テスト実行履歴に基づくテストケース生成手法の提案

野尻 周平[1,a)]　吉村 健太郎[1]　野中 誠[2]

概要：企業情報システム開発では費やせるコストが限られているため，テストの効率化が求められている．本論文では，欠陥の検出に要するテストケースの実行数を削減するテストケース生成手法として，テスト実行履歴に基づくテストケース生成手法を提案する．提案手法は，まずテストの実行履歴から優先度の高いテスト入力値の集合を抽出する．次に得られた入力値の集合を用いて，欠陥検出期待値の高いものが実行順序の上位に配置されるよう，テストケースのリストを自動生成する．本手法の前提となる欠陥発生の偏りを，筆者らが所属する組織のソフトウェア製品事例により確認した．また，提案手法をシミュレーションにより評価し，ソフトウェアに含まれる欠陥間の依存確率が 0.25 以上の場合，既存手法より効率的なテストが実施できることを明らかにした．

A Method of Test Case Generation with Applying a Testing History

Shuhei NOJIRI[1,a)]　Kentaro YOSHIMURA[1]　Makoto NONAKA[2]

Abstract: This paper describes a method to generate test cases based on a testing history for reducing the number of test cases to detect defects. Enterprise system development projects require a more efficient test method, because of cost limitation for software developments. We propose a testing method with extracting a set of high-priority input values from a testing history. The method automatically generates a list of test cases with using the result set of input values. A test case that is predicted high expectation about detecting defect is located in high rank of the list. A case study is conducted where there was a trend of defect incidents in module program units by analyzing results of integration testing and system testing. Also, the proposed method is evaluate quantitatively by simulation. When probability of dependency among defects included in a software is more than 0.25, the proposed method can test more effectively than conventional methods.

1. はじめに

企業情報システムを構成するソフトウェアは大規模であるため，機能を複数のモジュールに分解して設計・実装し，完成した個々のモジュールを組み上げて所望の機能を実現している．これら機能の品質を確認するために，テストが実施される．機能の信頼性を保証するためには，機能を構成するモジュールの各パラメータに対し，網羅的な値の組み合わせを入力してテストし，欠陥を取り除くことが望ましい．しかし，開発プロジェクトが投入できる工数には限りがあるため，完全な網羅は現実的でない．ソフトウェア製品の開発プロジェクトでは，限られた工数の中でいかにテストの効率を上げるか，すなわち，投入工数あたりの欠陥検出数をいかに向上させるかが重要な関心事である．

組み合わせによるテストを効率化する手法として，ペアワイズ法 [1] や直交表 [2] を用いた HAYST 法 [3] などが知られている．これらは，ソフトウェアに含まれる欠陥の多くが 2 つまでの入力値の組み合わせに由来して発生しているという知見 [4] に基づき，少数のパラメータ（多くの場合は 2 個）を効率的に網羅するように組み合わせて総テストケース数を抑え込む手法である．しかしながら，これらの手法はテストケースに含まれるパラメータやその入力値を等しく扱い，個々の値の重要度などを考慮しない．そのため，生成された全てのテストケースがコストの制約によって消化できない場合，ソフトウェアに含まれる欠陥が効率よく検出できるか否かはその時々に生成されたテストケースの順序に依存してしまう．

他方，欠陥は一部のモジュールに集中する傾向があり，

[1] (株) 日立製作所 横浜研究所
Hitachi, Ltd., Yokohama Research Laboratory
[2] 東洋大学 経営学部
Faculty of Business Administration, Toyo University
[a)] shuhei.nojiri.dd@hitachi.com

表 1 収集事例概要
Table 1 The overview of collected cases.

製品	製品 A	製品 B
構成モジュール数	77	40
欠陥検出モジュール数	7	5
欠陥数	62	19
テスト工程	結合，システム	結合，システム

また，早期テスト工程で欠陥が検出されたモジュールは，後期テスト工程においても欠陥が検出される傾向にあるという報告がある [5]．また，テスト工程において検出された欠陥の修正よってデグレードが発生し，欠陥修正に対し約 4 割の確率で新たな欠陥が埋め込まれるという報告もなされている [6]．この知見に基づき，欠陥の修正履歴を記憶し，欠陥モジュールを予測する研究がなされている [7]．また，欠陥が生じやすいモジュールの予測結果より，テストやレビューなどの品質保証にかける工数の最適化に関する研究もなされている [8][9]．しかしながら，各プロジェクトの欠陥検出傾向に基づいて具体的なテストケースを生成する取り組みについては知られていない．

本研究では，欠陥の検出に要するテストケースの実行数を削減するテストケースの生成手法の構築を目的とする．この目的に対して本論文では，テスト実行履歴に基づくテストケース生成方式を提案する．

本論文の構成を以下に述べる．2 章において，筆者らが所属する組織で開発されたソフトウェアの事例分析により欠陥の偏りが再現されるか否を検証する．3 章において，欠陥の偏りに基づくテスト効率化手法としてテスト実行履歴に基づくテスト生成方式を提案し，その特徴を述べる．4 章においてシミュレーションにより提案手法が有効に働く条件を明らかにする．5 章において提案手法について考察し，得られた知見について議論する．

2. 事例分析

2.1 欠陥発生の偏りに対する事例分析

Fenton ら [5] の報告する欠陥の偏りが，筆者らが所属している組織のソフトウェアにおいても生じているのか否かを検証するため，あるソフトウェア製品に関するデータを収集し，モジュール修正傾向を分析した．事例におけるモジュールの粒度はクラスファイルを単位とする．**表 1** に収集事例の概要を示す．

2.2 分析結果と評価

表 2, **表 3** に製品 A, 製品 B の結合テスト工程（Integrarion Testing）での欠陥検出数クラスに対する，システムテスト工程（System Testing）での欠陥検出数を示す．ST 平均欠陥検出数は，IT 欠陥検出数クラスに含まれる 1 モジュールあたりの欠陥数，ST 最大欠陥検出数は，IT 欠陥検出数クラスに含まれる 1 モジュールから検出された欠陥の最大数，同様に ST 最小欠陥検出数は，IT 欠陥検出数クラスに含まれるモジュールから検出された欠陥の最小数を示す．また，**図 1** には，製品 A の欠陥分布を箱ひげ図で示す，横軸は IT 欠陥検出数クラス，縦軸は IT 欠陥検出クラスに対する ST 欠陥検出数の分布である．

表 2 製品 A の IT 欠陥検出数クラスに対する ST 欠陥検出数
Table 2 The number of ST defects for the number of IT defects: Product A.

IT 欠陥検出件数クラス	0	1〜2	3〜
モジュール数	72	3	2
ST 平均欠陥検出数	0.11	2.0	15.5
ST 最大欠陥検出数	6	4	18
ST 最小欠陥検出数	0	0	13

表 3 製品 B の IT 欠陥検出数クラスに対する ST 欠陥検出数
Table 3 The number of ST defects for the number of IT defects: Product B.

IT 欠陥検出件数クラス	0	1〜2
モジュール数	36	4
ST 平均欠陥検出数	0.028	2.0
ST 最大欠陥検出数	1	4
ST 最小欠陥検出数	0	0

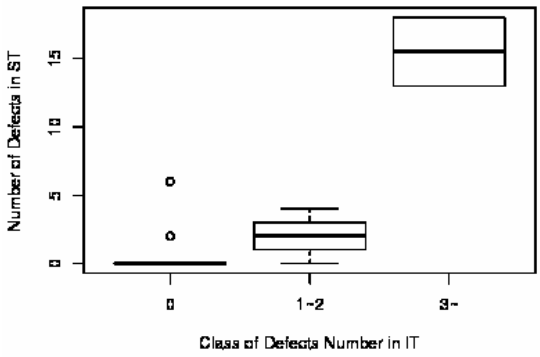

図 1 製品 A の欠陥検出数に対する箱ひげ図
Fig. 1 A boxplot of defects number: product A.

製品 A に着目すると，構成モジュール 77 個のうち 7 個（9.1%）から全ての欠陥が検出された．また，欠陥が多く検出された上位 2 モジュールから，全欠陥 62 件のうち 33 件（53.2%）が検出された．加えて，IT で比較的多く（本事例では 3 件以上）の欠陥が検出されたモジュールから，ST で検出された 41 件の欠陥のうち 27 件（68.9%）が検出された．

本事例分析により，筆者が所属する組織の製品においても，欠陥は特定のモジュールに偏る傾向があること，また，IT で欠陥が摘出されたモジュールは ST においても欠陥が摘出される傾向があることが確認された．

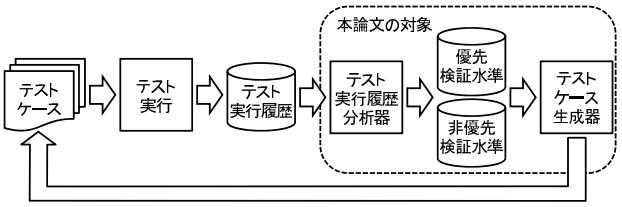

図 2 提案手法の概要
Fig. 2 An overview of the proposed method.

3. テスト実行履歴に基づくテスト生成方式の提案

3.1 仮説と方針

2 章の事例分析によって，欠陥は一部のモジュールに集中し発生し，また，早期テスト工程で欠陥が検出されたモジュールは，後期テスト工程においても欠陥が検出される傾向にあることが確認された．この検証結果に基づき，以下の仮説を立てた．

仮説 早期テスト工程で欠陥を検出した入力値は，後期テスト工程においても欠陥を検出する傾向を示す

本仮説に基づき，テスト入力値の欠陥検出傾向をテスト実行履歴より抽出し，欠陥検出傾向の高い入力値を優先的にテストケースに含めて生成する手法を構築する．

3.2 用語

以下，実験計画法 [2] を参考に，次のように語を用いる．

因子 対象の要因であり，ソフトウェアのモジュールに対するパラメータのこと

水準 各因子に設定する段階であり，モジュールのパラメータに対するテスト入力値のこと

3.3 提案手法の概要

図 2 に提案手法の概要を示す．また**図 3** には，提案手法によるテストケース生成の概念図を示す．提案手法は，テスト実行履歴分析器とテストケース生成器により構成する．テスト実行履歴分析器は，入力された実行履歴より水準の欠陥検出傾向を分析し，優先的にテストに組み込むべき水準と優先する必要がない水準を選別する．テストケース生成器は得られた水準選別結果に基づき，欠陥検出効率が高いと期待されるテストケースを生成する．以下の節で，各構成要素とその処理について述べる．

3.3.1 テスト実行履歴

テスト実行履歴は，テストケースとテストケースに含まれる因子，水準，および検出された欠陥と，その欠陥を検出した水準の組み合わせを含む．以降，本論文では，1 つ以上の異なる水準からなる水準の組合わせを**水準組**と呼ぶ．また，テストに用いられる因子，水準が明らかであり，また

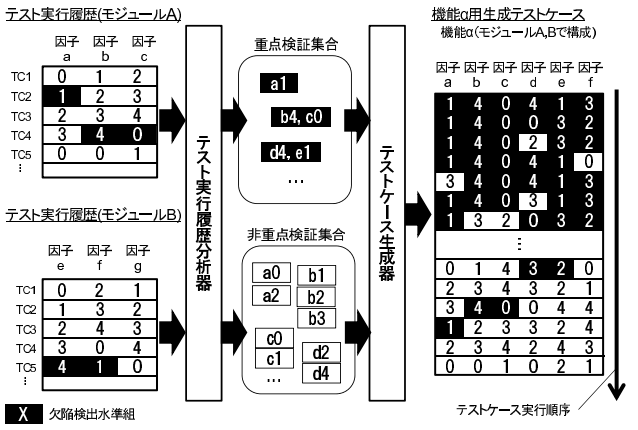

図 3 テストケース生成の概念図
Fig. 3 A conceptual diagram of test cases generation.

各因子の水準がある程度網羅されているという点から，本研究ではペアワイズ法などの因子網羅テストの実行履歴を前提とする．このテスト実行履歴には，テストケースに含まれるどの水準によって欠陥が検出されたのかについて，分析がなされた結果を含むものとする．

3.3.2 テスト実行履歴分析器

テスト実行履歴分析器は，下記の手順に従いテスト実行履歴を分析し，重点検証集合と非重点検証集合を生成する．

以下，次のテスト実行履歴例を用いて処理を説明する．A, B, C はそれぞれ因子であり，各々 3 つの水準を取る．

$$A = \{a_1, a_2, a_3\}, B = \{b_1, b_2, b_3\}, C = \{c_1, c_2, c_3\}$$

また，事前のテストにおいて $(a_1, b_1), (b_2, c_3)$ という 2 つの水準組により 2 つの欠陥が検出されたものとする．

手順 1 テスト実行履歴より，欠陥を検出した水準組を抽出
例. $(a_1, b_1), (b_2, c_3)$

手順 2 得られた水準組群を整理し，優先検査集合を構成
手順 2-1 水準組に含まれる各水準が所属する因子を参照
手順 2-2 所属する因子が共通な水準組ごとに欠陥検出水準集合（XY_{def}）を構成．この X, Y は，水準組に含まれる水準が所属する因子を表わす
例. $AB_{def} = \{(a_1, b_1)\}, BC_{def} = \{(b_2, c_3)\}$
手順 2-3 得られた欠陥検出水準集合を元とする優先検査集合（IV）を構成
例. $IV = \{AB_{def}, BC_{def}\}$

手順 3 テスト実行履歴より，欠陥検出に関わらなかった水準を因子ごとに整理し，非優先検査集合（nIV）を構成
例. $nIV = \{\bar{A}, \bar{B}, \bar{C}\}, \bar{A} = \{a_2, a_3\}, \bar{B} = \{b_3\}, \bar{C} = \{c_1, c_2\}$

3.3.3 テストケース生成器

テストケース生成器は，テスト実行履歴分析器によって生成された優先検証集合と非優先検証集合に基づき，下記

の手順に従ってテストケースを生成する．

手順 1　IV のべき集合 $Power(IV)$ を算出

手順 2　べき集合の各元について，要素数 (含まれる欠陥検出水準集合) が多いものから順に次の処理し，テストケースのリストを構成

（以下，べき集合の元 (AB_{def}, BC_{def}) についての処理を例に用いる）

手順 2-1　各欠陥集合を因子とした 2 因子網羅テストケース（TestCases）を生成

例．　$\mathrm{TestCases} = (((a_1, b_1), (b_2, c_3)))$

手順 2-2　得られた各ケースを元来のテスト因子ごとに整理する．重複する水準があれば除去

例．　$\mathrm{TestCases} = ((a_1, (b_1, b_2), c_3))$

手順 2-3　整理した結果，水準が 2 値以上とる場合は，ケースに含まれる他の因子の水準との組み合わせを網羅するように展開．展開した際のテストケースの順位は問わない．

例．　$\mathrm{TestCases} = ((a_1, b_1, c_3), (a_1, b_2, c_3))$

手順 2-4　ケースに不足する因子があれば，nIV より対応する因子の水準を補完

手順 2-5　生成されたテストケースをテストのリストに追加

手順 3　テストケースのリストに重複があれば，早期に生成されたテストケースを優先して重複を除去．

以上の処理によって，テスト履歴に基づき，過去に欠陥をよく検出した水準の優先度を高めたテストケースを得る．

4. シミュレーションによる検証

シミュレーションにより，ソフトウェアに含まれる欠陥の間にどの程度の確率で依存関係が存在すれば本手法が有効に働くかについて検証する．

4.1 欠陥の依存と依存確率

4.1.1 欠陥の依存

ある欠陥 D を検出するために入力が必要な水準の組み合わせを $D.Values$ とする．この際，2 つの欠陥 D_1 と D_2 が次の関係を満たしたときに，「欠陥 D_1 は欠陥 D_2 に依存する」と呼ぶ．

$$D_1.Values \supset D_2.Values$$

図 4 に欠陥検出の依存関係の例を示す．D_1 を検出するためには，画面 A の氏名欄に対して正しい値 (値 a)，画面 B の住所欄に対して空値 (値 b)，顧客テーブルにレコードが無いデータベースという 3 値テストデータに含むテストケース (TC_{D1}) を与える必要がある．しかし，ソフトウェア内に値 a，b が含まれるテストケースにより検出される欠陥 D_2 が同時に存在している場合，テストケースに TC_{D1} を与えた際，どちらの欠陥が発現するかは不定となる．このように，D_2 が取り除かれるまで，D_1 が存在しているのか否かを正しく検出することができないため，本研究では，欠陥 D_1 が欠陥 D_2 に依存すると表現する．

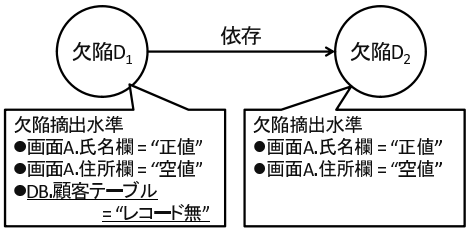

図 4　欠陥の依存関係
Fig. 4　A defect dependency between two defects.

表 4　テスト対象コンポーネントの構成
Table 4　A composition of a component for testing.

モジュール数	2 個
パラメタ数	各モジュールに対し 3 個
入力値数	各パラメタに対し 5 個
事前検出欠陥	2 因子欠陥
検出対象欠陥	3 因子欠陥

4.1.2 欠陥依存確率

本シミュレーションで検討する欠陥依存確率とは，ソフトウェアのある時点において，ソフトウェアに含まれる欠陥 D_1 を取り出した際，D_1 が依存する欠陥 D_n が 1 つ以上存在する確率である．

4.2 シミュレーション方法

シミュレーションは，テストケース生成手法によって生成したテストケースのうち，リストの冒頭から何件目のテストケース実行によって欠陥を検出できるかを計測する．本シミュレーションを提案手法と既存手法それぞれに対して実施し，結果を比較する．既存手法には，ペアワイズ法による 3 因子網羅テストケース生成手法を選んだ．シミュレーションの詳細を以下に述べる．

4.2.1 前提条件

表 4 に，シミュレーションの前提条件を示す．本前提条件は，各モジュールの単体テスト終了時点を想定している．一方のモジュールから 1 つの 2 因子由来の欠陥が検出され，修正が完了した状況である．

4.2.2 シミュレーション内容

シミュレーションは，次の流れに従い実施した．**表 5** にはシミュレーション条件を示す．

(1) 事前欠陥をランダムに生成
(2) 検出対象欠陥を欠陥依存確率に従い生成
(3) 事前欠陥を入力としてテストケース群を生成
(4) テストケース群を実行し，検出対象欠陥の検出に要したテストケース数を計測

表 5　シミュレーション条件
Table 5　Condition for simulation.

検出対象欠陥数	1 個
欠陥依存確率の刻み	0.05
各確率に対する試行回数	1000 回

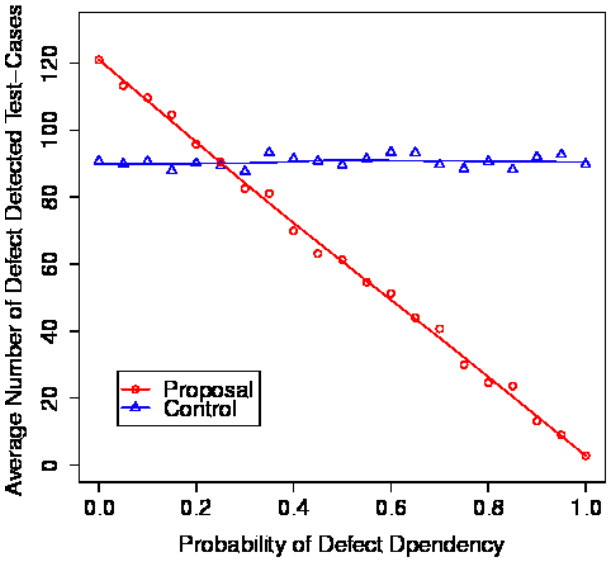

図 5　欠陥依存確率と平均テストケース数の関係
Fig. 5　A relation between probability of defect dependency and average of test cases number.

表 6　生成テストケース数の平均値
Table 6　Average number of generated test cases.

提案手法	既存手法
318.7	251.0

4.3　結果と評価

図 5 にシミュレーション結果を示す．横軸は 2 欠陥間の依存確率，縦軸は欠陥検出までに消費した平均テストケース数を表す．既存手法は欠陥検出に要した平均テストケースが依存確率に関わらずほぼ一定（約 90 ケース）であるのに対し，提案手法は依存確率の増加に対して単調減少している．提案手法は，欠陥の依存確率が 0 の時に欠陥発見のために平均で約 121.0 ケースを要し，依存確率が 0.25 の時には既存手法とほぼ同数の約 90.5 ケースを要した．依存確率が 0.3 の時より大きい場合には，提案手法の要した平均テストケース数が既存手法を下回り，依存確率 0.5 の時に約 61.3 ケース，依存確率 1 の時には約 2.8 ケースを要した．

また，**表 6** に生成された平均テストケース数を示す．生成される総テストケース数は提案手法の方が多く，既存手法に対し約 27%多い．

5.　考察

5.1　提案手法により生成されるテストケースついて

3.3 章において，テストケース欠陥傾向検出に基づく，優先順位付けされたテストケースの自動生成手法を提案した．提案手法は，前処理で得られた優先検査水準を優先的に組み合わせてテストケースを生成する．提案の手順によれば，自動的に優先度の高いテストケースがリストの上位に配置されたテストケース群を得ることができる．ただし，テストケース内に含まれる優先検査水準の数が同数である場合には，テストケースの優先度に順位付けできないため，各テストケースに完全な順序関係を定義するものではない．

5.2　提案手法の有効性について

続いて，提案手法の有効性について考察する．

図 5 に示したシミュレーション結果によれば，提案手法は，欠陥依存確率がおよそ 0.25 より高い値となる時に既存手法と比較し，より少ないテストケースで欠陥を検出できることが分かった．欠陥依存確率が 0.5 の時に，平均消費テストケース数を約 32%，欠陥依存確率が 1.0 の時には約 97%削減できる．欠陥の依存は，モジュールの欠陥発生傾向に偏りや，デグレードのようにある欠陥から他の欠陥が派生する場合に生じやすいと考えられる．このことより，モジュールの欠陥偏りやデグレード存在する開発プロジェクトにおいて，本手法が有効に働く可能性を示している．

しかし，表 6 に示した通り，生成される総テストケース数は既存手法より約 27%多い．ある程度の欠陥依存率が見込まれる場合，平均的には提案手法が勝るが，最悪のケースにおいて提案手法は劣る．これは提案手法がテスト水準群を複数の集合に分解し，それぞれに対して 2 因子網羅テストケースを生成した後にマージしているためである．

5.3　シミュレーションの妥当性について

最後に，提案手法の評価として実施したシミュレーションの妥当性について考察する．

4 章のシミュレーションは，欠陥間に依存関係がない場合の欠陥分布が完全にランダムであると仮定し，2 章で示した欠陥発生の偏りについては考慮していない．しかし，欠陥の偏りが存在すれば，同一の水準を含む欠陥が生じる確率が高まるため，提案手法により有利なシミュレーション結果が得られると考えられる．したがって，手法の有効性を検討する上で大きな障害とはならないと考える．

6.　関連研究

山本らは，欠陥の偏りを示すモジュールのメトリクス情報に基づき，テスト工数を注力すべきモジュールに優先的

にリソースを割り当てる手法を提案している [10]．これは欠陥の偏りに基づき，限られた工数の中でテストの効率化を試みた手法であるが，テスト時に実行する具体的なテストケースについては別途開発が必要となる．提案手法は，より細粒度のテストケースを自動生成している点が異なる．

秀島らは，ペアワイズ法の欠陥検出率向上を目的として，システム特性を入力情報にとり不要なテストケースが生成されないようパラメータの組み合わせを改良する手法を提案している [11]．しかし，本手法は入力値の重要度の決定に，その入力値に伴い実行されるソースコードの量を用いているが，一般的傾向に基づくものであるため選ばれた入力値が必ずしも，特定プロジェクトの欠陥の偏りに結びついているとは言えない．提案手法は，実際の各プロジェクトのテスト実行履歴から欠陥検出傾向を求めている点でアプローチが異なる．

また Kim らは，過去のテストケース実行有無や欠陥の検出履歴などのテストの実行履歴に基づいてテストケースの優先順位付けをし，回帰テストを効率化する手法を提案している [12]．しかしながら本手法は，回帰テストを対象に，すでに存在するテストケースについて順序付けをしている．提案手法は，ソフトウェアに潜在するより多くの欠陥を検出するため，欠陥検出傾向が高いと推測されるテストケースを新たに生成するという点でアプローチが異なる．

7. おわりに

7.1 まとめ

本論文では，欠陥検出に要するテストケース実行数を削減するテストケース生成手法として，テスト実行履歴に基づくテストケース生成手法を提案した．まず，本手法の前提である欠陥発生の偏りを，筆者らが所属する組織のソフトウェア製品事例により確認した．また，提案手法についてシミュレーションを実施した．この結果，ソフトウェアに含まれる欠陥間に 0.25 を上回る割合で欠陥依存が生じている場合，本手法はペアワイズ法に対して，平均的に少ないテストケース数の実行によって欠陥を検出できること示した．

7.2 今後の課題

今後の課題を以下に述べる．

7.2.1 テストケース生成アルゴリズムの改善

提案手法によって生成されるテスト総数は，既存手法より大きくなる．生成アルゴリズムの改善による総テスト数を抑え込みや，あらかじめ収集したテスト履歴から欠陥検出率を測定し，適切なテストケース生成手法を選択する手法の構築が今後の課題である．

7.2.2 事例による有効性検証

本論文では，シミュレーションにより本手法の有効性について評価したが，特に欠陥の依存の確率が実プロジェクトにおいてどの程度になるかなどについては，複数組織の実プロジェクトから得られた値による有効性の検証が必要である．

謝辞 本研究の一部は，国立情報学研究所が主催する「トップエスイープロジェクト」の修了制作として実施された．

参考文献

[1] 土屋達弘，菊野　亨：ペアワイズテスト：ソフトウェアテストの効率化を求めて，電子情報通信学会論文誌. D, 情報・システム， Vol. 90, No. 10, pp. 2663–2674 (2007-10-01).

[2] 田口玄一，横山巽子：実験計画法 (経営工学シリーズ)，日本規格協会 (1987).

[3] 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).

[4] Kuhn, D., Wallace, D. and Gallo, A.M., J.: Software fault interactions and implications for software testing, *Software Engineering, IEEE Transactions on*, Vol. 30, No. 6, pp. 418 – 421 (2004).

[5] Fenton, N. and Ohlsson, N.: Quantitative analysis of faults and failures in a complex software system, *Software Engineering, IEEE Transactions on*, Vol. 26, No. 8, pp. 797 –814 (2000).

[6] Mockus, A. and Votta, L.: Identifying reasons for software changes using historic databases, *Software Maintenance, 2000. Proceedings. International Conference on*, pp. 120 –130 (2000).

[7] Bell, R. M., Ostrand, T. J. and Weyuker, E. J.: Does measuring code change improve fault prediction?, *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, Promise '11, New York, NY, USA, ACM, pp. 2:1–2:8 (2011).

[8] 柿元健，門田暁人，亀井靖高，[マツ] 本真佑，松本健一，楠本真二：Fault-prone モジュール判別におけるテスト工数割当てとソフトウェア信頼性のモデル化，情報処理学会論文誌，Vol. 50, No. 7, pp. 1716–1724 (2009-07-15).

[9] 山下裕也，阿萬裕久：メトリクス値に基づいた重点レビュー対象モジュールの選択に関する考察: 整数計画法の利用，電子情報通信学会技術研究報告. KBSE, 知能ソフトウェア工学， Vol. 109, No. 432, pp. 1–5 (2010-02-25).

[10] 山本徹也，岡安二郎，新井本武士，平山雅之：機能モジュールの優先度評価に基づくテスト作業の効率化手法，情報処理学会研究報告. ソフトウェア工学研究会報告，Vol. 2002, No. 23, pp. 179–186 (2002-03-07).

[11] 秀島健太郎，土屋達弘，菊野　亨：システム特性を考慮したペアワイズテストの改良，電子情報通信学会技術研究報告. SS, ソフトウェアサイエンス， Vol. 108, No. 444, pp. 1–6 (2009-02-23).

[12] Kim, J.-M. and Porter, A.: A history-based test prioritization technique for regression testing in resource constrained environments, *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, New York, NY, USA, ACM, pp. 119–129 (2002).