟度モデル）やISO9000を導入したがプロセスが重くなった割に問題は収束しない」、「4. テストで問題を検出しようとシステム検証組織を新たに立ち上げて、強化したが、“想定外”の不具合（組合せバグ*1）の撲滅には至らない」といったものであった。

2. HAYST法の概要

2.1 HAYST法が対象とする問題

ソフトウェアのテストはソフトウェアやハードウェアを含めたシステムの品質を保証するための手段の一つである。

ソフトウェアのテストには、不具合をピンポイントに狙うものと、網羅的に狙うものがある。

ピンポイントとは、たとえば成人か未成年かの判定を行うソフトウェアがあったとする。このソフトウェアへ、19歳と20歳という2つのデータを与えて動作結果をテストする方法を境界値分析と呼ぶ。これがピンポイントで不具合を狙う方法（検出型のテスト）である。

網羅的とは、たとえばプログラムのすべての命令文のうちどれだけ実施したかをパーセントで示し、100%になるまでテストを追加したり、プログラムの分岐をどれだけ通ったかをテストしたりする方法（網羅型のテスト）である。網羅型とはソフトウェアを何らかの切り口で見てその切り口でモデル化し、そのモデルに対する確認割合でテストの十分性を示す方法である。

上記でいえば、プログラムを「命令文の集まり」という切り口や「分岐の集まり」（プログラムフロー）という切り口でモデル化しているわけである。したがって、網羅型では「どういうモデルを想定したか」がポイントとなる。そして基本的に想定したモデルで検出可能なバグの種類は決まる。どんなに命令網羅や分岐網羅の網羅率を高めたとしても“（要求から仕様を作るときの）仕様抜けの問題”は見つからない。

もしも、「不具合が発生しそうな場所（境界値、分岐、リソース枯渇など）」や「動作保証すべきもの（重要な業務シナリオ、安全、財産にかか

*1 組合せバグ：それぞれの機能を単独で動作させたときには問題はないが、複数の機能を同時に（あるいは特別な順序で）動作させると発生する不具合のことをいう

わる部分など)」がテスト前にわかっているのならば検出型テストが効率的である。したがって、テストを作るときには、まずは、検出型テストを設計すべきである。検出型テストは、デシジョンテーブル*2で表現すると条件と動作が明確に表現できるのでよい[3)]。

しかし、「1. はじめに」に書いたとおり、1990年代における富士ゼロックスのソフトウェア開発の悩みは“想定外”の不具合（組合せバグ）の検出であった。この“想定外”の不具合の発生条件を検出型で想定することはできない（矛盾となる)。

HAYST法は「組合せバグ」を検出することを目的とした網羅型のテスト技法である。HAYST法が検出するバグのタイプは次の5つである。

① 仕様として想定していなかった組合せ
② 母体バグの顕在化（派生開発において追加した部分から呼ばれる既存のプログラムコードバグの顕在化）
③ 多様な環境との組合せ問題
④ ノイズ（意地悪条件）下での動作。多様なお客様の動作環境
⑤ エラーの組合せ（複数のエラーが重なった場合）

2.2 HAYST法

2.2.1 ソフトウェアテストの本質的課題

ソフトウェアの不具合は、ハードウェアの故障とは異なり、設計問題である。設計問題ということは、リリース直後のソフトウェアでも同じ不具合を再現することができるということである。

一見、当たり前のことを言っているようであるが、ハードウェアの不具合は、製造時の組み立てミス、経年劣化、部品の品質ばらつきなどによって発生する。

英語では、ハードウェア不具合要因（欠陥）のことをfault（運用中に発生した欠陥、たとえばサビなども含む）と呼ぶ。ところが、ソフトウェアの不具合要因（欠陥）のことはfaultとは違うdefect（商品にはじめから存在した欠陥）という用語を使用して区別している。

ハードウェアの場合は、いかに機能が商品購入直後のように働き続けるかの信頼性や性能の維持が重要である。ところが、ソフトウェアの場合は商品購入直後に動作している機能は10年たとうが100年たとうが同じように働く（動作条件が同じであれば同じ性能で動作する)。このように考えると、ソフトウェアのテストや不具合の撲滅はハードウェアのそれと比較して簡単に行えるように思える。なぜなら、機能を1度でも確認すればよいからである。

ところが、簡単にはいかないことはご存じのとおりである。ソフトウェアテストの本質的な課題は田口玄一氏が述べたように「工学の分野の問題でコンカレント・エンジニアリング（並行機能、同時多機能）の分野の問題」である。

また同じく田口玄一氏の言葉どおり、「ソフトウェアテストとは、コンピューターの性能ではなく目的機能を中心とする機能で、品質工学としては、多信号に対する機能の正しさの評価」である[4)]。

HAYST法では上記、田口玄一氏の品質工学の考えを支持している。田口玄一氏の言葉を補足すると「ソフトウェアテストとは、コンピューターの性能（機能の能力をさまざまな側面から数値化したもの。ハードウェアでは、たとえば車の加速力＝加速性能）ではなく目的機能（利用者が求める価値）を中心とする機能で、品質工学としては、多信号（多入力・多状態・多動作条件などの組合せ)に対する機能の正しさ(正しく働くか否か）の評価」となる。

端的に述べればソフトウェア開発中に、ある機能（＝条件＋動作）が働くことは開発者がプログラミング直後に確認する。しかし、すべてのお客様環境下で、どのような入力値が来ても期待する働きをするかどうかは開発者ではなく、テストにより評価するということである。

逆にいえば、「条件」さえわかればすべての不具合は再現できる。ソフトウェアの不具合対応がテレフォンセンターで、「動作環境、操作手順、入力したデータ」を聞くことで対応できるゆえんである。

ソフトウェアテストは上記「(不具合の発生）条件」を見つける活動である。

*2 デシジョンテーブル：入力と原因の組み合わせと、それに対応する出力と結果をまとめた表

2.2.2　HAYST法で“想定外”の不具合を検出できる理由

前項で、ソフトウェアのテストの本質について、「不具合の発生条件」を見つけることであることを述べた。

本項では、「不具合の発生条件」を見つけることの困難さと、HAYST法の解決策について述べる。まずは、「不具合の発生条件」を見つけることが困難な理由を整理する。

① 入力データの中には多くの値を持つものがある。たとえば、年齢の場合、0歳から120歳まで（としても）、1歳刻みで121の値がある。
② 入力の組合せを考えると、{OFF、ON}の2つの値（値のことを水準と呼ぶ）を持つシンプルな入力要素（これを因子と呼ぶ）であったとしても因子が2つあれば、{OFF×OFF、OFF×ON、ON×OFF、ON×ON}の4とおりの組合せになる。3つあればその倍の8とおり（$=2^3$）、10個あれは、1024とおり（$=2^{10}$）、50個あれば、1,125,899,906,842,624とおり（$=2^{50}$）と天文学的な組合せ数となってしまいそのすべてをテストできない。
③ お客様の環境は多岐にわたり特定できない。（医療機器など）特別なケースを除き環境を提供側が指定することはできない。
④ 操作の順序も因子数の階乗で増加する。（因子が7個あれば操作順序は7!＝5,040パターンとなる）

このように、「多信号（多入力・多状態・多動作条件などの組合せ）に対する機能の正しさ（正しく働くか）の評価」をすべて確認しようとすると数十個程度の機能数であっても、すぐにテスト項目数が天文学的数字となりテストしきれないという問題が発生する。

そこで、HAYST法では、②、③の組合せという切り口や、④の操作順序という切り口に対して、単純に全組合せ・全順序の網羅性を取るのではなく、全体を表現する組合せモデル（や順序モデル）を考えて、それらのモデルを代表し、全体を被覆する部分集合（全体を広く覆うモデル）を考えて、その部分集合をどれだけテストで網羅したかを計測することとした。

HAYST法では全組合せの被覆モデルとして「任意の2因子間の全水準組合せ」ともっと細かく組合せを網羅したいときには、「任意の3因子間の全水準組合せ」を、全順序の被覆モデルとして「任意の3因子間の全順序」ともっと細かく順序性を網羅したいときには、「任意の4因子間の全順序」を考えた。そして、その実現手段として、前者について直交表、後者についてシーケンスカバリングアレイ[*3]を用いている[3)]。

このように、HAYST法ではピンポイントに不具合を狙うのではなく、組合せモデルと順序モデルによって網羅的にテスト対象のソフトウェア全体を被覆することで不具合を検出する。さらに網の目の大きさもテストの目的に合わせて細かくすることができる。

以上のとおり、HAYST法は網羅的に全体をテストエンジニアなどの意図によらず被覆したテストであり、使い方ほかを想定するテストではないため、網の目よりも大きな（つまりは重篤な）“想定外”の不具合を検出できる。かりに、網の目を小さくすれば軽微な“想定外”の不具合も検出できるが、テスト数が増加するため費用対効果の検討が必要である。ミッションクリティカルなソフトウェアや重要インフラを支えるソフトウェアの場合はさらに網の目を細かくする必要があるかもしれない。

2.2.3　HAYST法の特徴

HAYST法の特徴は次の3点である。

① 多因子・多水準を取り扱う直交表
② 複雑な禁則関係を簡単に指定可能
③ 因子・水準の選定方法を確立

まず、①の「多因子・多水準を取り扱う直交表」であるが、HAYST法以前の直交表によるソフトウェアテストは、すべての因子の水準数が同じ「n水準系直交表（素数系直交表）」（nは、2, 3, 4（$=2^2$）、5, 7, 8（$=2^3$）、9（$=3^2$），…のように素数もしくは、素数のべき乗を基にした直交表）、2水準と3水準の因子が

*3 シーケンスカバリングアレイ：すべての操作順序をテストするのではなく、任意の数個の操作を取り出したときの順列で被覆するテスト方法

表1　L_4直交表
L4 orthogonal array

No.	A	B	C
1	a1	b1	c1
2	a1	b2	c2
3	a2	b1	c2
4	a2	b2	c1

混合している「混合系直交表」（L_{12},=2^{11}、L_{18}=$2^1\cdot3^7$、L_{36}=$2^{11}\cdot3^{12}$、L_{108}=$2^{11}\cdot3^{48}$）のみであった。

表1にいちばん小さなL_4直交表を示す。

組合せテストは、因子・水準の数に適した直交表を用意し、その表に組合せテストをしたい因子・水準を入れ替える（この入れ替え作業を「因子・水準を直交表に割り付ける」と呼ぶ）という方法で組合せ表を作る。

例として、表2に、車の保険加入時に伝える「車種｛普通車、軽自動車｝」、「登録年月｛2013／11、不明｝」、「用途｛家庭用、業務用｝」について割り付けた結果を示す。

表2を見ると、No.1～No.4までの4組の組合せ（それぞれをテストケースと呼ぶ）の中に、「車種×登録年月」、「登録年月×用途」、「車種×用途」のそれぞれにおいて、その水準の組合せ、すなわち、「車種×登録年月」でいえば、「普通車×2013／11、普通車×不明、軽自動車×2013／11、軽自動車×不明」の4つの組合せがすべて出現している。

したがって、このNo.1～No.4までの4組を入力したテストを実施すれば、たとえば、「軽自動車で登録年月が不明のパターンを想定せずテストし忘れる」といった「“想定外”の2機能組合せのテスト漏れによる不具合」がなくなる。なお、3因子のすべての組合せは2^3の8とおりであり、L_4直交表を使用することで4テストケースに半減している（大きな直交表ではさらにテストケース削減効果は大きくなる。たとえば、L_8直交表では、2水準の因子を7列取ることができる。したがって、単純な全組合せは2^7=128とおりになるが、直交表は8行であるからテストケースは全組合せの1／16になる。L_{16}では、2^{15}=32768とおりが、16行なので、1／2048である）。

表2　L_4直交表（割り付け済）
L_4 orthogonal array with factors and levels assigned

No.	車種	登録年月	用途
1	普通車	2013／11	家庭用
2	普通車	不明	業務用
3	軽自動車	2013／11	業務用
4	軽自動車	不明	家庭用

このように適切な直交表を用意すれば効率よく少ないテストケース数で組合せテストを実施できる。しかし、先に述べた素数系直交表については、3水準系直交表以上では、2水準の因子列が存在しないため、チェックボックスのようなOFF／ONの2値を水準に持つ因子が多いソフトウェアテストでは使用しづらかった。

また、混合系直交表は、2水準の因子列と3水準の因子列しか存在しないので、用紙サイズ｛A4、A3、A5、B4、B5、B6、Letter、Legal、はがき、往復はがき、封筒角形20号、封筒角形2号、封筒洋形2号、封筒洋形3号、封筒洋形4号、ユーザー定義用紙…｝といった多水準を値に持つ大きな因子を割り付けることができないという問題があった。

そこで、HAYST法では、素数系直交表や混合系直交表をそのまま使わずに、2水準系直交表を基に多水準の因子がある場合は複数の因子列を1つの列に併合するという方法を取った。

列を併合することで多水準列を作るテクニックは、1950年代に実験計画法で田口玄一氏によって線点図と共に考案されたが、その後、田口玄一氏の品質工学では、ハードウェアのロバスト設計が解決したい課題の中心となり（実験計画法のテクニックは不要となり）、品質工学に適した混合系直交表のL_{18}=$2^1\cdot3^7$および、L_{36}=$2^{11}\cdot3^{12}$直交表のみが使われるようになったため、『忘れ去られた技術』であった。

2つめの②「複雑な禁則関係を簡単に指定可能」についてはHAYST法の核心（特許）技術である。まず、禁則とは「複数の因子を組み合わせて使用する際に実行できない水準の組合せ」のことである。たとえば、「エアコンで送風時に温度設定はできない」というような禁止された組合せ規則のことである。ハードウェアの場合、このような禁止したい組合せ規則があったとし

てもなかなかハードウェアのみでは実装が難しい。たとえば、「（車で）20km／h以上の速度で走行中はドアを開くことができない」という禁則をハードウェアのみで実現することは困難である。ところが、ソフトウェアの場合は、この禁則を積極的に利用して、処理プログラムに意味のない入力が入らないように、入力（ユーザーインターフェイス）で、「処理結果が出ない意味のない組合せ」を排除し、処理自体をシンプルにすること、そして、その意味のない組合せをテスト不要にすることを行っている。

たとえば、プリンターの印刷指示GUI（グラフィカル・ユーザーインターフェイス）の多くは、「白黒印刷のときにカラーの設定はできない」となっている。

ここで、禁則があることを無視して直交表に割り付けを行うと、（直交表は任意の2因子を取り出したときに、その水準組合せをすべて出すため、禁則組合せが多数入った表になってしまう。禁則があると、その行はテストができない（動作しない）ため具合が悪い。そこで、HAYST法では、相互排他因子融合手法、多層化手法、可変因子手法の3つの禁則回避アルゴリズムを考案し、禁則が現れないように禁則を回避しながら直交表に因子・水準を割り付けている[5]。

禁則のルールを仕様化・ツール化する方法は、現在も研究対象となっているほど難しい。また、禁則が複雑に絡み合うと禁則回避処理が著しく遅くなる（あるいは、処理が終わらずにアボートしてしまう）アルゴリズムもある。

HAYST法では、禁則マトリクスというシンプルでかつ直感的な方法を考えた（2.3.2項参照）。また、これまで10年以上、複雑な禁則でマトリクスの生成処理が終わらないということは発生していない（なお、HAYST法ツールでは一度アルゴリズムを見直し、禁則回避処理の高速化を実施し、その結果、最大で100倍以上のパフォーマンス向上を実現した）。

3つめの③「因子・水準の選定方法を確立」は、HAYST法では、因子と水準の選び方の標準手順があるということである。詳細な手順は書籍[6]にゆずり、本稿では、「因子・水準の選定の重要さ」と、「因子・水準の選定プロセスの概要とポイント」について述べる。

まず、「因子・水準の選定の重要さ」であるが、こちらは、どんなに強調しても強調しすぎることはない。なぜなら選定に漏れた因子・水準はテストされないか、テストされたとしてもデフォルトの水準のみだからである。

たとえば、スマートフォンのテスト環境で、「移動中｛移動していない、徒歩、車、在来線、新幹線｝」という因子が抜けたら｛徒歩、車、在来線、新幹線｝という条件の組合せテストは実施されない。これは、上記の水準での組合せ網羅率はゼロ％であることを意味する。したがって、どんなに素晴らしい組合せマトリクス生成ツールがあったとしても、因子・水準の見落としがあっては意味がない。実際にHAYST法などを用いた組合せテストを実施後に残るバグのほとんどは、組合せ網羅率が低い（網が粗い）ことではなく、因子・水準の見落としが原因である。以上が、「因子・水準の選定の重要さ」の理由である。

次に「因子・水準の選定プロセスの概要とポイント」について述べる。従来は、因子・水準を見落とさないためには、商品や業務の知識が豊富である必要があると考えられてきた。ところが、第1章で述べたように、「全体を理解している技術者（エキスパート）がごくわずかしかいない」、「そのため、設計レビューが十分に行えない」という状況であった。

そこでHAYST法では、因子・水準を選択するときの属人性をできるだけ排除し、エキスパートの知見（所要時間）を必要最小限とするプロセスを目指した。

HAYST法の因子・水準の選定プロセスは次の5ステップから成り立っている[6]。

① 6W2H
② ユーザーストーリー
③ FV表
④ ラルフチャート
⑤ FL表

①の6W2Hでは、テスト対象と、テスト目的の両面からテスト観点の詳細化を行う。テスト対象としては、Why（要求）、What（仕様）、How to（設計、コード）の2W1Hの詳細化を行う。また、テスト目的のほうは、When（い

つ)、Where（どこで)、Who（誰が)、Whom（誰のために)、How much（いくらで）という4W1Hの詳細化を行う。2W1H+4W1Hで6W2Hである。

②のユーザーストーリーでは、仕様をユーザーの価値に着目して分割する。分割がうまくいったかどうかはINVEST（独立か、交渉可能か、価値があるか、見積り可能か、適切な大きさか、テスト可能か）で評価する。

③のFV表では、目的機能と、検証条件と、テスト方法を明らかにする。このとき、目的機能はユーザーストーリーの書き直しでよいが、検証条件とテスト方法の導出が難しい場合は④のラルフチャートを作成する。③、④が終わる時点で因子がすべて選定される。最後に⑤のFL表で、水準の確定を行う。

このように、因子・水準の選定プロセス（＝テスト分析･テスト設計プロセス)を標準化し、実直に実施することで、そのあとのエキスパートを交えたテスト設計レビューで、短時間で有効な指摘ができるようになる。プロセスが人ごとにバラバラだと、思考過程の共有に時間がかかり「その因子・水準を選んだ理由」についてスムーズなレビューができない。

2.3 HAYST法ツール

前節では、HAYST法の意義と概略について述べた。本節ではHAYST法をテスト業務に適用するために開発したHAYST法ツール（MatrixTester®と呼ぶ）について述べる。

筆者は当初、Excel®を使用してテストケースを作成していた。しかし、禁則の回避処理や網羅率の計算に手間がかかりすぎた（L_{128}直交表で収まる程度のテスト規模で禁則回避を含めて作業では5日程度かかった)。Excelでは時間がかかりすぎるためHAYST法の横展開ができなかった。そこで、Visual C++を使用したWindows®で動作するツールを開発した[7),8)]。

2.3.1 GUI

MatrixTesterのGUI（図1）の設計においては標準的なWindowsアプリケーションの作法に則るとともに、ModelとViewの分離独立性を強く意識した。

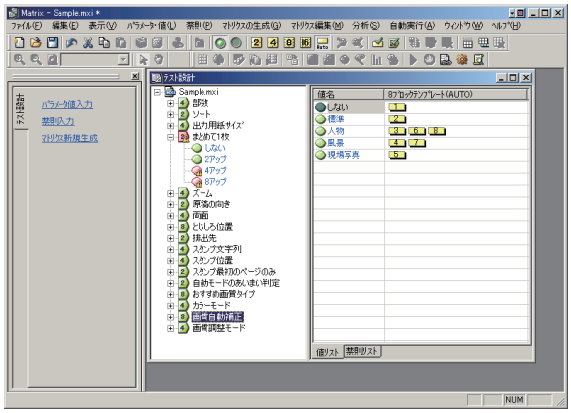

図1 MatrixTesterのGUI
The MatrixTester GUI

その結果、因子･水準名の変更などに対して、それを1カ所で行えばほかのすべてのViewに反映されるといったテスト設計の再利用が進む効果が得られた。

2.3.2 禁則の設定

MatrixTesterの禁則の指定方法について述べる。図2、図3のように、因子間の禁則条件は、その総組合せ表に入力する。入力した禁則を解析し、相互排他因子融合手法、多層化手法、可変因子手法の3つの禁則回避アルゴリズムから適切なアルゴリズムを自動で選択し、禁則回避処理を実行している[2)]。したがって、ユーザーは、禁則条件だけに着目して、その情報を整理すればよく、禁則回避アルゴリズムの知識を必要としない。

図2 2つの因子の禁則関係の指定画面
Screen for specifying constraint relationships between two factors

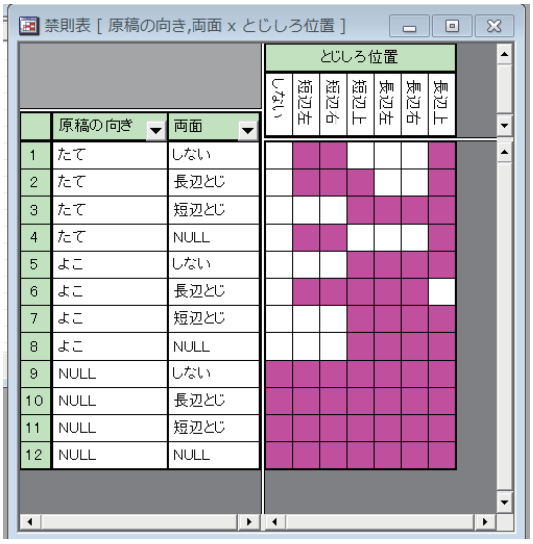

図3　3つの因子の禁則関係の指定画面
Screen for specifying constraint relationships among three factors

2.3.3　テストの重みづけ

因子・水準の組合せ表を生成するときに、

① 重要な因子
② 重要な水準
③ 重要な組合せ

をテストマトリクスへ反映して、必要十分なテストを実施したいという要求がある。

MatrixTesterでは、①の「重要な因子」について、重要な因子をグルーピングして因子群を作り、その因子群だけ組合せ強度を上げる機能を持っている。（任意の2個の因子の全水準組合せ　→　任意のk個の因子の全水準組合せへ）

②の重要な水準については、ダミー水準で調整する。③の重要な組合せについては、事前に重要な組合せを指定しておくことによって、直交表に割り付け後、その重要な組合せについて自動的に追加できる機能を持っている。

3. HAYST法適用結果

HAYST法は、1999年に手法を確立し、2000年4月に最初のバージョンのツールを開発し社内利用を始めた。社内では初めにカラー複合機のプリンタードライバーとカラー複合機の組み込みソフトへ適用し、テストケース数の半減と組合せ網羅率の3倍を同時に実現し、適用した領域で市場不具合ゼロを達成した。その結果について、2003年に品質工学会で発表した[8]。その後、業務ソリューションシステム案件へ適用し、組み込み系以外での活用事例について社外発表を続けた[9]。

社外発表を聞いたお客様から「自社でもHAYST法を試してみたい」とのお話を頂戴し、2004年から社外の会社へコンサルティング形式で支援を続け、2014年にツールの商品化を行った。現在までに、100社以上にご紹介に伺い、興味を持っていただいた。

そして、多数のお客様にツールやコンサルティングを提供してきた。提供先は、電機メーカー、精密機器メーカー、医療機器メーカー、自動車関連メーカー、保険業、通信機器、アプリケーションソフトウェア開発会社等々、業種を問わず活用いただき効果を上げている。お客様の共通の悩みはテストの効率化と市場不具合の撲滅であり、HAYST法を適用することで単機能のテストは当然として組合せの視点でどのようにテストすべきかの議論が進み、テスト担当者のモチベーション向上にも寄与した。

4. HAYST法ツールの商品化

4.1　商品化

2004年からお客様の要求に対応する形でHAYST法のコンサルティングとツールをお客様へ提供してきた。その後、2014年にツールの商品化および富士ゼロックス総合教育研究所にてHAYST法の集合教育の定期開催を開始した[10]。

4.2　社外適用の結果

さまざまな会社に適用することで、社内適用だけでは気付くことのできなった組合せテストへの要求について理解し、ツールへエンハンス機能として反映することができた。

4.3　今後

お客様のさまざまなテスト現場の声を聞き、状態遷移モデルについての対応の必要性を感じ、PathCombo法というテスト手法を考案し、適用し成果が認められた。

PathCombo法とは、状態遷移を、パス（Path：状態遷移の操作・処理の流れ）と組合せ（Combination：状態遷移を引き起こすイベ

ントの組合せ）に分けてモデル化し、前者に1スイッチテスト、後者にHAYST法の水準集約法を使う方法である。有則（パス）を漏らさずにすべてテストし、無則（組合せ）を直交表で網羅することで、効果と効率の両立を実現している。本手法は医療機器のテストへ適用し重要な不具合の検出を実現している[6)]。

これをソフトウェアシンポジウム（2013）で発表したところ最優秀論文賞を受賞し、書籍にも解説文を書いた[6)]。今後、PathCombo法を手軽に行えるようにツールをエンハンスしたいと考えている。

5. おわりに

約15年にわたり、社内外のソフトウェアテストの改善活動を通してHAYST法を進化させてきた。多数のお客様に適用し、100社以上のテスト技術者と対話をする中で、ソフトウェアテストにおいて「組合せ」と「順序」の条件出しに苦労していることを実感した。

また、HAYST法によってその多くの問題が解決され、「安心して商品を市場導入できる」、「プロジェクト後半になっても見通しがよくなり残業しなくて済むようになった」との声を頂いている。

近年、米国でも、“Combinatorial Testing”（組合せテスト）が注目を浴びてきており、方向性が正しいとの確信を深めている[1)]。今後も、HAYST法の普及を通じソフトウェアテストの効果と効率の向上に寄与していく。

6. 商標について

- CMMは、カーネギーメロン大学によって米国およびその他の国における登録商標または商標です。
- Excel、Windows、は、米国Microsoft Corporationの米国およびその他の国における登録商標または商標です。
- HAYST法およびMatrixTesterは、富士ゼロックス株式会社の登録商標です。
- その他の商品名、会社名は、一般に各社の商号、登録商標または商標です。

7. 参考文献

1) D. Richard Kuhn: Introduction to Combinatorial Testing, Chapman and Hall/CRC (2013).
2) 田口玄一: 品質工学便覧, 日刊工業新聞社, (2007).
3) 秋山浩一: ソフトウェアテスト技法ドリル, 日科技連出版社, (2010).
4) 田口玄一: “信号因子に対する目的機能の評価”, 標準化と品質管理 Vol.52, No.3～No.7, (1999).
5) 吉澤正孝, 秋山浩一, 仙石太郎: ソフトウェアテストHAYST法入門, 日科技連出版社, (2007).
6) 秋山浩一: 事例とツールで学ぶHAYST法, 日科技連出版社, (2014).
7) K. Akiyama, T. Takagi, Z. Furukawa: “Development and Evaluation of HAYST Method Tool”, New Trends in Software Methodologies, Tools and Techniques. , Proc. of the 9th SoMeT_10, pp.398-414, (2010).
8) 秋山浩一: “HAYST法の戦略的組織的展開と開発現場での効果実績”, 2003年度品質工学会研究会発表, (2003).
9) 秋山浩一, 高木智彦, 古川善吾: “直交表を用いたソフトウェアテストにおける効果的な因子選択・割り付け手法”, 品質, Vol.42, No.4, (2012).
10) HAYST法®ツールを活用したソフトウェア品質向上支援ソリューション: http://www.fujixerox.co.jp/solution/management/hayst.html

筆者紹介

秋山　浩一
ソリューション・サービス開発本部 ソリューションプロジェクトマネジメント部に所属
専門分野：ソフトウェアテスト、信頼性工学、品質工学