ソフトウェア工学 96−15
（1994．1．21）

再試験を可能とするソフトウェア自動試験環境

金子 斉　立元 慎也　白石 智　黒田 憲一
NTT交換システム研究所

本稿では、まず、試験プロセスを形式的に記述し、これを自動的に解釈／実行することによる効果を示し、その実現環境について述べている。ついで、既に記述された試験シナリオを、デグレードチェック時などに異なる試験条件下で再実行する『試験シナリオの再利用』に着目し、これを実現するための試験シナリオの記述法および自動試験システムの構成法について述べている。さらに、再試験時に必要な試験シナリオを検索するデータベースの構成を提案している。

Automatic Testing Environment for Retesting

Hitoshi KANEKO　Shinya TACHIMOTO　Satoshi SHIRAISHI　Ken-ichi KURODA
NTT Communication Switching Laboratories

This paper proposes methodologies for constructing an automatic testing environment for retesting. First, the effect of formally describing the testing process as a testing scenario and automatically interpreting it. Then, the methodologies for describing the testing scenario and constructing the environment for realizing scenario reusing, which is efficient for retesting, e.g. degrade-check. Furthermore, we propose structure of a data-base which searches testing scenarios for retesting.

1．はじめに

一般にソフトウェア開発は要求分析、設計、製造、試験、保守運用といったプロセスに分類されるが、その中でも試験プロセスは多くの工数を要し、このプロセスの品質および生産性がソフトウェアの品質を左右するといっても過言ではない。

交換ソフトウェアでは、試験プロセスにおいて多くの特殊な試験リソースを要する。ここでいう試験リソースとは、試験者、試験対象、試験ツールなどの総称と定義する。試験リソースの利用可能な量は限られており、また常に同じ条件で利用できるとは限らない。さらに、交換ソフトウェアのライフサイクルは十数年と一般のものに比べてかなり長く、その間には、数多くの機能追加がなされ、その度に新機能の試験および既存機能のデグレードチェックが必要になる。こうした点から、交換ソフトウェアにおける試験リソースの有効利用および試験プロセスの効率化は重要な課題といえる。

また、交換システムのような大規模システムでは、その試験項目数は膨大（数万のオーダー）であり、この中から再利用したい項目を様々な基準に基づいて効率的に検索することも生産性向上に向けた大きな課題と言える。

筆者らは、このような観点から試験プロセスの形式的記述と自動化に関して検討を行い、試験項目の実施内容を形式的に記述した試験シナリオに基づいた自動試験システムの試作[1)]を行ってきた。本稿では、上記のような交換ソフトウェアの試験プロセスの特徴から、既に記述された試験シナリオを、デグレードチェック時などに異なる試験リソース条件下で再実行する『試験シナリオの再利用』に着目し、これを実現するための試験シナリオの記述法および自動試験システムの構成法について述べる。また、再試験時に必要な試験シナリオを検索するデータベースの構成および検索ロジックを提案する。

2．試験自動化へのアプローチ

2.1 試験プロセスの形式的記述

試験の各工程における作業（試験プロセス）は、試験項目の抽出・手順書作成・試験実行・結果解析等のサブプロセスに分解できる。サブプロセスは、さらに詳細なプロセス、たとえば試験実行の場合、条件設定・実行・結果収集と分解できる。このような手順を繰り返し、プロセスの構造的な詳細化を図る。この過程で、問題解決部分を人間に、形式変換部分をツールに分担させることで、自動試験へ向けてのプロセスが導出できる（図1）。形式変換部の最終的な出力は、試験項目毎に試験者の活動を記述したもの（試験シナリオ）となる。

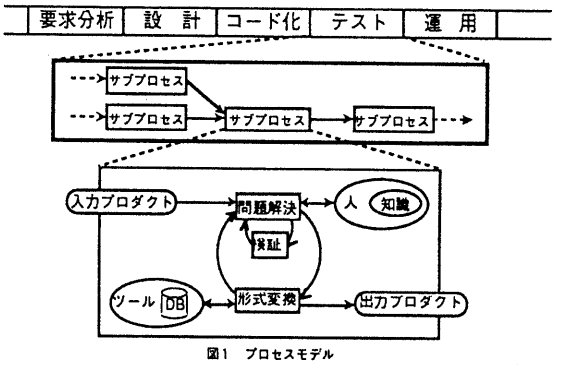

図1 プロセスモデル

2.2 自動試験システムの構築

このように試験シナリオを記述し、インタプリタによってこれを解釈実行する自動試験システムを構築することで、試験の自動化が実現され、試験実行の作業効率の向上が図られる。

自動試験システムは、Unix+ネットワーク環境の上に、”試験の進行を制御する部分”と”試験内容（個々のツールが行う動作）を制御する部分”の2階層から構成される（図2）。前者をIntegration Platformと呼び、試験者が記述した試験シナリオを解釈し、逐次実行するインタプリタとなる。このとき、バックエンドにて配下の様々なツールの環境の違いを吸収し、連動するツール相互間のタイミングを制御する等の協調動作を可能としている。また、各ツールの実行状態に基づき、次動作への移行やツール初期設定等のエラー処理を行う。

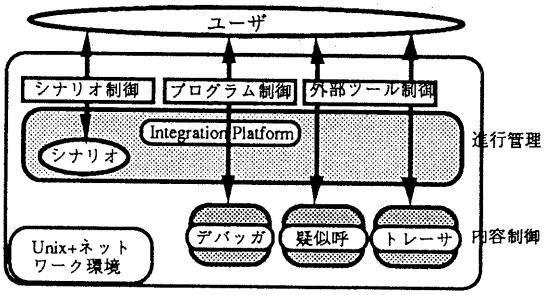

図2 自動試験システム構成

2.3 シナリオ記述言語

試験プロセスを試験シナリオとして形式的に記述するためには、記述手段としての形式的な言語を定める必要がある。この言語は、ツール間のインタフェースに応じて3種類のコマンド群に分類される（表1）。

試験の進行の制御を行うコマンド（シナリオ制御コマンド）は、Integration Platformに対してのみ有効となり、シナリオの実行シーケンスを制御する。個々のツールには、試験対象プログラムへ直接操作を行うツールと、それ以外の外部ツールとがある。前者の操作は、ソフトウェア試験という観点からは共通のものであり、使用頻度も高いことから、そのインタフェースは試験者に開放してあり、ダイレクトに記述できるレベルのコマンドで行う（プログラム制御コマンド）。後者の操作は、既存ツールの活用・新規ツールの独立した発展を狙い、ツール間で共通のコマンドで試験進行レベルを制御し（外部ツール制御コマンド）、各ツール独自のコマンドパラメータおよび外部制御ファイルにあらかじめ記述された制御データで詳細な指示を行う。これによって、ツールの変更や、新規ツールの追加などが行われた場合は、パラメータや外部制御ファイルを修正／作成することで、柔軟に対応することが可能となる。

3．試験シナリオに基づく自動試験の効果

試験シナリオによる自動試験では、試験実行の作業効率が向上するだけでなく、以下のような効果も期待できる。

（1）再試験の効率化：修正箇所の確認試験、機能追加／修正時のデグレを防ぐための試験、流用部分に対する確認試験にシナリオが再利用でき、効率を上げられる。特に交換システムのような寿命が長く、機能追加が頻繁に行われるシステムでは、再試験の必要性は大であり、その効率化はシステムの信頼性、生産性向上に多大な効果をもたらすと考えられる。

（2）作業レベルの確保：標準化により、作業の最低レベルは確保でき、また、問題発生時には、残した記録が再試験指示等の有力な判断材料となる。そのため、初心者でも、シナリオさえ与えれば一定のレベルの試験は行うことが可能となる。また、形式的な試験シナリオ、試験結果を参照することで、試験の十分性、妥当性を判断することが可能となる。

（3）教育・ノウハウの継承：徒弟制度的、口承的であった試験活動のノウハウが、明示的書物として引き継がれる。新規機能追加時や、新規システム開発時などにおいて、既存の試験シナリオを参照／流用することで、ノウハウ継承の手段の充実が図られる。

表1　シナリオ記述言語（例）

I.F.種別	コマンド	概要
シナリオ制御 I.F	サブシナリオコマンド (sbt)	サブルーチンシナリオを実行するためのコマンド。このコマンドによってシナリオの部品化、階層化が可能となる。
	試験制御コマンド (proc)	試験項目毎のテスト条件（試験項目、ログ、エラー手順、タイムアウト手順、結果判定）を設定するためのコマンド。
	変数設定コマンド (var)	試験シナリオ中で用いる変数を設定するコマンド。このコマンドによってシナリオ中でのデータ内容の流用や実行回数カウントが可能となる。
	条件文 (if, switch)	試験シナリオ中で条件分岐を行なうためのコマンド。
プログラム制御 I.F.	基本コマンド (ld, g, stp etc.)	プログラムを制御するための基本コマンド　（一般のデバッガ機能に相当する。）
	データ設定／収集コマンド(e, en)	データを設定、収集するときにシンボリックに指定する。従って、プログラムが変更になっても、データ設定、収集指示は影響受けず、シナリオの再利用が可能である。
	ブレイク／チェックポイントコマンド (bsi)	要求ポイントでプログラムの実行を停止させるコマンド。物理アドレスやソース行を指定せず、ソースコード中の論理ラベル（シンボル）により示される。従ってプログラムが変更されてもシナリオは再利用可能である。
外部ツール制御 I.F.	ツール起動コマンド (!, fork)	UNIXベースのツール、子プロセスとしてのツールを起動するコマンド。本コマンドのパラメータとしてツールのコマンド列が与えられる。

（4）プロセスの改良：記録を計算機で操作できる形で残すため、再現性の向上及び工数の削減が実現され、従来、再現ができない、どこが悪かったのか調べられない、あるいは別途実験するには金と時間がかかり過ぎる等で困難であった、客観的な評価・改善活動を行える。

本論文では、上記のうち、特に大きな効果が期待される再試験の効率化に注目し、以下に検討を進めていく。

4．再試験の課題と実現法

試験を行った結果、バグが発見されると、それを修正し、再度試験を行うことになる。また、機能追加時に、既存機能の正常性を確認するデグレードチェックを行う際にも、既に試験した項目を再度試験することになる。交換ソフトウェアでは、試験項目数が膨大であるのに加え、頻繁に機能追加が行われるため、全工数のうち再試験によるものが占める割合も大きいといえる。こうした点からも、再試験の効率化が、開発プロセス全体の効率化及びプロダクトの高品質化につながると言っても過言ではない。

4.1 再試験実現に向けた課題

交換ソフトウェア開発において、試験ツールの環境条件や試験対象マシンの実装条件、ソフトウェア条件などは、再試験を行う時期や場所によって大きく異なってくる。そのため、試験リソース（試験者、試験対象、試験ツール等の総称）は再試験時に前回と全く同じ条件で用いることは不可能な場合が多い。こうした試験リソースの条件の差異をユーザに意識させずに試験が可能になれば、再試験時には既に記述されている試験シナリオがそのまま再利用可能となり、大幅な効率化が実現される。これを実現するためには、試験リソース条件に依存した情報を試験シナリオ中から排除し、試験時にそのリソース条件に応じた情報を付与する必要がある。

また、再試験時には、その必要性に応じて様々な観点から再試験項目が選定される。その際に、膨大な試験項目の中から、必要なものを検索する作業を人間が行うと多くの工数を要する。そこで、既存の試験項目の中から、所望の再試験項目を検索するために、こうした項目を管理するデータベースを構築する必要がある。

4.2 試験シナリオの再利用

4.2.1 試験リソース依存情報

交換ソフトウェアにおける試験リソース条件に依存した情報のうち、再試験に大きな影響を与えるものを表2に示す。アドレス情報、行番号情報は、ファイル改版時には毎回変化するもので、最も影響の大きいものといえる。試験ツール情報は、試験実施場所、ツールの故障による入れ替え、ネットワーク環境の変化などによって変化するもので、複数のハードモジュール上で同時に試験を行うような環境では、モジュール毎に異なってくる。モジュール情報は、通信ソフトウェアのようにモジュール間で通信を行うシステムでは通信条件によってモジュール毎に異なってくる。開発時に加え保守運用時には、試験条件を共通化することは困難であり、このような異なる条件下で同一の試験シナリオを再利用するためには、こうした試験リソース依存情報を試験シナリオ中から排除しなければならない。

4.2.2 試験シナリオの再利用法

前述のような試験リソース依存情報を試験シナリオ中から排除し、これらを試験条件毎に管理して同一試験シナリオの再利用を可能にする手法を提案する。本手法を実現するシステムの構成を図3に示す。試験シナリオ中には、アドレス情報は記述せず、すべてデータ名や関数名などのシンボル名で記述する。また、ブレークポイントなどでは、行番号を記述せず、あらかじめソースプログラム中にコメントとして埋め込んだ行番号シンボルを記述する。これらは、自動試験システム中のシンボリックデバッグ機能および行情報シンボル解析機能によって試験実施時にアドレス情報や行番号情報に自動的に変換される。また、モジュール情報や試験ツール環境情報はすべて特定の論理ラベルで記述する。これらの論理ラベルは、モジュールや試験環境毎に設定されたリソース依存情報データベースを参照してリソース管理機能によって実際のリソース情報へと変換される。

こうした機能によって、リソース条件の異なる環境においても、同一の試験シナリオが利用可能となる。

表2　試験リソース依存情報

情報種別	シナリオ中での利用箇所	依存関係
アドレス情報	プログラムが割り付けられているアドレス情報。データ指定時に用いられる。	ファイル化時にアドレス割付が変化するとこの情報も変化する。（ファイル版数依存）
行番号情報	ソースプログラム中の行番号。実行アドレスやブレークポイント指定時に用いられる。	ソース修正時に行の挿入や削除が行われるとこの情報も変化する。（ファイル版数依存）
試験ツール情報	試験時に用いるツール類の論理名称（ネットワーク上のホストネームなど）やツール毎の装置条件など。試験呼やコマンド、擬似障害などの発生指定時、データ取得／格納先指定時に用いられる。	使用ツールの変化、ネットワークの変化時にこの情報も変化する。（環境依存）
モジュール情報	システムの装置実装条件、収容加入者条件、サービス条件など。試験呼やコマンド、擬似障害などの発生対象指定時に用いられる。	各種条件の異なるモジュールで試験を行う場合、この情報も変化する。一般に通信システムでは、モジュール毎にこれらの条件は異なる。（モジュール依存）

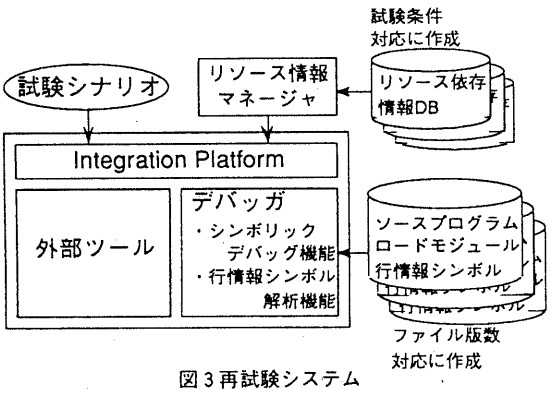

図3再試験システム

4.3　試験項目の抽出法

以下では、試験項目の抽出範囲と試験項目抽出システムについて述べる。

4.3.1　試験項目抽出範囲

既存機能の確認をするためには、既存の試験シナリオを全て実行すればよい。しかし、その数は膨大（局用交換システムでは数万）にわたるため、再試験を効率的に実施するには、できるかぎり試験範囲を絞る必要がある。本稿では、試験シナリオの数が多い場合には、試験シナリオを変更箇所とインタラクションをもつ部分とそうでない部分とに分離しインタラクションをもたない部分は試験対象から削除することで、再試験項目数を絞る方法を考察する。以下、インタラクションをもつ部分の抽出法を挙げる。

（1）変更関数を通過する試験シナリオを再試験対象とする。

例えば、図4に示すように、分析結果が1つ追加になった場合には、分析関数の変更により、既存のルートが破壊されている恐れがあるため新規ルートのみならず、既存のルートについても試験しなければならない。また、図5に示すような、信号フォーマットの変更があった場合、新規に作成された項目に対応するルートのみならず、既存の項目に対応するルートについても試験を行う。そこで、このような変更関数を通過する試験項目は、再試験対象とする。

また、関数を通過する試験シナリオを抽出したとしても、まだ、数が多い場合が考えられる。その場合、更に試験シナリオを絞り込む必要がある。そこで、if文やswitch文などのブランチ命令に着目し、関数内をいくつかのブロックに分解し、それぞれのブロックにラベルを付与することを考える。概要を図6に示す。これにより、更に通過ブロック（ラベル）により、試験シナリオを絞り込むことができる。例えば、図6のDブロックのみに変更があった場合、その関数を通過するシナリオのうちDブロックを通過するもののみ試験すれば足りる。それゆえ、試験シナリオの通過するブロックを管理していれば、変更ブロックを通過する試験シナリオ抽出することにより更に試験シナリオを絞ることができる。詳細については4.3.2に示す。

（2）変更関数を含むモジュールを通過する試験シナリオを試験対象とする。

図7に示すように、新規に作成された項目に対応するルートのみならず、既存の項目に対応するルートについても試験を行う。

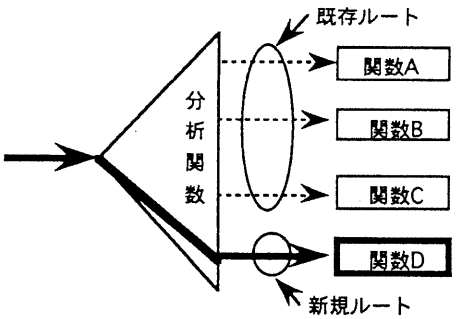

図４　分析関数の変更

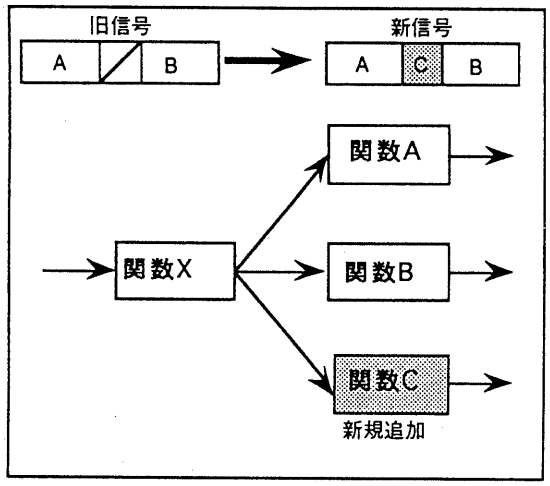

図５　信号フォーマットの変更

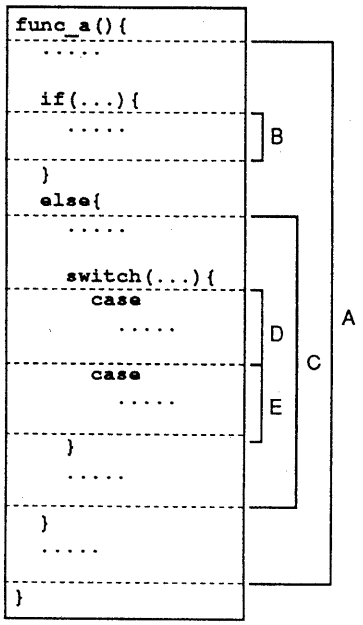

図６　ブロックの切り分け

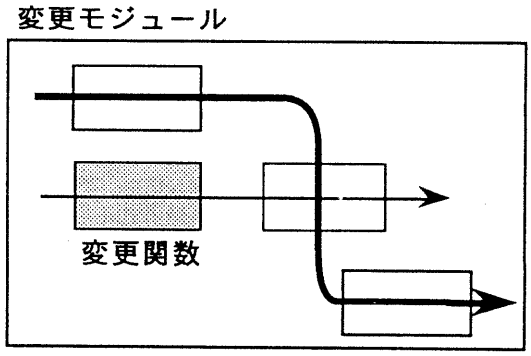

図７　モジュール内関数の変更

（３）データの構造やメモリ割付領域に変更があった場合、そのデータに対して割付上隣り合ったデータにアクセスする試験シナリオを再試験対象とする。

（４）機能追加した機能と関連のありそうな（競合機能など）試験シナリオを再試験対象とする。

単体試験や結合試験工程における試験では、基本的に試験対象システムは各項目毎に初期化され、１つの項目における実行ルートにおいてエラーがなければOKとすることで、個々機能の検証およびエラー原因の切り分けを効率的に行うことが重要である。つまり、項目毎にシステムの状態がクリアされるため、試験時の実行ルートが変更関数を通過しなければ、試験結果に変更関数が影響を及ぼすことはないといえる。そのため、単体試験および結合試験工程では、変更関数を通過する試験のみで十分といえる（4.3.1(1)）。これに対し、複数複合試験や安定化試験では、複数の項目を連続的に実施することで、機能間の影響やシステムとしての安定性を確認する。この場合は、変更関数を通過する試験項目を実施後、システムを初期化することなく連続的に、変更関数は通過しないが変更関数を含むモジュールを通過する試験シナリオ（4.3.1(2)）や、関連機能（4.3.1(4)）についても試験対象とする事が必要な場合がある。

さらに、データの構造、サイズおよび割付領域等に変更があった場合には、メモリ割付上そのデータに隣り合って割り付けられているデータを破壊している可能性があるため、このような隣り合ったデータにアクセスする試験シナリオについて再試験する必要がある（4.3.1(3)）。

4.3.2　試験項目抽出システム

試験シナリオを、機能毎、工程毎にまとめて管理することで、機能毎、工程毎の試験シナリオの抽出が容易になる。さらにここでは、4.1.1で述べた4つの抽出法を実現するシナリオ管理システムを提案する。

図8に示すようなデータベースシステムを用いて関数毎にその関数を通過する試験シナリオをまとめる。また、関数毎にその関数の所属モジュールを登録しておく。このデータベースを関数名、モジュール名で検索することによって、その関数、モジュールを通過する試験シナリオを抽出することが可能となる。（4.3.1(1),(2)）。抽出レベルをブロックまで落とした場合には、データベースにブロック名の項目を追加することになる。それにより、モジュール名、関数名、ブロック名を指定することによって、そのブロックを通過する試験シナリオを抽出することも可能となる。

また、試験シナリオから、関数名、モジュール名を検索する機能を作ることによって、試験シナリオの通過する関数名、モジュール名を抽出することができる。さらに、抽出された試験シナリオが自動的に実行されるようにシステムを構築すれば、関数名を指定するのみで、その関数、モジュールを通過する試験シナリオを自動的に実行することができる。また、試験シナリオの登録は、試験実行時に自動的に行うようにし、登録ミス、登録稼働をなくすことで、効率化を図る。

データベース内のデータの配置イメージを図9に示す。関数を通過するシナリオの数は、関数毎に異なる。例えば、交換プログラムは、レイヤの高いほうから、サービス階層、リソース階層、基本OS階層に分割される。サービス階層は、サービス毎にプログラムが異なるが、リソース階層、基本OS階層とレイヤが低くなるにつれて、各種サービスで共通に用いられるようになる。

それゆえ、サービス階層等では、そこに含まれる関数を通過するシナリオは少ないが、レイヤの低い部分、基本OS階層に含まれる関数などについては、それを通過するシナリオは多いことになる。

4.3.1の(3)については、メモリ割付表から変更されたデータに隣り合ったデータを調べ、そのデータにアクセスする試験シナリオを検索する。特に、コーディング言語やプロセッサのデータ保護機能などによって、そのデータへの影響範囲が限定される場合がある。例えば、この影響範囲が1つのモジュール内に閉じていれば、4.3.1の(2)の方式で再試験対象とされるシナリオに吸収することができる。

4.3.1の(4)については、機械的な抽出法はない。しかし、上記で述べたようにデータが1つのモジュール内に閉じていれば、新規追加機能の試験ルートが通過するモジュールのデータに着目して4.3.1の(2)の手法で試験項目を抽出することは可能である。また、この節の最初で述べたように試験シナリオが機能毎に管理する事で、「要求仕様と関連のある機能」という形で試験項目を抽出することが可能である。

所属モジュール名	関数名	ブロック名	通過試験シナリオ名
モジュール1	関数1	ブロック1	試験シナリオ1，試験シナリオ2，・・・・
モジュール1	関数1	ブロック2	
モジュール1	関数1	ブロック3	試験シナリオ1，試験シナリオ2，・・・・
モジュール1	関数2	ブロック1	試験シナリオ2，　試験シナリオ3，・・・・
モジュール1	関数2	ブロック2	
モジュール2	関数1	ブロック1	
・	・	・	・
・	・	・	・
・	・	・	・

図8　試験シナリオ管理データベース

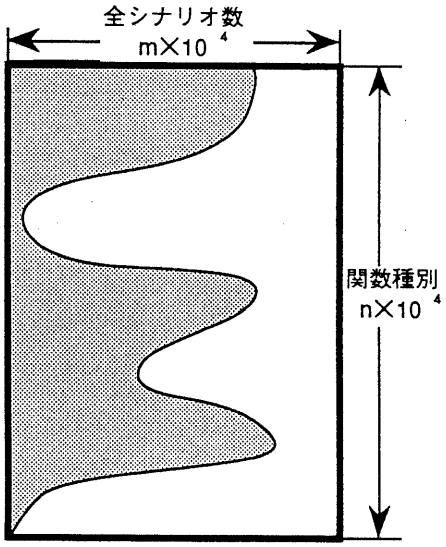

図9　通過関数とシナリオの関係

4.3.3　今後の課題

4.3.1と4.3.2で述べた方式を実際の開発に適用し、品質がどれだけ向上するか、また生産性がどれだけ向上するかの評価を行う。更に、バグの分析等を行うことにより、4.3.1と4.3.2で述べた方式の問題点を抽出する。

5.　まとめ

今回の報告では、再試験の自動実行における問題点と解決案について示した。その内容をまとめると以下のようになる。

(1) 再試験を自動的に行うためには、試験シナリオを用いた手法が有効である。
(2)試験シナリオの再利用性の問題点については、物理情報を論理化することによって対処できる。
(3)試験シナリオの抽出法については、場合に応じた抽出が必要であり、それらの殆どは機械的な自動抽出が可能である。
(4)試験シナリオの自動抽出は、関数名、モジュール名を縦軸に、試験シナリオ名を横軸にしたデータベースを構築し、それへの登録を自動的に行うことにより、ミスなく、工数もかからずに行うことができる。

参考文献

1) S. Tachimoto, S. Shiraishi : "Automatic Testing System Based on Process Programming for Telecommunication Software", GLOBECOM '93 pp.1017-1021