ソフトウェアエンジニアリングシンポジウム 2020
IPSJ/SIGSE Software Engineering Symposium (SES2020)

大規模複雑化機能向け単体テストケース設計手法の提案

森拓郎 [1] 石川誠 [1] 金子昌永 [1]

概要：近年の医療機器では安全性はもとより，ユーザニーズの多様化やグローバルな製品展開に伴い機能の多様さが求められる傾向にある．そのためソフトウェアは短いサイクルで継続的に開発されており，品質確認のためには効率的なソフトウェアテストが重要となる．本報告では，大規模複雑化したソフトウェアのテストにおけるテストケース設計への取り組みについて示す．テストケース設計効率化のためのテストケース増大抑止手法を試行した結果，100 種の関数の自動単体テストを 3 時間以内に実行可能となる試算を得た．今後の課題は本手法の本格的な利用を通じた評価およびブラッシュアップである．

キーワード：単体テスト，テスト設計，オールペア法

Proposal of Unit Test Case Design Methodology for Large Scale Complex Functions

TAKURO MORI[†1] MAKOTO ISHIKAWA[†1]
MASANORI KANEKO[†1]

1. はじめに

医療機器のグローバル市場規模は 2016 年で 3362 億米ドルに及ぶ．国別では，米国が最大の市場であり，日本は米国に次ぐ世界 2 位である．一方で中国，アジア等の地域では今後急速な市場拡大が期待される成長分野である[1].

近年の医療機器はユーザニーズの多様化やグローバルな製品展開に伴い機能の多様さが求められる傾向にある．そのため医療機器メーカは製品競争力の維持・拡大のために放射線科，産科，循環器科などの幅広い診療科のニーズにいち早く対応する必要がある．

この実現のために医療機器によっては半年から一年に一度という高い頻度で機能のバージョンアップを実施している．バージョンアップにおいては，ユーザの利便性とコストの双方の点からハードウェア変更よりもソフトウェア変更が適している場合が多くソフトウェアの拡張開発は頻繁に実施される傾向にある．また医療機器では安全は最重要課題であり，そのためにはソフトウェアの拡張開発における品質の確保が必要である．

本研究では実施したソフトウェア品質の分析結果に基づき，単体テストの強化に注力した．単体テストの強化における課題は，繰り返しの拡張開発により大規模複雑化した関数に対するテストである．一度の開発における改修規模はごく一部であるにも関わらず，関数内でアクセスさせる変数は大規模であることため，関数に対するテストケース全体を見直した場合のテストコストが膨大となる．

本研究では開発範囲に対する網羅的なテストと，それ以外の機能に対する効率的なテストを両立するテストケース生成手法を開発した．

ある機能のテストに適用した試行では，200 を超える変数を有する関数の単体テストを 0.1 秒程度で実施できており手法の対象拡大においても現実的な時間内で単体テストが実施できる見通しを得た．

1 (株)日立製作所
Hitachi Ltd.

2. 課題と改善手法の検討

本研究で対象としたソフトウェアについて，現状分析を通じて問題点を明らかにして改善手法を検討する．

2.1 テスト対象の現状分析

本研究ではある医療機器のソフトウェア開発を対象として，開発プロセスの調査や担当者へのヒアリングを通じて状況を分析した．その結果，単体テストの自動化を通じた品質改善が有効との結果が得られると共に，単体テストに関する以下の 2 点の問題点が明らかになった．

問題点 1 テストケースの抜け漏れ

開発した機能をテストする場合には，機能の動作するバリエーションに対する網羅的なテストの実施が望ましい．しかし大規模複雑化した関数の全ての処理を確認するテストはコストの観点から現実的ではなく，テストケースは間引いたものが用いられる．間引きに伴うテストケースの設計は担当者の判断に任されるためテストケースの品質にばらつきが出やすくなり，結果としてテストの抜け漏れの要因となる．

問題点 2 変数の多さ

大規模な関数では読み込みおよび書き込みを行う変数が 200 個を超える場合がしばしばある．このため開発担当者は 1 つの関数でもその全体の仕様や挙動の把握が難しく，しばしば検討漏れの原因となる．

これらの問題を受け，単体テスト自動化による品質改善のためには，開発部分に対する網羅的なテストと，開発部分以外に対する影響波及テストが必要であると判断した．網羅的なテストを必要最小限とすると共に影響波及テストを効率的に実施し費用対効果の高い単体テスト自動化を実現する必要がある．

単体テストの自動実行においては，レポートを通じて毎日その経過を監視できるとマネジメントの点から有用である．そこで本研究では，単体テストは開発中の機能について夜間のバッチ処理で十分実行可能な時間に収まることを性能目標として 100 種の関数を 6 時間でテストすることを目標とする．

2.2 改善アプローチ

開発部分に対する網羅的なテストと，開発部分以外に対する影響波及テストを両立するために，関数がアクセスする変数に着目したテストケース生成を提案する．

ある機能開発対象関数があった場合，関数内の変数は機能開発の範囲に含まれているか否かで 2 種類に分類できる．ここで，機能開発の範囲に含まれている変数は機能のテストに影響を与える変数であるため「有則変数」と呼び，機能のテストに影響を与えない変数を「無則変数」と呼ぶ．

2.1 節で述べた問題点 1 は，有則変数を入出力としたテストにより解決が期待できる．問題点 2 は，有則・無則双方の変数の検討が必要であるが，無則変数は機能開発に原則影響を及ぼさないと考えられる変数であり，網羅的なテストほどの厳密性は不要と考える．

そこで本研究では有則変数と無則変数でそれぞれテストケースを生成して合算する手法を提案する．2.2.1 項では有則変数の網羅的なテストケース生成について，2.2.2 項では無則変数の効率的なテストケース生成について示す．

2.2.1 網羅的なテストケース生成

有則変数は開発機能に関連する変数群であり，ソフトウェア設計を通じてある程度仕様が定義されている変数である．テストケースの入力値はツールで生成可能であるが，それに対応するテストケースの期待結果(出力)については，テストの正しさに関する判断基準であるテストオラクルが必要でありツールでの作成は難しい．仕様やプログラムの情報に基づき，テストオラクルを定義して人的作業で作成する必要がある．

2.2.2 効率的な影響波及テストケース生成

無則変数は開発機能とは無関係と見なされている変数群である．言い換えると，無則変数のテストケースは仕様からは導出できない．また無則変数に値の候補を設定して総当たりするアプローチは，今回の例では規模が膨大となるため適用できない．

そこでオールペア法の適用を提案する．オールペア法は，ソフトウェア開発における欠陥の 90%以上は 1 個か 2 個の変数の組み合わせに起因するという調査[2]を前提としており，2 個の変数の組み合わせ全てを含むテストを実施して効率的に欠陥を抽出するというアプローチである．この前提のためオールペア法は変数間の関連性が疎なテストに向いており，密なテストには不向きである．無則変数はテスト対象に対して疎な変数であり，オールペア法による効率化が期待できる．

オールペア法では変数およびバリエーションの増加に対してテストケースが対数的に増加するため変数が膨大なテストにおいて有効である．オールペアツールである PICT[3]を用いて，変数とそのバリエーションごとに必要なテストケース数を算出した一例を表 1 に示す．32 個の変数がそれぞれ 8 種のバリエーションを持つ場合，総当たりでは 7.9×10^{23} 件となるテストケースが，オールペア法では 162 件と大幅に削減できる．

表 1　テストケース数比較

変数の個数	4		8		16		32	
値のバリエーション	2	8	2	8	2	8	2	8
総当たり テストケース数	16	4.1×10^{4}	256	1.7×10^{7}	6.5×10^{5}	2.8×10^{14}	4.2×10^{9}	7.9×10^{23}
オールペア テストケース数	5	97	8	101	10	133	12	162

3. テストケース生成方式

本章では，改善手法の検討により有力と仮説を立てたテストケース生成について，その実現方式および適用結果について示す．

3.1 テストケース生成方式の検討

検討したアプローチに基づきテストケース生成の要件を整理する．まずアプローチから事前条件および方式を以下のように定義した．

適用の前提条件

- テスト対象関数の名称およびバージョンが明確であり一意に特定できる
- テスト対象関数のうち開発部分が明確である

- 開発部分については開発内容の正しさを判断する仕様が存在する

方式概要

- テスト自動実行に利用可能なテストケースを生成する（入力情報に加えテスト自動実行プログラムが成否を判断するための出力情報を含む）
- 開発に影響する変数(有則変数)は網羅的なテストケースでテストする
- 開発に影響しない変数(無則変数)は影響波及をテストする

3.2 テストケース生成プロセス

本節ではテストケースを生成するための具体的なプロセスを定義する．全体像を図 1 に示し，本節の各項ではそれぞれのプロセスについて処理内容と共に試行結果を示す．なお本研究の段階では，(4)は既存のツールを，(5)は本研究で開発したツールを用いた自動化がなされている．

(1) 変数情報の抽出(3.2.1 項)
(2) 変数情報の分類(3.2.2 項)
(3) 有則テストケースの生成(3.2.3 項)
(4) 無則テストケースの生成(3.2.4 項)
(5) テストケースの統合(3.2.5 項)

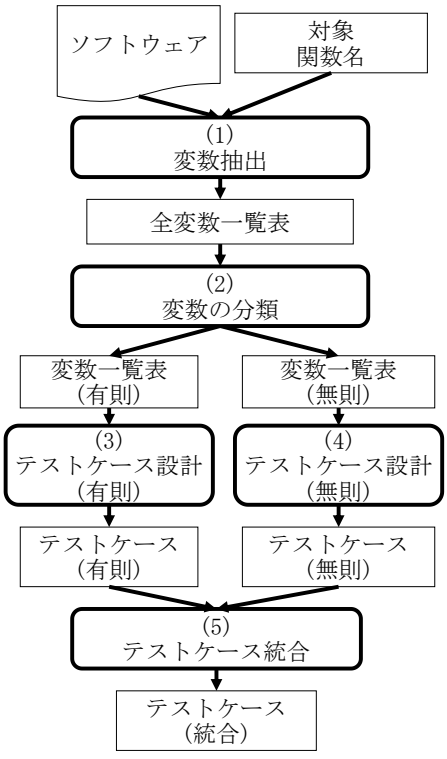

図 1 テストケース生成プロセス

3.2.1 変数情報の抽出

テスト対象関数を分析して，関数内の変数をアクセス(read/write)属性と併せて一覧化する(表 2 "変数", "read", "write")．変数が入出力のいずれも行う場合には，read および write 双方の属性を持つ．ここで read となる変数は単体テストにおけるテスト入力であり，write となる変数は単体テストにおける出力である．

3.2.2 変数情報の分類

テスト対象となる開発範囲を参考に，抽出した変数を有則変数と無則変数に分類する(表 2 "有則", "無則")．ここで有則変数か無則変数かの判断基準は，開発に伴い変更した変数ではなく，開発した部分的な機能の挙動に影響を与える変数か否かである．

表 2 有則・無則変数の例

変数	read	write	有則	無則
:	:	:	:	:
v21	○	－	－	○
v22	○	－	－	○
v23	○	－	○	－
v24	○	－	－	○
v25	○	－	－	○
v26	○	○	－	○
:	:	:	:	:

3.2.3 有則テストケースの生成

関数がアクセスする変数は整数型や浮動小数点型，二値型などさまざまであり，テストにおいてはこれらの様々な組み合わせについてテストする必要がある．そこで本研究ではプログラムにおける if-else 文や switch 文との相性の良さが知られている決定表[4]を採用する．決定表は値の組み合わせを表現する手法の 1 つである．

表 3 に決定表の例を示す．各入力変数に対して，出力変数が取るべき値を定義している．決定表では「変化しない(表 3 "変化せず")」も重要なテスト項目である．

次に決定表に基づいて具体的なテストケースを設計する．ここでは決定表に対して仕様の観点から変数のバリエーションを定義する．結果に影響を与える変数のみならず変化しないはずの変数についてもバリエーションを与え，誤った値で上書きされている可能性についてテストする．

与えるバリエーションには同値分割と境界値分析の考え方が有効である．同値分割とは，変数を同じ処理を行うグループに分割する作業で，ソフトウェアの仕様から導かれる．境界値分析では同値分割したグループに着目し，グループの境界付近の変数を重点的にテストするアプローチで

ある．

境界値分析を適用した例を表 4 に示す．期待結果として値が変更されるパターン(表 4 "パターン" 1〜3)については境界値である 0 近辺のテストケースを設計している．期待結果として値が変更されないパターン(表 4 "パターン" 4〜8)については 2 値の出力値と整数値の出力値をまとめたテストケースを作成している．

表 3　決定表の例

	変数	パターン							
		1	2	3	4	5	6	7	8
入力	v31	1	0	1	0	1	0	1	0
	v32	1	1	0	0	1	1	0	0
	v33	1	1	1	1	0	0	0	0
出力	v34	(v36)			(変化せず)				
	v35	ON			(変化せず)				

表 4　テストケースの例

#	パターン	入力						出力	
		v31	v32	v33	v36	v34	v35	v34	v35
1	1	1	1	1	10	0	OFF	0	ON
2						1		1	ON
3						-1		-1	ON
4	2	0	1	1	10	0	OFF	0	ON
5						1		1	ON
6						-1		-1	ON
7	3	1	0	1	10	0	OFF	0	ON
8						1		1	ON
9						-1		-1	ON
10	4	0	0	1	0	1	OFF	0	OFF
11					10	1	ON	10	ON
12	5	1	1	0	0	1	OFF	0	OFF
13					10	1	ON	10	ON
14	6	0	1	0	0	1	OFF	0	OFF
15					10	1	ON	10	ON
16	7	1	0	0	0	1	OFF	0	OFF
17					10	1	ON	10	ON
18	8	0	0	0	0	1	OFF	0	OFF
19					10	1	ON	10	ON

3.2.4 無則テストケースの生成

無則テストケースについてオールペア法を用いたテストケース生成を行う．本アプローチでは無則変数はテスト入力にのみ利用する．無則変数はテスト対象の実行結果とは無関係であるため，テストで確認すべき出力変数では不要である．

オールペア法適用にあたり，無則変数のバリエーションを分析する必要がある．例を表 5 に示す．ツール(PICT)によるオールペア法適用結果を表 6 に示す．

表 5　有則・無則変数の例

変数	read	write	有則	無則	バリエーション
:	:	:	:	:	:
v61	○	−	−	○	**0.0** **1.0** **2.0**
v62	○	−	−	○	**0, 1**
v63	○	−	○	−	**−**
v64	○	−	−	○	**0, 1**
v65	○	−	−	○	**0, 1**
v66	○	○	−	○	**0, 1**
:	:	:	:	:	:

表 6　有則・無則変数のバリエーションの例

variable	Test Cases							
	1	2	3	4	5	6	7	…
:	:	:	:	:	:	:	:	
v135	0.0	2.0	1.0	1.0	0.0	2.0	1.0	
v136	0	0	1	0	0	1	1	
v137	1	0	0	0	1	1	1	
v138	1	0	1	0	1	0	0	
v139	0	1	0	0	1	0	1	
v140	0	0	1	0	1	1	0	
:	:	:	:	:	:	:	:	

3.2.5 テストケースの統合

3.2.3 項で算出した有則テストケースと，3.2.4 項で算出した無則テストケースを統合し自動実行用のテストケースを作成する．

有則変数と無則変数の間に重複は無いため，それぞれのテストケースは合算が可能である．この統合においてはテスト対象を最大化するため，有則テストケースと無則テストケースの組み合わせを全てテストする．結果としてテストケースは有則テストケースと無則テストケースの乗算となる．

試行においては，有則テストケースが 19 件，無則テストケースが 23 件であったため，統合したテストケースは 437

件となった．テスト用の計算機上で実行した結果，テスト 1 件あたりに費やす実行時間は平均 0.1 秒以下であった．同様のテストを 100 種の他関数に実施した場合においても 1 台の計算機を約 2.4 時間利用する概算となる．これは目標としていた 6 時間以内でのテスト完了を 1 台の PC で実施可能であることを示す．各単体テストは独立しているため，テスト規模がさらに増大したとしても，計算機を増設し並列実行すれば対応可能である．

4. 結果の検討及び効果

3.2.5 項で得られた試行結果について，テストの規模から本研究の結果を検討する．有則テストケースと無則テストケースで規模の傾向は異なる．有則テストケースは網羅的なテストを実施するためその傾向は変数のバリエーションの積に比例する．無則テストケースはオールペア法で算出するためその傾向は無則変数数に対して対数的となる．統合テストケースは有則テストケースと無則テストケースの乗算となる．

そのため，本施策においては有則変数の規模がテストケースに強く影響する．有則変数の増大を防ぐためには，単体テスト 1 つで必要最低限の機能をテストするようにテスト対象を絞り込んだテスト設計が必要である．

方式比較として，手動テスト，総当たり，オールペア法との比較を実施した結果を表 7 に示す. 既存手法(表 7 (1))ではテストの設計・実装共に手作業であるため，その品質は作業者のスキルに依存し，また実行効率に課題がある．総当たり(表 7 (2))でテストを実施する場合，テスト品質は期待できる一方で，そのコストは自動テストでも改善できないほどの大規模となる．オールペア法(表 7 (3))ではコスト面では有効であるものの，複雑なロジックのテストにおける網羅性に課題がある．提案手法(表 7 (4))では大規模複雑化したソフトウェアの自動単体テストにおいて，他手法と比較して品質およびコストの双方で優位なテストが期待できる．ただし，そのためには有則変数が無則変数と比較して十分小さくなるよう，テスト対象を絞り込んだテスト設計が必要となる．

表 7　方式比較

		(1) 手作業	(2) 総当たり	(3) オールペア法	(4) 提案手法
品質	開発範囲	※	✔	−	✔
	開発範囲外	※	✔	✔	✔
コスト	テスト実施コスト	※	−	✔	✔
	テスト設計コスト	※	✔	✔	✔

✔:良好　−:不十分　※:担当者スキルに依存

5. おわりに

5.1 結論

本研究では医療装置向けソフトウェアの自動単体テスト改善を目標に，(1)テストの現状分析，(2)自動テストに向けた改善手法の検討および設計，(3)改善手法の試行を実施した．結果として，大規模複雑化したモジュールの単体テスト設計において，開発部分の網羅的なテストと，それ以外への影響波及テストを効率的に実施するためのテストケース生成手法を確立した．この手法を 100 種の関数に適用した場合，そのテストケース数は夜間のバッチ実行のような日ごとの自動テストで実行可能な規模(6 時間)に収まる見通しを得た．これにより欠陥の発見を迅速化し製品品質の向上に寄与できる．

(1) テストの現状分析

本研究の対象ソフトウェアでは関数がアクセスする変数が膨大であり，有則変数と無則変数に分類した分析が重要と分かった．これにより本研究では各変数に合致したテストケースの設計手法を実現した．

(2) 自動テストに向けた改善手法の検討および設計

開発部分に対する網羅的なテストケースの設計と，テストケース件数の増大抑止を両立するテストケース生成手法を設計した．具体的には同値分割法と境界値分析を用いた有則変数に対する網羅的なテストケース生成と，オールペア法を用いた無則テストケースに対する効率的なテストケース生成である．

(3) 改善手法の試行

試行では 203 の変数から開発対象と関連する有則変数 6 種を抽出し，437 件のテストケースを生成，実行した．同種のモジュール 100 種の関数に適用した場合の自動テストの実行時間は約 2.4 時間と概算でき，目標である 6 時間を達成できる見込みを得た．

5.2 今後の課題

本研究におけるテスト設計手法は大規模複雑化したソフトウェアの品質改善に有効と期待できる．一方でこの実現のためにはプログラムの分析を通じた変数情報抽出が必要である．本試行では手作業で実施しているが，さらなる適用範囲拡大のためにはツール化による作業効率化が必要である．

今後は自動単体テストの適用を通じた品質改善効果とコストのトレードオフについて評価する必要がある．対象関数が拡張された際の品質の変化や，品質をさらに向上するためのソフトウェアとテストのライフサイクルについて検

討する必要がある．

参考文献

[1] 我が国医療機器のイノベーションの加速化に関する研究会（第 2 回），経済産業省 商務・サービスグループ, Online at https://www.meti.go.jp/committee/kenkyukai/iryou_innovation/002_haifu.html, 2018.
[2] R.D.Kuhn et al. "Software Fault Interactions and Implications for Software Testing", IEEE Transactions on Software Engineering, 30(6), June 2004.
[3] Pairwise Independent Combinatorial Testing, Online at https://github.com/microsoft/pict, Microsoft.
[4] JIS X 0125:1986 決定表，日本工業規格, 1986.