機能モジュールに対する優先度に基づいた選択的ソフトウェアテスト手法の提案

平山雅之[*1*2]，山本徹也[*1]，岡安二郎[*1]
水野修[*2]，菊野亨[*2]

[*1] (株)東芝　研究開発センター　システム技術ラボラトリ
phone : 044-549-2403
e-mail: masayuki.hirayama@toshiba.co.jp
[*2] 大阪大学　大学院基礎工学研究科　情報数理系専攻

あらまし　近年のソフトウェアシステムの応用範囲の拡大に伴い，ソフトウェアの開発規模も飛躍的に増大する傾向にある．それと同時に，短い期間での製品開発が求められるようになっている．このような状況の下で，限られた時間内で効率的にソフトウェアテストを行う手法が必要とされている．本報告では，テスト対象となる機能に対して優先度の評価を行い，その評価結果に基づいてテスト仕様書の作成とテスト計画の作成をする「選択的テスト手法」を提案する．優先度の評価においては，構造の複雑さ，不具合の致命度などのプロダクト特性とレビュー充実度，開発者のスキルなどのプロセス特性を考慮している．テスト仕様書では優先度が高いほどチェックすべき具体的な項目を指示している．また，テスト計画では優先度が高い程，スキルを持った開発者が早めに実施するようにしている．

また，実際のソフトウェアテスト現場への適用実験を行った．その結果，提案する選択的テスト手法を用いたグループでは従来のテスト手法を用いたグループよりも多くの致命度の高い不具合を検出できることを確認した．

キーワード　ソフトウェアテスト，選択的テスト手法，テスト仕様書の設計，ソフトウェアメトリクス，優先度

A New Selective Software Testing Method Based on Priorities Assigned to Functional Modules

Masayuki Hirayama[*1*2], Tetsuya Yamamoto[*1], Jiro Okayasu[*1]
Osamu Mizuno[*2], Tohru Kikuno[*2]

*1　R&D Center System Engineering Lab., TOSHIBA Corporation
*2　Department of Informatics and Mathematical Science, Graduate School of Engineering Science, Osaka University

Abstract　As software systems have been introduced to many advanced applications, the size of software systems increases so much. Simultaneously, the lifetime of software systems becomes very small and thus their development is required to accomplish within relatively short period. In this paper, we propose a new selective software testing method that aims to attain the requirement of short period development. The proposed method consists of 3 steps: Assign priorities to functional modules (Step 1), Derive a test specification (Step 2), and Construct a test plan (Step 3) according to the priorities. In Step 1, for development of functional modules, we select both product and process properties to calculate priorities. Then, in Step 2, we generate detailed test items for each module according to its priority. Finally, in Step 3, we manage test resources including time and developer's skill to attain the requirement. As a result of experimental application, we can show superiority of the proposed testing method to the conventional testing method.

Keywords　Software testing, Selective software testing method, Testing priority, Design of testing specification, Software metrics, Priority

1. はじめに

ソフトウェア利用の拡大に伴い，開発されるソフトウェアの規模も飛躍的に大きくなる傾向にある. この結果, 実装後のソフトウェアのテストに関しても，確認すべき機能や操作，あるいは動作が多岐にわたってきている[1]. そしてそのことが，テスト工数の増大を招いている[8].

しかし，一方でソフトウェア開発に充当できる開発期間は限られており，現実には，テストの積み残しなどを引き起こす恐れさえ少なくない.

こうした傾向は製品組込みを前提とするマイコンソフトウェアで特に顕著である. 例えば, 近年の IT ブームを背景とした情報通信機器では，ソフトウェアが複雑化の一途をたどっており，そのテスト作業は困難を極めている.

ソフトウェアのテストは，一般に単体テスト，結合テスト, システムテストの3つに分けられる[5]. この中で, ソフトウェアシステムが提供する機能が当初のシステム仕様に合致しているかどうかを判定するのがシステムテストである. 近年のソフトウェアシステムが提供する機能は膨大になっており，システムテストでテストすべき機能も増加の一途をたどっている[8].

ここでシステムテストに関する現状の課題は以下のように整理できる.

(1) 機能が多岐にわたっており，テスト項目の全てを予め抽出することが難しい.
(2) 正常系の機能についてはシステム仕様書などを利用してテスト項目を抽出できる. しかし，異常系については仕様書に記述されないので，テスト項目としての抽出が難しい.
(3) システムが実現する機能について，どこまでテストすべきかの見極めが難しい.
(4) 限られた開発期間の中では，抽出された全てのテスト項目をテストするのが難しい.

こうした状況の下で「限られた開発期間内で，ソフトウェアのより高い信頼性を如何に保証していくか」という命題については多くの研究がなされてきている[2, 3, 6, 9]. 従来のソフトウェアシステムテストでは，機能やコードレベルで，網羅率を向上させる技術に主眼が置かれてきた. しかし, 巨大化したシステムの開発においては, ほとんど無限に近いテスト項目が必要となる[8]. 又，これらを実際にテストするには同様に無限の時間が必要となってしまう. このため，従来の網羅率偏重の考え方は成立しえない.

この場合に有望な方法として，「多くのテスト項目の中から，実際に不具合を検出する率の高いテスト項目を選りすぐって，効率良くテストする」あるいは「フィールドでの影響がより深刻な不具合を確実に除去する」といったことが考えられる. こうした考えに基づいてテスト項目の選別を行う手法が述べられている[7]. こうしたテスト手法をここでは選択的テスト手法と呼ぶ.

我々は，ソフトウェアが提供する機能に関して，様々な観点からテストの優先度付けを行い，それを利用して効率的にテストをする手法の研究を進めている. 本報告では，この優先度に基づいて，ソフトウェア機能を選択的にテストする手法の概要を提案する. また，実際の開発プロジェクトへの適用事例をもとに，提案法の効果について考察する.

2. 選択的テスト手法

2.1 概要

提案する選択的テスト手法は「機能の優先度決定」「テスト仕様書の作成」「テスト計画の作成」の3つのフェーズから構成される. 図1にその概要を示す.

選択的テスト手法では，まずソフトウェアシステムが提供する機能にテスト優先度を付与する. そして，重要な機能ほど詳細にテストし，逆に優先度の低い機能については軽めにテストをする. また，同時にこれらのテストの実行順序も制御する. つまり，許されたシステムテスト期間を有効に使い，より短い期間で最大の信頼性を得ることを目指す.

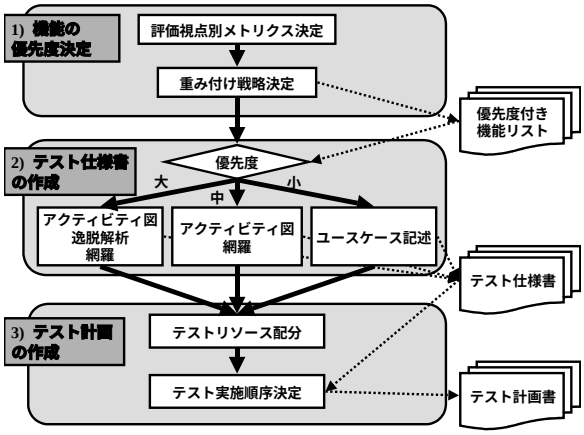

図 1　選択的テスト手法の概要

2.2 機能の優先度決定

ソフトウェアシステムが提供する各機能に対して，ユースケースの情報などを分析して，テストに関する優先度付けを行う. このとき，テスト優先度評価の視点およびメトリクスを利用する. この優先度の値を付加した機能のリストを作成する.

2.3 テスト仕様書の作成

優先度評価された個々の機能に対して具体的なテスト項目を設計していく. そのときテスト優先度に応じて, 必要なテスト項目の質や量を変化させる.

優先度の高い機能については，必要に応じて，ユースケース記述を利用した正常ケースからの逸脱(デビエーション)解析なども行う[4, 10]. こうして詳細なテスト項目を抽出する. 一方，優先度の低い機能については，ソフトウェア仕様書に記載された一通りの動作について確

認するのにとどめる．この結果はテスト仕様書として整理される．

2.4 テスト計画の作成

実際に利用できるテスト期間やリソースなどを考慮して，生成されたテスト項目の実行順序やテスト担当者の割当を行う．限られたテスト期間内により高い信頼性を保証するためには，重要な不具合ほど早期に検出することが求められる．

このためテスト作業計画の作成では，テスト優先度を考慮しつつ，一方でテストのしやすさなども加味した実行順序の決定を行う．また優先度の高い機能についてはスキルの高いテスト担当者を割り当てるといったことについても考慮する．これらはテスト計画書としてまとめられ，実際のテスト作業への入力となる．

3. テスト優先度決定手法

3.1 概要

ソフトウェア中に不具合が存在する確率は均一ではない．また，各不具合がソフトウェアシステムやユーザに与える影響も全てが同じではない．ここでは，「影響の大きい不具合がより多く存在する箇所を重点的にテストする」ことでテストの効率を向上させることを目指す．

本手法ではこうしたテストを実現するため，複数の視点から評価を行い，テスト優先度を決定する．

テスト優先度導出は次に示す 4 つのステップによって実現される．

Step-1: テスト対象の機能の抽出

ソフトウェアの仕様書をもとに，テスト対象となる機能をリストアップする．

Step-2: 評価視点の決定

抽出した機能に対し，どのような視点からテスト優先度付けを行うか，その視点を決定する．

Step-3: 評価メトリクスの重み付け

上記の視点毎に評価メトリクスを決定する．またそれぞれの評価メトリクスをどの程度重視するのか(重み付け)についても決定する．

Step-4: 機能の優先度付け

抽出した機能に対して，決定した重み付きメトリクスに基づいてテスト優先度を決定する．

3.2 優先度付けのための評価視点

ソフトウェアの不具合に関する基本動作としては，**不具合の混入，発現，影響**の 3 つが存在する．機能の優先度付けでもこれらを考慮する必要がある．

不具合の混入に対しては，開発したソフトウェア自身の機能規模や複雑さなどが大きく影響する．また同時にソフトウェア設計，実装段階での開発プロセスや開発者のスキルの影響も少なくない．

一方，**不具合の発現**についてはユーザがどのような場面でどう利用するか，あるいはその利用頻度がどの程度なのかなどが大きく影響する．**不具合の影響**については，発生した不具合がユーザにどのような影響を及ぼすかといった点を特に考慮する必要がある．

機能に対する優先度付けでは，不具合の混入，発現，影響の各基本動作を念頭において，評価範囲として「S_1: プロダクト特性」「S_2: プロセス特性」の 2 つに分けて考える．(図 2 参照)

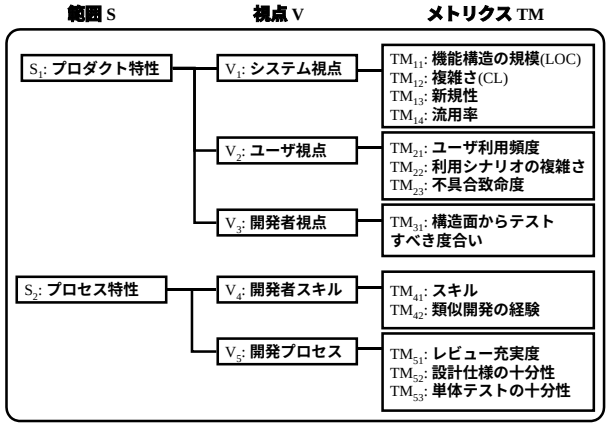

図 2　優先度評価の視点とメトリクス

(a) プロダクト特性

プロダクト特性はいわゆるソフトウェアプロダクトに関する評価のためのメトリクスを含む．これはソフトウェアとして実現される機能毎に決定される．

V_1: システム視点

特に不具合の混入に関する評価として注目する必要がある．ソフトウェアの規模や複雑さ，あるいはソフトウェアを構成する各部分の新規性や流用度合いなどを含む．

V_2: ユーザ視点

ユーザ視点は不具合の発現と影響に深い関係をもっている．提供される機能を利用する立場から評価するもので，ユーザの利用シナリオ(操作)の複雑さ，利用頻度，またその機能で発生した不具合の致命度などを考慮する．

例えば，携帯電話ソフトウェアを考えると，若年層とそれ以外の年齢層では全く異なった機能を重視する．

V_3: 開発者視点

通常，開発者は担当した機能の構造や機能間の関連などを熟知している．また過去の不具合事例なども考慮して, テストしておくべき機能についても良く知っている．このように開発者から見てどの機能に重点を置いたテストをすべきかを考慮するのが開発者視点である．

(b) プロセス特性

プロセス特性は不具合の混入に大きな影響を持つと考えられる．このためプロセス特性では，ソフトウェアで実現する個々の機能の開発について，

・どのようなスキルをもった開発担当者が開発したか

・どのようなプロセスで開発を進めたか

といった点を評価する．

V_4: 開発者スキルの視点

ソフトウェア不具合の原因の多くは開発者のスキルなどのせいにされる．同じ機能あるいは同じ複雑さを持つソフトウェアの開発であっても，スキルの低い開発者は不具合を多く作りこみ，そうでない開発者は不具合をほとんど作りこまない．

V_5: 開発プロセスの視点

同じスキルの開発者であっても，適切な開発プロセスを利用したか否かによって不具合を作りこむ度合いは異なってくる．経験的には開発プロセスが不適切な場合，設計や実装チェックなどが不十分になり，その結果として不具合を作りこむことが多い．

3.3 評価メトリクス

テスト優先度付けでは，各機能に上記のそれぞれの視点から見たテスト優先度を定量的に評価する．このため，上記の各視点V_i毎に，これを計測，評価するためのメトリクス(TM_{ij})を用意する．

例えば，システム視点V_1に関しては，TM_{11}:機能構造の規模(LOC)，TM_{12}:複雑さ(CL:サイクロマティック数)，TM_{13}:新規性，TM_{14}:流用率などをメトリクスとする．

それぞれのメトリクス(TM_{ij})は 1〜10 の値をとる．この値はテスト対象となる機能間での相対的な評価と考える．例えば，システム規模は図 3 に示すように，LOCの大きさに従って，1 から 10 の間の値をつける．

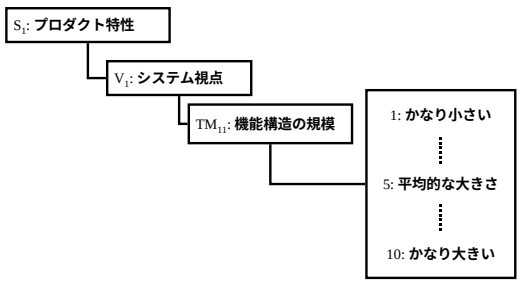

図 3　メトリクス評価の例

3.4 テスト戦略

機能の優先度付けに関しては上記のように様々な視点を考える必要がある．しかし，例えばエンドユーザが重視する機能と開発担当者がテストした方が良いと考える機能とは，必ずしも一致しない．またプロセスの視点から見てテストをするとよい機能とも一致するとは限らない．このため一つの視点からの評価ではなく，複数視点からの評価結果を総合的に判断することが必要となる．

実際のテストにおいては，どの視点を重視するか，どのメトリクスを重視するかを決定する必要がある．こうした戦略はメトリクスに重み付けを行うことで実現する．

例えば，ある機能Kに関して，評価メトリクスTM_{ij}の値がX_{ij}，その評価メトリクスの重みをW_{ij}とする．テスト優先度は次の式で計算される．(ここでW_{ij}は重み係数で，$0 \leqq W_{ij} \leqq 1.0$ の任意の値を設定する．)

$$\text{テスト優先度(機能K)} = \Sigma W_{ij} \times X_{ij}$$

例えば，あるテストでユーザ視点(V_2)を重視する戦略を採用する場合には，ユーザ視点に関するメトリクスであるユーザ利用頻度(TM_{21})，不具合致命度(TM_{23})などの重み係数を 1.0 に近い値にすればよい．同時に，開発者視点(V_3)などに対する係数を 0.2 程度にすればよい．

また，評価のためのメトリクス(図 2)は，その一部だけを利用することも可能である．例えば，プロセス特性を考慮しない場合には，プロセス特性に関するメトリクスの係数をすべて 0 とすればよい．

例えば，ワープロソフトの文書保存機能を考えてみる．評価メトリクスとして

TM_{12}：機能構造の複雑さ
TM_{21}：ユーザ利用頻度
TM_{23}：不具合致命度

の 3 つを考え，それぞれの値をTM_{12}=4, TM_{21}=6, TM_{23}=8 とする．また重み係数として，W_{12}=0.2, W_{21}=1.0, W_{23}=1.0 を設定する．この文書保存機能に関するテスト優先度は 0.2×4＋1.0×6+1.0×8=14.8 となる．

4. 適用実験

4.1 実験の目的

ここでは選択的テスト手法の有効性を評価するために行った実験について紹介する．実験ではソフトウェア開発の支援ツール(具体的には単体テスト支援ツール) の実際の開発に提案手法を適用した．

実験では図 4 に示すように，比較のために独立した 2 つのテストチーム A, B を用意し，それぞれに選択的テストと従来方式のテストを行わせ，検出される不具合について比較した．ここで従来方式とは与えられるテスト仕様書の順番にテストを実行していくものとする．

この実験の目的は，選択的テスト手法では機能に優先度を付け，この優先度に応じてテストの詳細度(入力のバリエーションなど))を変えることで，重要な不具合が多く検出されること(一方，従来方式のテストでは，選択的テスト手法に比べて重要な不具合の検出率が低くなっているということ)を確認することである．

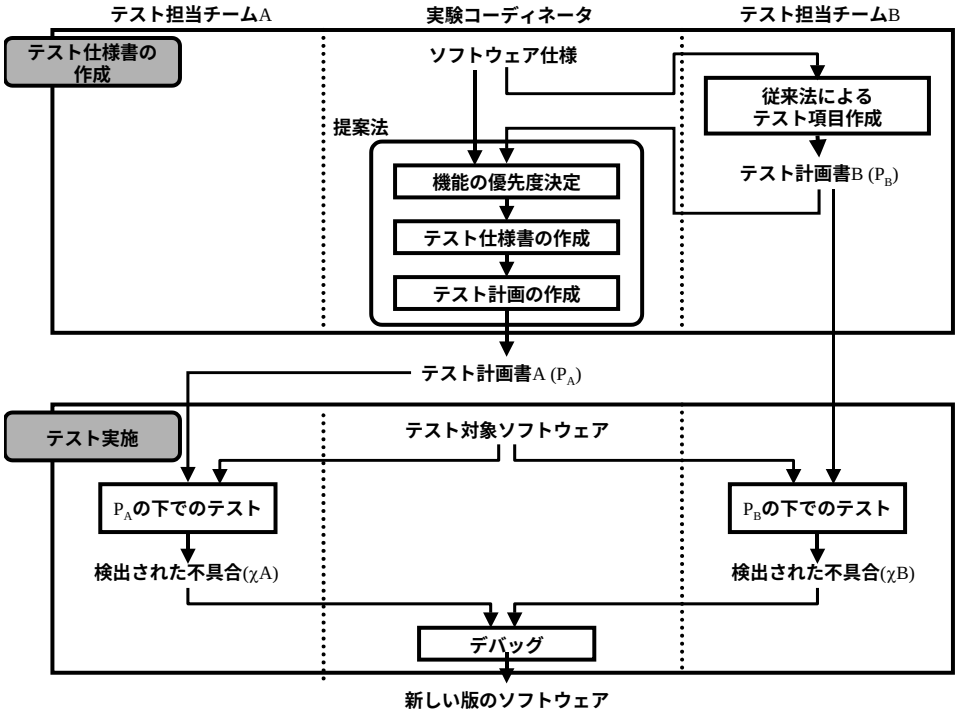

図4　実験の概要

4.2 実験対象のソフトウェア

今回のテストでは単体テストを支援するツールを対象とする．本ツールは，テスト対象を選定する機能，テストデータを自動生成する機能，スタブ/ドライバを自動生成する機能，などを含む．主な特性を次に示す．

使用言語：　C 言語
規模：　30KLOC
新規性：　全てが新規開発

4.3 実験方法

(1) テストチーム

独立した2チーム A と B が，それぞれ，選択的テスト手法，従来方式のテスト手法を利用してテストする．今回の実験ではそれぞれのチームの担当者のソフトウェア開発経験,スキルなどはほぼ同じレベルにし，能力差はほとんど生じないようにした．

またソフトウェアの設計，実装はテストとは別のグループが担当した．更に，これらのグループとは別に，ソフトウェア仕様の作成者がテスト実験コーディネータとして実験全体の調整を行った．

(2) テスト仕様書作成

まず，ソフトウェアの仕様書に基づいてグループ B の担当者が従来方式の考えでテスト仕様書 B を作成した．次に，テスト実験コーディネータがテスト仕様書 B について確認した後，ソフトウェアの機能の優先度付けを行った．続いてこの結果をもとに選択的テストのためのテスト仕様書 A を作成した．

テスト仕様書 A の中では各機能項目について，優先度を3段階(高，中，低)で明記した．このテスト仕様書 A でテストチーム A にテストをするように指示をした．なおテスト仕様書 B はテストチーム B が利用した．

(3) テストの実施

実際のテストでは，テストチーム A，B はそれぞれ独立してテストを実施した．テスト期間は1サイクルを1週間程度として，これを3回くり返した(図5参照)．

各サイクル内で両チームによって検出された不具合は，サイクルの終了時点でまとめて開発担当者にフィードバックされ，修正・デバッグ作業を行った．この作業を終えた時点で，再び次のサイクルのテストに着手する．そこでは前回と同じテスト仕様を用いて，新しい版のソフトウェアについてテストを行う．

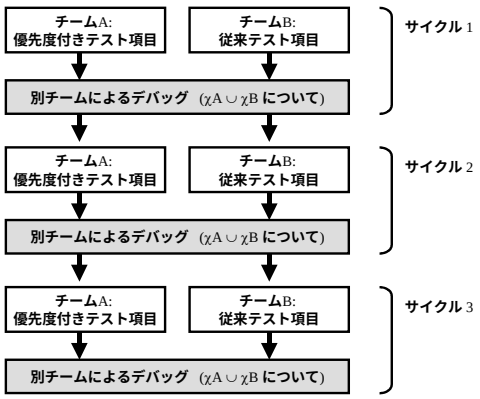

図5　テスト実施の詳細

テスト戦略		
致命度	利用頻度	複雑さ
0.33333	0.33333	0.33333

	機能名		システム特性値			
機能ID	大項目	中項目	致命度	利用頻度	複雑さ	Score
9	プロジェクトウィンドウ	最新の情報に更新	8	6	9	7.666667
2	メイン	プロジェクトを開く	8	5	10	7.666667
42	テストケースと結果ウィンドウ	一通りの表示	6	10	7	7.666667
1	メイン	プロジェクトの新規作成	10	2	9	7
0	起動	起動	10	5	3	6
34	具体値設定ダイアログ	具体値編集	6	7	5	6
23	生成パターンウィンドウ	一通りの表示	6	5	7	6
30	具体値設定ダイアログ	ランダム	5	7	6	6
31	具体値設定ダイアログ	境界値	7	4	7	6
63	具体値設定ダイアログ	表示	6	8	4	6
7	プロジェクトウィンドウ	ツリーの展開、折り畳み	5	10	2	5.666667
8	プロジェクトウィンドウ	プロジェクトにソースの追加	7	2	8	5.666667
45	テストケースと結果ウィンドウ	行選択(一行、複数、全行)	4	10	3	5.666667
58	テスト実行ダイアログ	実行	7	6	3	5.333333
28	生成パターンウィンドウ	生成パターンの上書き保存	9	3	3	5
54	テストケースと結果ウィンドウ	テストケースの上書き保存	8	4	3	5
40	構造体初期化ダイアログ	更新	7	2	6	5
6	プロジェクトウィンドウ	一通りの表示	6	5	4	5
38	構造体初期化ダイアログ	編集	6	4	5	5
64	構造体初期化ダイアログ	表示	6	4	5	5
21	コピー先関数選択ダイアログ	他の関数にコピー	6	2	7	5
48	テストケースと結果ウィンドウ	値の編集(基本型の時)	5	4	6	5
55	テストケースと結果ウィンドウ	テストケースの別ファイルへの保存	6	4	4	4.666667
16	プロジェクトウィンドウ	(テストパターン)削除	6	3	5	4.666667

図６　テスト対象の機能

4.4 機能の優先度付け

ここでは選択的テスト手法による機能の優先度付けの評価点と評価メトリクスを示す．今回の実験では，特にプロダクト特性のユーザ視点(V_2)に絞って評価を行った．

実験では各機能の利用頻度(TM_{21}) ,利用シナリオの複雑さ(TM_{22}) および不具合致命度(TM_{23})を評価メトリクスとして採用した．各機能毎の評価メトリクスの値は，実験コーディネータがその動作仕様から概要・特徴などを考慮して決定した．

またテスト戦略としては各評価メトリクスに均等な重みを与えることにした．したがって，各機能のテスト優先度は $0.33\times TM_{21}+0.33\times TM_{22}+ 0.33\times TM_{23}$ を利用して求めることができる．

4.5 テスト仕様書

各機能に対して評価点をつけ，優先度順にソートした機能の一覧を図 6 に示す．優先度が 5 以上のものを優先度=大，4 以上で 5 未満のものを優先度=中，4 未満のものを優先度=小とする．優先度別のテスト項目数については，優先度=大の項目が 118，優先度=中の項目が 51，優先度=小の項目が 56 となった．

またチーム A 向けに用意したテスト仕様書 A の一部を図 7(a)に示す．同図に示すように，テスト仕様書 A の優先度が高い機能のテスト項目には，どのような点を重視してテストをするべきか，あるいは，テストのバリエーションなどに関する具体的な指示を与えている．一方，優先度の低い機能に対するテスト項目には，軽めのテストで十分であることが明記されている．一方，図 7(b)に示すように, チーム B 向けのテスト仕様書 B にはどの項目にも特別な指示は与えられていない．

5. 実験結果

以降では，1 つのテスト項目で検出された不具合はまとめて 1 件と数えることにする．したがって，不具合の件数と不具合を検出したテスト項目数は一致する．

5.1 検出された不具合の数

表 1 (a)～(c)に各サイクルにおける不具合の検出の状況を示す.

(1) サイクル-1: 選択的テストと従来方式を合わせると優先度=大として検出された不具合の数は全部で 14 件であった．このうちの約 8 割を選択的テストで検出した．従来方式と選択的テストでともに検出されたものは 5 件である.

一方，選択的テストでは未検出であるにもかかわらず，従来方式で検出された優先度=大の不具合の数は 3 件であった. その原因について詳細に調査したところ, 本来，優先度=大であるため，テストに関する詳細な指示があ

表 1　検出された不具合の総数

(a)

サイクル1	優先度			計
	大	中	小	
χ_A	11	3	5	19
χ_B	8	2	7	17
$\chi_A \cap \chi_B$	5	2	4	11
$\chi_A \cup \chi_B$	14	3	8	25

(b)

サイクル2	優先度			計
	大	中	小	
χ_A	9	1	5	15
χ_B	4	2	5	11
$\chi_A \cap \chi_B$	0	0	3	3
$\chi_A \cup \chi_B$	13	3	7	23

(c)

サイクル3	優先度			計
	大	中	小	
χ_A	6	0	2	8
χ_B	0	1	1	2
$\chi_A \cap \chi_B$	0	0	1	1
$\chi_A \cup \chi_B$	6	1	2	9

χ_A: チームAのテスト項目で発見された不具合　　χ_B: チームBのテスト項目で発見された不具合

(a) 選択的テストのテスト仕様書A

項目番号	試験項目	期待される結果	合否
1	生成パターンウィンドウを起動する [既存開く，新規の両方に対して行う．具体値設定は全てのタイプ(引数，グローバル，スタブ)について行う]	(1) 変数名，型，具体値生成方針，具体値，ループ生成方針の各欄が表示される． (2) テスト対象関数で使用している引数，グローバル変数，スタブ関数の名前がこの順に変数名の欄に一覧表示される． (3) テスト対象関数から呼び出される関数で使用されるグローバル変数とスタブ変数の名前も変数名 (4) …	
2	行を選択してダブルクリックする． [基本パターンのみでOK]	具体値設定ダイアログが表示される．ただし，1の(7)(8)に該当する場合は，ダイアログは表示しない．	
3	行を選択して編集メニューの項目「詳細設定」を選択する． [基本パターンのみでOK]	2と同じ結果．	

優先度=高の項目に対する指示

優先度=低の項目に対する指示

(b) 従来法式のテスト仕様書B

項目番号	試験項目	期待される結果	合否
1	生成パターンウィンドウを起動する	(1) 変数名，型，具体値生成方針，具体値，ループ生成方針の各欄が表示される． (2) テスト対象関数で使用している引数，グローバル変数，スタブ関数の名前がこの順に変数名の欄に一覧表示される． (3) テスト対象関数から呼び出される関数で使用されるグローバル変数とスタブ変数の名前も変数名の欄に表示される． (4) …	
2	行を選択してダブルクリックする．	具体値設定ダイアログが表示される．ただし，1の(7)(8)に該当する場合は，ダイアログは表示しない．	
3	行を選択して編集メニューの項目「詳細設定」を選択する．	2と同じ結果．	

図7　テスト仕様書

たえられるはずであるが，この指示が不明確で結果的には従来方式と変わらなかった．また，優先度=小の不具合は全部で8件検出されたが，このうちの9割を従来方式で検出している．

(2) サイクル-2: 検出された不具合の総数は23件となっており，サイクル-1とほぼ同数であった．この23件の不具合の内訳について分析すると，次の3種類に分類できることが分かった．

a. サイクル-1の時点では未実装であった新しい機能(ソフトウェア)に関する不具合
b. サイクル-1での不具合の修正に起因する新たな不具合
c. サイクル-1での不具合の修正漏れ

このうち，b.またはc.と見なされる不具合は6件であった．

このサイクル2でも優先度=大の不具合13件のうち，選択的テストでは約7割を検出している．一方，従来方式では4割強にとどまっている．

(3) サイクル-3: 2回のテストにより不具合の多くが除去された状況で，最終の確認テストとして実施した．このサイクル3で検出された不具合の総数は9件である．そのうち約7割を優先度=大の不具合が占めており，その全てを選択的テストで検出している．検出された不具合の半分はサイクル-2での修正ミスであった．

5.2 不具合を検出したテスト項目

表2に3つのサイクルを通しての不具合の検出状況を示す．なお，表2では重複する不具合は除いている．

表2　不具合を検出したテスト項目

優先度	総数	選択的テストで検出したテスト項目数		従来方式で検出したテスト項目数	
大	26	22	85%	11	42%
中	6	4	67%	4	67%
小	13	7	54%	12	82%
合計	45	33	73%	27	60%

優先度=大の118個のテスト項目の内で，選択的テストでは22項目で不具合を検出した．一方，従来方式では11項目で不具合を検出した．優先度=大で不具合を検出したテスト項目の総数は26個なので，このうちの85%を選択的テストで検出したことになる．一方，従来方式は42%を検出したにとどまっている．

また，優先度=中の51個のテスト項目に関しては，従来方式，選択的テストともにそれぞれが4項目で不具合を検出している．なお，不具合を検出したテスト項目の総数は重複を除くと6個であった．一方，優先度=小の56個のテスト項目に関しては選択的テストと従来方式で検出した不具合の総数は13個であった．このうち選択的テストでは7個(54%)，従来方式では12個(92%)で検出している．

5.3 所要工数

今回の実験でテスト作業に要した工数を表3に示す．工数に関しては，従来方式と選択的テストの平均時間の

間には有意な差は見られなかった[1].

表 3 所要工数

	選択的テスト (時間)	従来方式 (時間)
サイクル 1	30	33
サイクル 2	25	23
サイクル 3	13.5	22

6. 評価

6.1 テスト項目に対する具体的な指示の影響

選択的テストでは優先度の高い機能に対するテスト項目にはテストに関する具体的な指示をした．こうすることによって，テストの質をコントロールする試みを行ってきた．先の結果を見ると，優先度=大のテスト項目では，選択的テストが従来方式に比べて約 2 倍近くの不具合を検出していることが確認できた．また優先度=中のテスト項目については選択的テストでも特に指示を与えていない．その結果、不具合検出率の上では優先度＝大のものほどの大きな差異は見られない．さらに優先度＝小のテスト項目については優先度＝大のものとは逆に従来方式の方が多くの不具合を検出している．

このように選択的テストによって，システムの重要な部分の不具合の効率的な検出が可能となる．一方，仕様書順にテストしていく従来方式では，システムの重要な機能や部分に対して必ずしも十分なテストが行われるという保証がなく，結果として不具合の取り残す可能性もある．

6.2 テスト期間の効率的な利用

表 1～表 3 より，選択的テストでは重要な機能に関するテスト項目に重きをおいた時間の利用がなされており，効率的に不具合を検出していることが分かる．特に，サイクル-3 では従来方式よりも短い時間で優先度の高い不具合を集中的に検出していることが推測できる．一方，従来方式のテストでは重要度に関係無く均等に工数を割くために，テスト効率面では必ずしも最適ではない．

今回の実験からは，プロダクト特性から優先度をつけテストの詳細度を変えるだけでも，テスト時の不具合の検出の状況を大きく変えることが可能であるとの結論が得られた．ここで更にプロセス特性も加味することで，より一層の効率の向上が期待できる．

7. まとめと今後の課題

本報告ではソフトウェアのシステムテストを効率化する選択的テスト手法について報告した．選択的テスト手法では機能の特性分析を行い，機能の優先度付けを行い，それに基づいてテスト計画書を作成する．

また，この提案手法を実際のソフトウェアのシステムテストに適用して実験的評価を行った．実験ではプロダクト特性からの評価を基にした優先度付けを行った．その結果，重要な機能のテストに十分な工数を投入することが可能になり，不具合が確実に検出できることを確認した．

今後の課題としては，プロセス特性からの評価を優先度付けに反映させること，テスト戦略に関しての各メトリクスに対する重み付けを決めること，などがある．

[1]有意水準 5%の統計的仮説検定による．

参考文献

[1] B. Beizer, Black-box Testing: techniques for functional testing of software and systems, John Wiley & Sons, 1995.

[2] D. M. Cohen, S. R. Dalal, J. Parelius and G. C. Patton, “The combinatorial design approach to automatic test generation,” IEEE Software, vol.26, no.9, pp.83-88, 1996.

[3] P. D. Coward, “A review of software testing,” Information and Software Technology, vol.30, no.3, pp.189-198, 1988.

[4] M. Hirayama, T. Yamamoto, T. Kishimoto, O. Mizuno, and T. Kikuno, “Generating test items for checking illegal behavior in software testing,” Proc. of 9th Asian Test Symposium (ATS2000), pp.235-240, 2000.

[5] M. R. Lyu, Handbook of Software Reliability Engineering, McGraw-Hill, 1995.

[6] Y. K. Malaiya, “Antirandom testing: Getting the most out of black-box testing,” Proc. 6th International Symposium on Software Reliability Engineering, pp.86-95, 1995.

[7] D. M. Marks, Testing Very Big Systems, McGraw-Hill, 1992.

[8] I. Sommerville, Software Engineering, 4th edition, Addison-Wesley, 1992.

[9] W. E. Wong, J. R. Horgan, S. London, and A. P. Mathur, “Effect of test set minimization on fault detection effectiveness,” Proc. of 17th International Conference on Software Engineering (ICSE’95), pp.41-50, 1995.

[10] 平山他, “UseCASE を利用したソフトウェアフォールトに対する SS-FTA の提案”, 電子情報通信学会 SS 研究会，SS99-53，2000.