論 文 の 内 容 の 要 旨

論文題目　ソフトウェアシステムに対するシステムテストの設計

氏　名　西 康晴

本論文の第１章ではまず、研究の背景と目的について述べた。

ソフトウェアは、人間が開発するシステムの欠かせない要素になってきている。さまざまなシステムの中核となっているのは、ソフトウェアである。ソフトウェアの果たす役割はどんどん大きくなってきている。そしてソフトウェアは非常に複雑かつ巨大になってきた。

しかしソフトウェアの品質は決して高いとは言えない。バグのないソフトウェアは開発できない。

バグのないソフトウェアが開発できないとすると、どうすればよいのであろうか。答えは簡単である。早くバグを見つけて直せばよい。すなわち、バグを見つける作業、ソフトウェアテストを効率よく効果的に行えばよいのである。しかし、ソフトウェアテストは十分に研究されているとは言えない。ほとんどの技術者が勘と経験に基づいてテストを行っている。

ある程度製品が組みあがった状態でのテストには、さまざまな過酷な条件を設定した上で、それでもきちんと動作することを調べる「システムテスト（System Testing）」がある。

複雑な不具合はシステムテストを行わないと検出できない。そのためシステムテストはソフトウェアの品質の確保に非常に重要となる。しかしシステムテストも勘と経験によって行われているのが現状である。

本研究では、ソフトウェアシステムに対するシステムテストの効果的な設計手法を提案することを目的とする。すなわち、より早く、より少ない工数で、より危険な不具合を発生させる手法を提案することを目的とする。

第２章では、経験を積んだ技術者が持つ、システムテスト設計において着目すべき観点について、調査・分析を行った。偏りを極力抑えるために、種々の大手ベンダからテストを依頼されているテスト専門組織で３年以上経験のある技術者を調査対象とした。グループウェアからプリンタ制御ソフトウェアまで幅広いテスト対象を技術者に提示し、システムテスト項目を設計させる形式をとった。

調査の結果、12 のカテゴリで 411 件のシステムテスト項目が得られた。分析の結果、広い意味を持つさまざまな抽象度のリソースのネットワークとしてソフトウェアが構成されており、ソフトウェアの機能はそのパスに沿って連鎖的に実行されるという概念が、システムテスト設計するための観点として必要であるという結果を得ることができた。

第３章では、第２章で得られた観点にしたがい、システムテスト設計の際にソフトウェアを捉えるモデルとして、リソースパスモデル（Resource Path Model）を提案した。まずリソースを、Beizer の定義より拡張し、ネットワークの帯域なども意味することができるよう、「プログラムの実行のために自らが保持する情報をやり取りしあうオブジェクト」とモデル化した。次にリソースパスを「ソフトウェアが動作を行うために情報を連鎖的にやり取りするリソースの列」とモデル化した。

リソースパスモデルでは、リソースはさまざまな属性を持つ。DLL であれば作成日時といった属性を、ネットワークであれば帯域の広さといった属性を持つ。またリソースは、自らの持つ属性の種類で分類することができる。この分類名をリソースの「型」と呼ぶ。

リソースパスモデルに従うと、リソースの属性の変化の影響が、リソースパスを媒介とすることにより、予想と一致しない出力結果や不正な現象として、外部に露呈したものであると捉えることができる。以後リソースの属性の変化を「衝突」、衝突を引き起こす条件を「衝突条件」、衝突条件の原因となったものを「衝突対象」、衝突対象と共有するリソースを「共有リソース」と呼ぶ。

また第３章では、不具合のメカニズムをリソースパスモデルにしたがって記述した。

まず、衝突対象のインストールなどの動作によって、衝突条件が起こる。例えば、ストレステストでは大きなサイズのデータを読み込み、構成テストでは衝突対象のインストールを行う。

次に衝突条件によって、衝突が発生する。例えば、大きなサイズのデータを読み込んだら、ヒープというリソースの残り容量という属性がほとんどゼロに近い値になる。

そして衝突の発生したリソースを含むリソースパスに沿って、悪影響が連鎖的に伝播し衝突不具合として露呈する。例えば DLL が新しいバージョンに変化したら、テスト対象は起動に失敗してしまう。

リソースパスによる不具合のメカニズムを図１に示す。

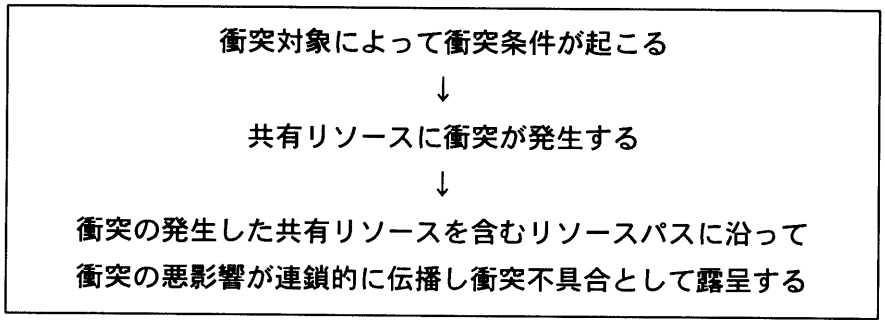


図１　衝突不具合のメカニズム

第４章では、リソースパスモデルにしたがい、新しいシステムテスト設計の方法としてリソースパステスト法（Resource Path Testing）を提案した。通常のシステムテスト設計は、経験に基づくため、いわば暗闇の中を手探りで歩くような方法である。それに対して、リソースパステスト法は、システムの構造を俯瞰する見通しの良い方法であると言える。

リソースパステスト法は、リソースパスモデルから導出された図１のメカニズムにしたがって、共有リソースに衝突を発生させることにより衝突不具合を発生させる方法である。すなわちテスト設計を行うためには、主に以下の３つの項目を決定しなくてはならない。

I. **共有リソースはどれか**

II. **共有リソースを変化させるにはどうすればよいか（衝突条件は何か）**

III. **どのリソースのどの機能を実行して、衝突不具合を検出するか**

この項目を順に決定するよう図２に示す設計フェーズを提案した。

I. **衝突対象選択フェーズ**
　あらかじめ与えられている衝突対象の候補から共有リソースを持つものを選択する
II. **衝突条件列挙フェーズ**
　選択された衝突対象が共有リソースの属性を変化させる条件を列挙する
III. **テスト機能選択フェーズ**
　属性の変化した共有リソースによる不具合を検出するために実行する機能を列挙する
IV. **優先順位決定フェーズ**
　製品の特徴や共有リソースの構造的な重要性などを考慮してテスト項目の優先順位の決定を行う
V. **テスト実行フェーズ**
　テスト項目を実装し実行して、不具合レポートと不具合の検出できなかったテスト項目のリストを作成する

図２　リソースパステスト法の設計フェーズ

リソースパステストでは、不具合の原因となるリソースの変化をテスト設計時に想定する。そのため、少ないテスト工数で早く不具合を発見することができ、また不具合の原因の特定も容易となる。また納期遅れが発生した場合に、テスト項目の省略による工数の削減と、省略によって検出不能となる不具合の危険度とのトレードオフによる評価を行うことができる。

リソースを単位としてテスト項目を管理するため、テスト項目の重複は発生しない。また発生させたい衝突をあらかじめ想定して設計するため、抜けも生じにくい。

また第４章では、リソースパステスト法で用いるツールおよび設計プロセスを提案した。

まず共有リソースを特定するため、リソースの依存関係を表したリソース依存グラフ（Resource Dependency Graph, RDG）をツールとして提案した。図３に概要を示す。

次に共有リソースの属性の変化と、属性の変化を発生させる条件とを対応させたリソース変化マトリックス（Resource Change Matrix, RCM）をツールとして提案した。図４に概要を示す。

最後に、共有リソースの属性の変化によって以前と異なる動作をするようになった機能および、共有リソース、機能を起動するリソース、結果を観測できるリソースを含むリソースパスを列挙したリソースパスリスト（Resource Path List）をツールとして提案した。図５に概要を示す。

さらに、図６に示すリソースパステスト法における設計プロセスも合わせて提案した。

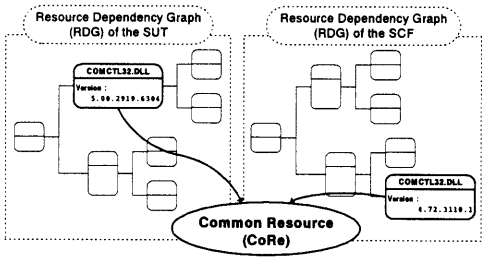
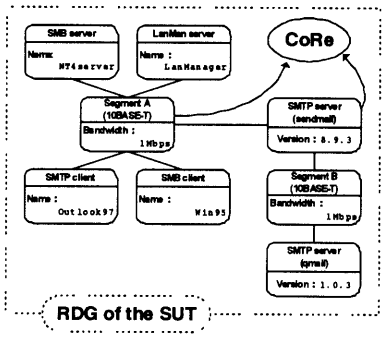


図３　リソース依存グラフ

CoRes / Failure Conditions	...	comctl32.dll	...	HKLM/Software/.../OpenGLDrivers	...	HKLM/System/.../dmd_nv4/Device0	...	PCI bus	...
Install the driver using install.exe		Add		Change RIVATNT key to nv4oglm(REG_SZ)					
Uninstall the driver from Control Pannel		Delete							
Change the refresh rate to 90Hz from Properies for Display						Change DefaultSettings.Vrefresh key to 90(REG_DWORD)			
Play a mpeg file								Occupy the bandwidth	

図４　リソース変化マトリックス



Resource Path List of the SUT

Resource Paths			Functions
Activation Resources	Common Resources	Observation Resources	
SMTP client	Segment A (10Base-T)	sendmail	send a mail to sendmail
		qmail	send a mail to qmail
SMB client		SMB server	browse other clients
			open a file
			⋮
		LanMan server	open a file
⋮		⋮	⋮
SMTP client	sendmail	qmail	send a mail to qmail
⋮	⋮	⋮	⋮

図５　リソース依存グラフとリソースパスリスト

I. 衝突対象選択フェーズ
- (1) テスト対象のリソース依存グラフ G0 および衝突対象候補のリソース依存グラフ G1～Gn を作成する。
- (2) G0 と G1～Gn を比較して、共有リソースを探す。
- (3) 共有リソースを持つ衝突対象候補を、衝突対象とする。

II. 衝突条件列挙フェーズ
- (4) テスト対象のリソース変化マトリックス M0 に共有リソースを書き込む。
- (5) 衝突対象それぞれについて、共有リソースを変化させる衝突条件を M0 に書き込む。
- (6) それぞれの衝突条件について、共有リソースの性質の変化をできるだけ具体的に M0 に書き込む。

III. テスト機能選択フェーズ
- (7) 共有リソースをリソース機能リスト L0 に書き込む。
- (8) 共有リソースを用いるテスト対象の機能 F1～Fk を L0 に書き込む
- (9) F1～Fk を起動する起動リソースを L0 に書き込む。
- (10) F1～Fk を実行した結果を観測できるリソースを L0 に書き込む。

IV. 優先順位決定フェーズ
- (11) 共有リソースごとに、衝突条件とテスト機能を整理し、テスト項目とする。
- (12) マーケティング的な機能の重要性や、共有リソースの構造的な重要性などから想定される、不具合の致命度、衝突条件の前後関係による工数の増減などを考慮して、テスト項目の優先順位および実行順序を定める。

V. テスト実行フェーズ
- (13) テストデータやテスト環境など、実行に必要な事項を決定する。
- (14) テスト項目ごとに、仕様から期待結果を導出する。
- (15) テスト環境を準備する。
- (16) テスト項目を実行する。
- (17) テスト項目の実行結果と期待結果とを照合する。
- (18) 不具合レポートと、不具合の検出できなかったテスト項目のリストを作成する

図６　リソースパステスト法の設計プロセス

第５章では、リソースパステスト法をストレステスト設計に適用するための理論的考察を行った。

リソースパステスト法で効果的にテスト設計を行うには、リソースと衝突、および衝突条件の知識が不可欠である。中でもストレステストに関する衝突条件は複雑である。そこでストレステストの設計を効果的に行うため、リソースステートモデル（Resource State Model）を提案し、衝突条件についてＦＴ図を用い理論的考察を行った。リソースステートモデルの概要を図７に示す。

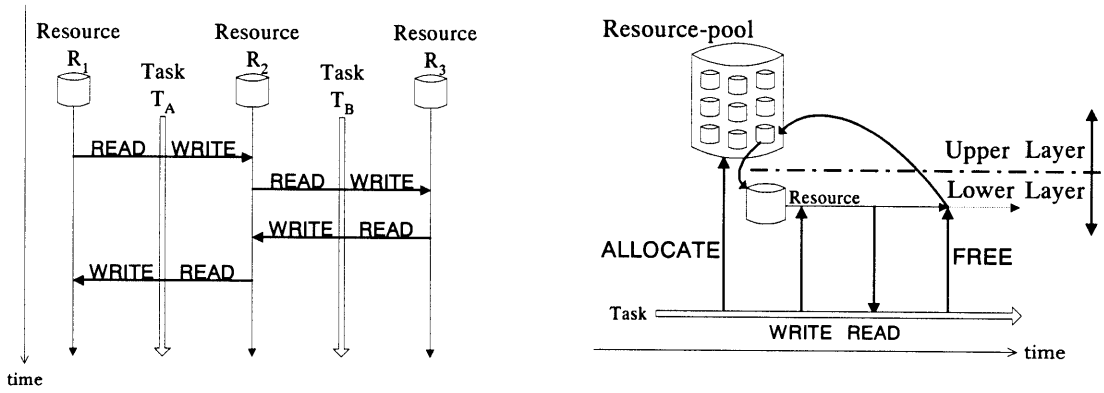


図７　リソースステートモデルの概要

第６章では、リソースパステスト法による品質評価について議論した。

テストの役割は、不具合を検出するだけではない。テストの真の役割は、ソフトウェアの品質特性を動的に測定することである。品質特性とは、ソフトウェアに要求される特性のことである。

まず、リソースパステスト法における障害対応性の評価について述べ、リソースの障害対応性の基準を図８に示すような５段階（対応せずを含めると６段階）として提案した。障害対応性の基準を図８に示す。また障害対応性の評価を簡便するための、障害対応性評価シートも合わせて提案した。

次に、リソースパステスト法によるセキュリティの評価について議論を行った。セキュリティホールのうち、設定ミスなどは、衝突が発生していないためリソースパステスト法では検出できない。しかしバッファオーバーフローなどは衝突が発生しているため、リソースパステスト法で検出することができる。

また新たな品質特性として「対象ソフトウェアのインストールやアンインストールといった構成変更や機能の実行によって同一システム内の他のソフトウェアの機能の実行を阻害しない属性」を両立性と定義し、両立性を調べるための両立性テストを提案した。両立性テストは、構成テストにおけるテスト対象と衝突対象を交換したテストであると考えてよい。そのためリソースパステスト法をそのまま適用してテスト設計を行うことができる。

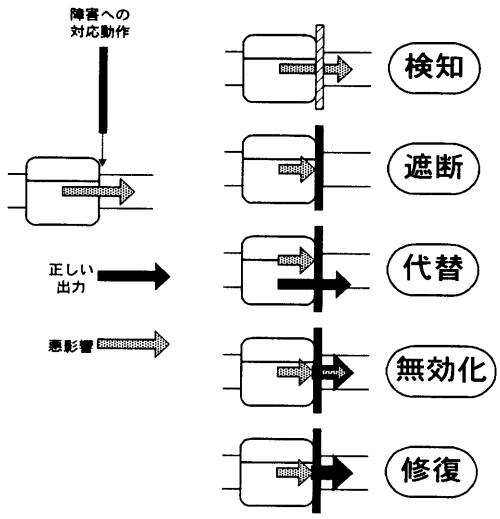


図８　リソースの障害対応性の基準

第7章では、実際に市場に出荷されたソフトウェアを対象として、リソースパステスト法を用いてストレステスト設計および構成テスト設計を行った適用例を示した。

ストレステストの設計では、参考として経験的手法による結果を示し、比較および考察を行ったテスト対象は2種のWWWブラウザとした。

提案した手法では、テスト対象の開発組織では検出できなかった不具合を簡便に検出することができた。特に、セキュリティホールになりうる確実に検出し修正しておかなくてはならない不具合を検出できた意義は非常に大きい。一方別種のテストでは、提案した手法を用いると、経験的手法で検出された改行位置ずれおよび改行余計という不具合が検出できず、また優先順位が後になってしまった不具合もあった。とはいえ提案した手法による検出結果をテスト実行後に検討した結果、リソース推測漏れの設計ミスが明らかになった。しかしその後設計レビューを行った結果、リソース推測漏れを容易に指摘することができた。一方経験的手法による設計レビューはテストケースごとに検討しなくてはならないため繁雑かつ困難であり、検出漏れの原因も判明しなかった。

構成テストの設計では、参考として経験的手法およびMyers、Beizer、Kanerの手法との比較及び考察を行った。テスト対象は商用ワードプロセッサ、衝突対象は3種のWWWブラウザとした。

不具合の検出に関しては、提案した手法が最も早いという結果となった。工数に関しては、提案した手法と経験的手法、Kanerの手法がそれぞれ3構成で最も少なかった。原因となる衝突対象の特定に関しても、提案した手法が最も早いという結果となった。一方テスト機能の順序に関しては、提案した手法でも経験的に定めざるを得なかった。しかし既存の手法では順不同となり、実際のテスト現場では適宜経験によって補完するものと推察されるため、あまり差がないという結論が得られた。またテスト工数の削減に関して、経験的手法では不具合の発生傾向とユーザの使用頻度から経験的に判断しており、Kanerの手法では品種が近ければ挙動も似ているという経験的な判断基準を用いていた。すなわち客観的に判断できるのは、省略により見逃す不具合の危険性を考慮できる提案した手法のみであった。したがって、この例では提案した手法が最も効果的であり、テスト経験が十分豊富であれば経験的手法も次いで効果的であるという結果になった。

さらに第7章では、リソースパステスト法を応用することによって、検出した不具合の原因の究明を試みた。その結果、検出した不具合はある補助的な機能が含まれているために発生する、という結論を、非常に少ない工数で得ることができた。

その結果、リソースパステスト法は、既存の手法に比べて、より早く、より少ない工数でより危険な不具合を検出できる方法であることが明らかになった。そして現実のテスト設計プロセスの抱える工数の減少という問題点を解決すべく、テストを止めた時点での危険性を評価することのできる手法であることも明らかになった。

最後に今後の展望として、リソースパステスト法を通じた"Test-Centered Development"などと呼ばれる開発プロセスの充実について触れた。これは、製品設計と同時にテスト設計を行うことによって、Testabilityを、ひいては製品の品質を向上させる開発プロセスのことである。これまでTCDという概念そのものは提示されていたが、具現化するテスト手法やツールは提示されてこなかった。リソースパステスト法の充実を通じてTCDへの適用を行い、より品質の高いソフトウェアを生み出す開発プロセスの構築に寄与すべきであろう。