修　士　論　文

題　　目　　組み合わせテストにおける実行順序に起因する
非決定的不具合誘発要因特定法の提案

主任指導教員　　水野　修　教授

指導教員　　崔　恩瀞　助教

京都工芸繊維大学 大学院工芸科学研究科

情報工学専攻

学生番号　　18622053

氏　　名　　渡辺　大輝

令和2年2月10日提出

様式 2 号

学位論文内容の要旨（和文）

令和 2 年 2 月 10 日

京都工芸繊維大学
大学院工芸科学研究科長　殿

大学院工芸科学研究科 情報工学専攻
平成 30 年入学
学生番号　18622053
氏　名　渡辺　大輝　㊞

（主任指導教員　水野　修　㊞　）

本学学位規則第 4 条に基づき、下記のとおり学位論文内容の要旨を提出いたします。

1.　論文題目

組み合わせテストにおける実行順序に起因する非決定的不具合誘発要因特定法の提案

2.　論文内容の要旨（400 字程度）

組み合わせテスト結果から不具合を誘発しているパラメータ組（MFS）を特定することを目的とした組み合わせ不具合局所化（FIL）は，開発者がソフトウェアシステムから不具合を取り除くための有益な情報をもたらす．これまでに提案された FIL 手法は MFS が決定的に不具合を誘発させることを前提にしたテスト結果を基としたものであるが，ソフトウェア開発においては非決定的なテスト結果を与える MFS の出現も珍しいものでは無い．

そこで本論文では，非決定的なテスト結果が得られる原因を調査し，その中でも実行順序に起因する不具合誘発要因の特定に焦点を当てた新しい FIL 手法である F-CODE を提案した．F-CODE はまず実行順序依存の原因となるパラメータ組（TS）を含むテストケースを特定し，以降の実行プロセスでは MFS 特定操作の前にそのテストケースを実行することで順序依存性を排除する．これにより，MFS と TS 特定作業において既存の FIL 操作の適用を可能にした．評価実験では，既存の FIL 手法と F-CODE を順序依存に依存した組み合わせテスト結果に適用し比較することで，F-CODE の有効性を示した．

A faulty interaction localization approach for non-deterministic failure-inducing combination depends on execution order

2020 18622053 *WATANABE Daiki*

Abstract

A faulty interaction localization (FIL) aims to identify a minimal failure-causing schema (MFS), which is a combination of parameter values that induces a defect from the result of a combinatorial testing. A FIL provides useful information for developers to remove causes of a failure from the software system. It is a common practice that MFS gives developers deterministic test results, but it is not a rare case that MFS gives developers non-deterministic test results that make developers impossible to use traditional FIL approaches.

In this thesis, we investigate causes of non-deterministic test results and then propose a new FIL approach named F-CODE that focuses on the detection of failure-inducing combinations that depend on the execution order of test cases. F-CODE first detects a test case that includes a trigger schema (TS), which is a combination of parameter values that lets MFS induce a defect. We then eliminating non-deterministic test results in MFS and TS detection process by executing the test cases first. This process makes developers possible to use traditional FIL approaches. We confirm that F-CODE can detect the failures while traditional FIL approach cannot.

目　次

1. 緒言

ソフトウェアの開発において，システムの動作に影響を与える不具合の出現は常に付き物である．リリースしたソフトウェアに不具合が存在すると製品の誤動作を招くだけでなく，ユーザーの信頼低下などにも繋がる恐れがある．この不具合が現代社会に及ぼす影響は大きいものであるがゆえに，それらを事前に検出するテストの重要性は非常に高い．また，ソフトウェアの不具合は単一の条件によって誘発されるだけでなく，時に複数の条件の相互作用によっても誘発される．このため全ての不具合を検出するためには，想定されるあらゆる条件でテストを行わなければならない．しかし，ソフトウェアの規模が大きくなるほどその候補は膨大になるため，それらを全てテストすることは現実的でない．

組み合わせテストは，ある一定のパラメータ値の組み合わせを網羅的に実行できるよう設計された一続きのテストケース，またそれを使用するテスト方法である．組み合わせテストを使用することは，不具合検出にかかるコストを削減する上で重要な役割を果たす．また，組み合わせテストによって得られたテスト結果から不具合誘発要因を特定する作業は一般に FIL（Faulty Interaction Localization）と呼ばれる．Zheng らによって提案された comFIL[1]，Ghandehari らによって提案された BEN[2] などは代表的な FIL 手法として知られている．FIL を行うことで，開発者はどの要因の相互作用によって不具合が誘発されるのかを明らかにすることができる．

しかし，テストの中には同じ状況下においてもその出力結果が一意に定まらない，いわゆる非決定的であるものも存在する [3]．既存の FIL 手法は同じテストケースの実行から常に同じ結果が得られることを前提においているため，こうした非決定的なテスト結果から不具合を特定することは難しい．

Luo らは 51 件の Apache オープンソースプロジェクトから非決定的な不具合を含む 201 のコミットを調査し，10 種類の非決定的要因に分類した [4]．更に，その内の約 12%が実行順序に依存するものであり，これら 10 種類の非決定的要因の中で大きな割合を占めていることを示した．そこで本論文では非決定的なテスト結果が得られる原因に着目し，実行順序に依存して不具合を誘発する要因の特定に焦点を絞った新しい FIL 手法である F-CODE（FIL Considering Order DEpendency）を提案する．F-CODE を用いることによって，既存の FIL 手法では困難だった非決定的不

具合誘発要因とその順序依存性の特定が実現される．F-CODE は，まず失敗したテストケースが事前に実行されたどのテストケースに依存して失敗したのかを特定する．次に，以降の実行プロセスではそのテストケースが最初に実行されるように設計し，非決定性を排除することで既存の FIL 手法を適用可能な状態にする．これによって失敗したテストケースに含まれる不具合誘発要因を特定する．最後に，実行順序依存の原因となるパラメータの組み合わせを特定する．

提案手法の有効性を示すため，本論文では二つの実験を行った．一つ目の実験では，順序依存性を持つ単一の不具合誘発要因による非決定的組み合わせテスト結果に提案手法と既存手法を適用し，精度と特定に要した追加テストケース数を比較した．二つ目の実験では，不具合誘発要因を複数含むテストスイートの実行結果に対して提案手法を適用し，一つ目の実験で得られた結果と比較した．実験の結果，提案手法は既存手法と比べて効果的であることが示された．また，複数の不具合誘発要因に対しては精度は低くなるものの，高い水準で特定が可能であることが示された．

本論文の構成は以下の通りである．第 2 章では，本研究の対象である組み合わせテスト，FIL，flaky test について説明する．第 3 章では，本研究の目的やそれを達成するための研究設問の設定を行う．第 4 章では，提案手法のアルゴリズムを具体例と共に示す．第 5 章では，従来手法と提案手法による比較実験を行い，その詳細を示す．第 6 章では，得られた実験結果を示す．第 7 章では，得られた実験結果を元に考察を行い，研究設問の回答，実験結果の妥当性，今後の課題について述べる．第 8 章では，本研究に関連する先行研究について説明を行う．第 9 章で結言とする．

2. 背景

2.1 組み合わせテスト

組み合わせテストとは，システムの入力パラメータの取る値において，ある組み合わせ数を与えたときにその組み合わせパターンの全てを網羅的にテストすることを目的としたブラックボックステストである [5, 6]．本論文ではパラメータが取る値のことをパラメータ値と呼称する．

t 個から成るパラメータ間の組み合わせに着目し，それが最小のテストケース数で全て網羅されるようにテストスイートを設計する t-way テストはその代表例である．本論文では t の値を網羅度と呼称する．ここでテストスイートとは，テストケースの集合を指す．例えば，あるシステムの入力として表 2.1 のパラメータ仕様が与えられたとする．全ての組み合わせを網羅する場合，必要なテストケース数は 3 種類のパラメータ値を取るパラメータが 1 つ，2 種類のパラメータ値を取るパラメータが 2 つであるから $3 \times 2 \times 2 = 12$ となる．対して表 2.2 に示す 2-way テストを考えると，テストケース数は 6 となる．考えられる全ての 2 つの組み合わせ (X, Y)，(Y, Z)，(Z, X) のパラメータ値を確認すると，確かに全てのパターンを網羅している．

組み合わせテスト生成ツールも数多く開発されており，使用されるアルゴリズム等によって分類ができる [7]．組み合わせテスト生成ツールには Czerwonka らが開発した PICT[8, 9]，Choi らが開発した pricot[10]，Bryce らが開発した DDA[11] などがある．

2.2 FIL（Faulty Interaction Localization）

FIL（Faulty Interaction Localization）とは，組み合わせテストの結果から不具合誘発要因である入力パラメータの組み合わせを特定する作業である．FIL によって開発者はどういった不具合誘発要因によってテストの失敗が引き起こされているかを特定することができ，不具合修正に役立てることができる．本論文ではこの特定すべき不具合誘発要因パラメータ組を MFS（Minimal Failure-causing Schema）と呼称する．スキーマとはパラメータ値の組み合わせの表現であり，$(X, Y, Z) = (1, 1, -)$ のように表される．これは X = 1 かつ Y = 1 であり，Z が組み合わせに含まれてい

表 2.1　パラメータの仕様例

パラメータ	パラメータ値
X	1, 2, 3
Y	1, 2
Z	1, 2

表 2.2　表 2.1 に対する 2-way テスト例

テストケース	X	Y	Z
t_1	1	1	1
t_2	1	2	2
t_3	2	1	2
t_4	2	2	1
t_5	3	1	1
t_6	3	2	2

ないことを意味する．また，スキーマに含まれるパラメータの数をスキーマの大きさと言う．例えば，$(X, Y, Z) = (1, 1, -)$ の大きさは 2 である．

代表的な FIL 手法として，Nie らは OFOT 法を開発している [12]．これは失敗したテストケースのパラメータ値を一つずつ変更しその結果から MFS を特定する手法を用いたものである．例として，表 2.1 のパラメータ仕様で MFS$(X, Y, Z) = (-, 2, 1)$ が存在している．これは $Y = 2$ かつ $Z = 1$ であるテストケースが失敗することを意味する．このとき，表 2.2 で与えられたテストスイートを実行するとテストケース t_4 が失敗し，それ以外のテストケースは成功する．

ここで失敗したテストケースである $t_4 = (2, 2, 1)$ に OFOT 法を適用すると，表 2.3 に示される追加テストケースが作成され実行される．X のパラメータ値を変更した t_7 の実行結果が失敗であることから，$X = 2$ は MFS に含まれないパラメータであることが分かる．反対に Y のパラメータ値を変更した t_8 と Z のパラメータ値を変更した t_9 の実行結果が成功であることから，$Y = 2$ と $Z = 1$ は MFS を構成するパラメータであることが分かる．これにより，MFS の特定結果が $(X, Y, Z) = (-, 2, 1)$ とされ，これは実際に存在している MFS と一致していることが確認できる．

他にも Zheng らによって提案された comFIL[1]，Ghandehari らによって提案された BEN[2]，Zhang らによって提案された FIC[13] などは代表的な FIL 手法として知られている．

2.3 flaky test

Micco は非決定的な出力結果をもたらすテストについて報告し，flaky test と命名している [3].

Luo らは 51 件の Apache オープンソースプロジェクトから 201 の flaky test のコミットを調査し，これらを 10 種類の非決定的要因に分類し，その結果として非同期待機，並行性，順序依存が上位 3 つを占めていることを明らかにした [4].

不具合検出が非決定性を持つ要因の一つとして，実行順序に起因する順序依存を例に挙げる．図 2.1 に示したソースコードはその一例である．これは testWrite メソッドが fs を介してファイルに書き込みを行い，他のテストで読み取られるデータを準備するテストクラスのスニペットである．JUnit はテスト順序の決定性を保証しない

表 2.3　OFOT 法適用例

追加テストケース	X	Y	Z	実行結果
t_7	3	2	1	失敗
t_8	2	1	1	成功
t_9	2	2	2	成功

```
1  int main(){
2  @BeforeClass
3  public static void beforeClass() throws Exception {
4    bench = new TestDFSIO
5    ...
6    cluster = new MiniDFSCluster.Builder(...).build()
7    FileSystem fs = cluster.getFileSystem();
8    bench.createControlFile(fs, ...);
9
10   testWrite();
11 }
```

図 2.1　実行順序に起因する flaky test の例

ため testWrite の前に読み取りテストが実行される可能性があり，その場合テストは失敗する．

3. 目的

3.1 動機

既存の FIL 手法は同じテストケースの実行から常に同じ結果が得られることを前提においている．そのため，非決定的なテスト結果にそれらを適用して正しい MFS を特定することは困難である．例えば，既存の FIL 手法では基本的に成功したテストケースに一度でも出現したパラメータ組は MFS ではないと判断されるが，非決定的なテスト結果では成功したテストケースに MFS が含まれる可能性がある．一部の研究では同じテストを何度も再実行することによって対策がなされていたが [14]，試行回数の分だけ不具合検出にかかるコストも増加するため非効率であり，また MFS を確実に発見できる保証もない．

2.3 節でも述べた通り，Luo らが調査した 51 件の Apache オープンソースプロジェクトの 201 のコミットではテスト結果が非決定的になる要因として順序依存が 12%程度存在することが報告されており，これを解決する FIL 手法の提案は有益であると考える．FIL を非決定的なテスト結果に対しても適用可能にするため，テスト結果が非決定的になってしまう原因のうち，本研究では順序依存に着目する．実行順序に依存する不具合は，その順序依存性を先に特定し，以降の実行プロセスでその順序依存性を保つように追加テストケースを作成することによって非決定性を排除する．これにより，既存の FIL 手法を適用できると考える．

以上の理由から，本論文では実行順序に依存して不具合を誘発する要因の特定に焦点を絞り，そうした不具合要因の特定を目的とした新しい FIL 手法である F-CODE を提案し，その有効性を明らかにする．F-CODE の目的は，既存の FIL 手法では困難だった非決定的不具合誘発要因とその順序依存性の特定である．

3.2 研究設問

前節で示した目的達成に向けて，以下の研究設問を設定する．

RQ1 提案した手法はどれほど正確に順序依存性を持つ不具合誘発要因を特定できるか．

RQ2 提案した手法はどれほど効率的に順序依存性を持つ不具合誘発要因を特定できるか.

RQ3 提案手法は順序依存性を持つ複数の不具合誘発要因を特定できるか.

4. 提案手法

4.1 前提条件

手法の提案に当たり，前提条件を以下のように設定する．

- 連続して複数回のテストを実行できる．全てのテストが終了すると，テスト対象の状態は初期化される．
- 全ての MFS にはそれぞれに対応したトリガースキーマ（以下 TS と呼称）が存在する．これは，そのスキーマを含むテストケースが先に実行されていることによって，MFS を含むテストケースが失敗するようなスキーマを指す．また，これらをまとめてバグセットと呼称する．
- MFS を含むテストケースは，TS を含むテストケースがそれ以前に実行されている時に限り必ず失敗する．そうでない場合には必ず成功する．MFS と TS が同一のテストケースに含まれている場合，その TS がそれ以前のテストケースに含まれていないなら，テストは成功する．
- 本手法の適用時において，使用者は対象のテストスイート内に非決定的なテスト結果が含まれていること，および決定的なテスト失敗が含まれないことを知っている．テスト結果が決定的であれば既存手法によって MFS を特定しその原因を修正できることや，その過程で非決定的なテスト結果を認識できることから，この仮定は妥当であると考える．

4.2 動作

失敗したテストケースのリスト L_{fail} から一つのテストケースを取り出し，バグセットの特定作業に移る．バグセットの特定作業は以下の 3 つのフェイズで構成される．

フェイズ 1：TS を含むテストケースの特定

着目したテストケースが失敗するための条件となる，それ以前に実行されたテストケースを特定する．失敗したテストケースを t_{fail} とする．実行したテストスイー

トにおいて t_{fail} より先に実行されたテストケースを順に選び，そのテストケースと t_{fail} を連続して実行する．t_{fail} が失敗した場合，そのテストケースは TS を含むテストケースとして特定される．これを t_{tri} とする．

フェイズ 2：MFS の特定

失敗したテストケースに含まれる MFS を特定する．2.2 節で示した OFOT 法を t_{fail} に適用することで，t_{fail} に含まれる MFS を特定する．このとき，OFOT 法によって t_{fail} のパラメータ値を一部変更したテストケースを実行する際に，t_{tri} の実行に続けて実行させる．これにより，実行順序によるテスト結果の非決定性を排除することができる．

フェイズ 3：TS の特定

フェイズ 1 で特定した t_{tri} に含まれる TS を特定する．フェイズ 2 と同様に，t_{tri} に OFOT 法を適用することで TS を特定する．具体的には，OFOT 法に従って t_{tri} の一部を変更したテストケースを作成し，そのテストケースと t_{fail} を続けて実行する．このときの t_{fail} の実行結果の変化によって TS が特定できる．

これらのフェイズを通して，t_{fail} の失敗を説明するバグセットの特定結果が得られる．L_{fail} が複数のテストケースからなる場合，既に得られた特定結果によって同様にその失敗を説明できるテストケースを L_{fail} から取り除く．L_{fail} が空にならなければ次の t_{fail} を取り出し，同様の手順でバグセットを特定する．これを L_{fail} が空になるまで繰り返す．

4.3 適用例

表 4.1 に示されたシステムモデルに対して提案手法を適用する．

例として，TS が $(a, b, c, d) = (-, 2, 1, -)$，MFS が $(a, b, c, d) = (-, -, 0, 0)$，計 1 個のバグセットが存在する場合を考える．まず表 4.2 に示す 2-way テストスイートを用いて実行する．テストケースの中で最初に実行される TS を含むテストケースは t_3 である．t_3 が実行されることにより，今後の実行では MFS を含むテストケースが失敗するようになる．この場合，t_4 以降のテストケースの中で MFS を含むのは t_5，t_8，

表 4.1　例題システムモデル

システムモデル	入力
例題（仮）	$3^2 \times 2^2$

表 4.2　表 4.1 に対する 2-way テスト例

テストケース	a	b	c	d	実行結果
t_1	0	0	0	0	成功
t_2	0	1	1	1	成功
t_3	0	2	1	0	成功
t_4	1	0	0	1	成功
t_5	1	1	0	0	失敗
t_6	1	2	1	1	成功
t_7	2	0	1	1	成功
t_8	2	1	0	0	失敗
t_9	2	2	0	0	失敗

t_9 であり，これらの実行結果は失敗となる．ゆえに，$L_{fail} = \{t_5, t_8, t_9\}$ が得られる．t_1 にも MFS が含まれているが，これは trigger を含むテストケースである t_3 が実行される前であるため，実行結果は成功となる．

実行結果が得られた後，L_{fail} から t_5 を取り出し，$t_{fail} = t_5$ とする．次に，前節で示したフェイズ 1 として，t_{tri} の特定を行う．今回の場合，TS を含むテストケースの候補は t_1, t_2, t_3, t_4 である．まず t_1 を実行した後に t_5 を実行すると，t_1 には TS が含まれていないことから直後の t_5 の実行結果は成功となり，ゆえに $t_{tri} \neq t_1$ であることが分かる．同様に，$t_{tri} \neq t_2$ であることが分かる．次に，t_3 を実行した後に t_5 を実行すると，t_3 には TS が含まれていることから t_5 の実行結果は失敗となり，ゆえに $t_{tri} = t_3$ であることが分かる．このフェイズ 1 全体において使用された追加テストケースを表 4.3 に示す．表 4.3 の横線はシステムの初期化を表している．

次に前節で示したフェイズ 2 として，MFS を特定する．MFS 特定作業の前に，フェイズ 1 で特定した t_{tri}（$= t_3$）を実行する．これにより以降の実行では MFS を含むテストケースが失敗するようになり，MFS の特定を可能にする．MFS を含む t_{fail}（$= t_5$）に OFOT 法を適用させたときに生成される追加テストケースを表 4.4 に示す．$t_{10}, t_{11}, t_{12}, t_{13}$ はそれぞれ t_{fail} の各パラメータを一つ変化させたものである．t_{10} と t_{11} の実行結果は失敗であることから，これらには MFS が含まれていることが分かる．パラメータ c，d を変化させることによって MFS が消失したことから，MFS は $(a, b, c, d) = (-, -, 0, 0)$ と判明する．

最後に前節で示したフェイズ 3 として，TS を特定する．MFS 特定作業と同様に t_{tri} に OFOT 法を適用させ，生成した各テストケースを実行後に t_{fail} を実行する．生成された追加テストケースを表 4.5 に示す．表 4.3 と同様に，横線はシステムの初期化を表している．t_{14}，t_{17} を実行した後の t_{fail} の実行結果が失敗であることから，これらには TS が含まれていることが分かる．パラメータ b，c を変化させることによって TS が消失したことから，TS は $(a, b, c, d) = (-, 2, 1, -)$ と判明する．

t_5 における特定作業が終了し MFS$(a, b, c, d) = (-, -, 0, 0)$ が得られたことから，これを含む t_8 と t_9 を L_{fail} から取り除く．これにより L_{fail} が空となったため，TS$(a, b, c, d) = (-, 2, 1, -)$ および MFS$(a, b, c, d) = (-, -, 0, 0)$ がバグセットの特定結果として出力される．

表 4.3　フェイズ 1 において生成される追加テストケース

テストケース	a	b	c	d	実行結果
t_1	0	0	0	0	——
t_5	1	1	0	0	成功
t_2	0	1	1	1	——
t_5	1	1	0	0	成功
t_3	0	2	1	0	——
t_5	1	1	0	0	失敗

表 4.4　フェイズ 2 において生成される追加テストケース

テストケース	a	b	c	d	実行結果
t_3	0	2	1	0	——
t_{10}	2	1	0	0	失敗
t_{11}	1	2	0	0	失敗
t_{12}	1	1	1	0	成功
t_{13}	1	1	0	1	成功

表 4.5　フェイズ 3 において生成される追加テストケース

テストケース	a	b	c	d	実行結果
t_{14}	1	2	1	0	——
t_5	1	1	0	0	失敗
t_{15}	0	0	1	0	——
t_5	1	1	0	0	成功
t_{16}	0	2	0	0	——
t_5	1	1	0	0	成功
t_{17}	0	2	1	1	——
t_5	1	1	0	0	失敗

5. 評価実験

5.1 実験内容

提案手法の有効性を評価するため，2 つの評価実験を行った．

実験 1

RQ1，RQ2 への回答のために，実行順序依存性を持つ単一のバグセットによる組み合わせテストスイートの実行結果に提案手法を適用し，精度を評価する．また，特定に要した追加テストケースの数を数える．実験対象には実在するシステムの入力モデルを用いてランダムベースで生成したテスト結果を使用する．これらを既存手法についても行い，提案手法の結果と比較する．

実験 2

RQ3 への回答のために，複数のバグセットが存在する場合のテスト結果への適用について，実験 1 と同様に評価する．

5.2 実験対象

5.2.1 概要

これまでの先行研究では，実際に不具合報告のあったシステムを対象に FIL 手法を適用することで評価を行っている．しかし，本手法が評価したい順序依存性のある不具合と，それを検出するテストケースおよびシステムモデルを兼ね備えた報告はこれまでに確認できておらず，再現ができない．また，先行研究のように数件の事例だけの評価では結果を一般化できない．こうした理由を踏まえ，実際のシステムにおける架空の不具合を多数想定し，そのテスト結果を対象として評価を行う．この不具合の想定は 5.3 節の実験準備で詳述する．

5.2.2 対象プロジェクト

本実験で用いた対象プロジェクトのシステムモデルを表 5.1 に示す．これらのプロジェクトは実際に先行研究で使われていたものであることから本実験でも用いるこ

ととした [14].

5.2.3 対象テストスイート

対象プロジェクトのシステムモデルの t-way テストを構成するテストケース数を表 5.2 に示す．t-way テストの生成には組み合わせテスト生成ツールである PICT を使用した．PICT は組み合わせテスト生成ツールの中では認知度が高く広く用いられているため，本実験においても用いることとした．

5.3 実験準備

まず，各システムモデルに対応する t-way テストスイートを作成する．本実験では 4-way テストスイートまでを用いた．

次に，各システム毎にその存在を想定するバグセットをランダムに設定する．それぞれのバグセットごとに MFS および TS を構成するパラメータ値をランダムに決定する．ただし，これらのスキーマの大きさは 4 までとする．またこのとき，MFS と TS は一致しない．

続けて，設定したバグセットを想定した場合の組み合わせテストの結果を生成する．この組み合わせテストは，設定したバグセットによる不具合の検出を最小限の網羅度で行える t-way テストであるとする．つまり，最初に 2-way テストのそれぞれのテストケースの実行結果を調べ，全て成功したなら，網羅度の一つ高い 3-way テストの結果を確認する．一つでも失敗したなら，そのテスト結果を本実験における FIL 実行対象として用いる．4-way テストでも全て成功したなら，設定したバグセットは使用しない．

こうして，あるバグセットが想定されている t-way テスト結果を得ることができる．実験 1 のために，1 つのバグセットが想定されたテスト結果をそれぞれのシステムに対して重複なく 1,000 個収集する．実験 2 のために，2 つおよび 3 つのバグセットが想定されたテスト結果をそれぞれのシステムに対して重複なく 1,000 個ずつ収集する．ただし実験 2 において，複数のバグセットの内容は一致しない．

表 5.1　実験に用いたシステムモデル

システムモデル	入力	パラメータ数
Tomcat	$2^8 \times 3^1 \times 4^1$	10
Hsqldb	$2^9 \times 3^2 \times 4^1$	12
Gcc	$2^9 \times 6^1$	10
Jflex	$2^{10} \times 3^2 \times 4^1$	13
Tcas	$2^7 \times 3^2 \times 4^1 \times 10^2$	12

表 5.2　t-way テストを構成するテストケース数

システムモデル	t = 2	t = 3	t = 4
Tomcat	14	36	89
Hsqldb	14	43	122
Gcc	14	48	105
Jflex	12	46	120
Tcas	100	404	1,382

5.4 比較手法

比較手法には 2.2 節で示した OFOT 法を用いる．生成した追加テストケースを一つずつ実行した場合，順序依存性を持つ MFS の特定には少なくとも 2 つ以上のテストケースの実行が必要であるため，OFOT 法による特定は不可能である．全ての追加テストを先に設計し全ての追加テストケースを連続して実行することで，OFOT 法によって特定できる可能性がある．例として，表 4.1 に示した例題システムモデルに TS が $(a, b, c, d) = (0, -, -, -)$，MFS が $(a, b, c, d) = (2, -, 1, -)$ であるバグセットを想定した 2-way テスト結果を表 5.3 に示す．失敗したテストケースである t_7 に OFOT 法を適用すると，表 5.4 に示す 4 つの追加テストケースが生成される．これらを連続して実行した場合，t_{10} に TS が含まれるため MFS を含む t_{11}，t_{13} が失敗する．成功したテストケースが t_{10} と t_{12} であることから，MFS の特定結果 $(a, b, c, d) = (2, -, 1, -)$ が得られ，これは想定した MFS と一致する．

5.5 実験手順

本実験を行うにあたり，提案手法である F-CODE を実現するツール，および 5.4 節で述べた比較手法を実現するツールを実装する．その後実験準備で生成した組み合わせテスト結果に提案手法および比較手法を適用し，特定結果を得る．特定結果から次のメトリクスを取得する．

全特定成功数

1 回の実行において，出現したバグセットの全てを正確に特定できた回数．

単体特定成功数

1 回の特定作業において，その特定結果がバグセットと一致した回数．

単体特定失敗数

1 回の特定作業において，その特定結果がバグセットと一致しなかった回数．

追加テストケース数

1 回の実行において，特定作業を行うのに要したテストケースの総数．

表 5.3　表 4.1 に対する 2-way テスト例

テストケース	a	b	c	d	実行結果
t_1	0	0	0	0	成功
t_2	0	1	1	1	成功
t_3	0	2	1	0	成功
t_4	1	0	0	1	成功
t_5	1	1	0	0	成功
t_6	1	2	1	1	成功
t_7	2	0	1	1	失敗
t_8	2	1	0	0	成功
t_9	2	2	0	0	成功

表 5.4　OFOT 法により生成される追加テストケース

テストケース	a	b	c	d	実行結果
t_{10}	0	0	1	1	成功
t_{11}	2	1	1	1	失敗
t_{12}	2	0	0	1	成功
t_{13}	2	0	1	0	失敗

特定成功率

特定成功率は 1 回の実行において出現したバグセットの全てを正確に特定できた割合を指し，式 5.1 により算出される．

$$\text{特定成功率 [\%]} = \frac{\text{全特定成功数}}{\text{実行回数}} \times 100 \tag{5.1}$$

単体特定精度

単体特定精度は 1 回の特定作業において正しいバグセットを特定できた割合を指し，式 5.2 により算出される．

$$\text{単体特定精度 [\%]} = \frac{\text{単体特定成功数}}{\text{単体特定成功数} + \text{単体特定失敗数}} \times 100 \tag{5.2}$$

ここで 1 回の実行とは，L_{fail} にテストケースが格納されてから空になるまでの一連の操作である．また 1 回の特定作業とは，L_{fail} からテストケースを一つ取り出してバグセット特定結果を出力するまでの操作である．

また，準備した組み合わせテスト結果を用いて実験 2 を同様に行った．

6. 実験結果

6.1 実験 1

表 6.1 に各手法の特定成功率，表 6.2 に追加テストケース数の平均値，中央値，最小値，最大値を示す．平均値は小数点第 2 位以下を切り捨てて値を算出した．

特定成功率を比較すると，F-CODE は全てのシステムモデルにおいてバグセットを正確に特定することができたのに対し，比較手法では MFS を特定できたのはどのシステムモデルにおいても 10%以下であった．

追加テストケース数を比較すると，平均値と中央値において F-CODE は比較手法と比べて多かった．最小値においては F-CODE は比較手法の 3 倍以上の値となった．反対に，最大値においては比較手法の方が F-CODE に比べて非常に多かった．

6.2 実験 2

表 6.3 に特定成功率と単体特定精度を，表 6.4 に追加テストケース数の平均値，中央値，最小値，最大値を示す．bugset の値は設定したバグセットの個数を表す．ただし，比較手法は実験 1 の時点で特定成功率がかなり低かったため，実験 2 では用いなかった．

特定成功率と単体特定精度はバグセットの数が増えるにつれて下がっていることが分かる．しかし，特定成功率はバグセットの数が 3 個の場合でも 77%以上を保った．また，単体特定精度は全ての場合において 95%以上と，特定成功率と比較して高い値を示した．

追加テストケース数はバグセットの数が増えるにつれて多くなっていることが確認できるが，Tcas のみバグセットの個数が 1 個のときと比べて 2 個のときの方が低い平均値を示した．

表 6.1　バグセットが単一の場合の精度

	特定成功率 (%)	
システムモデル	F-CODE	OFOT
Tomcat	100.0	8.3
Hsqldb	100.0	8.2
Gcc	100.0	9.5
Jflex	100.0	6.4
Tcas	100.0	10.0

表 6.2　バグセットが単一の場合の追加テストケース数

システムモデル	追加テストケース数 (個)							
	F-CODE				OFOT			
	Ave.	Med.	Min.	Max.	Ave.	Med.	Min.	Max.
Tomcat	43.1	37	33	175	30.5	20	10	410
Hsqldb	49.7	43	39	161	37.5	24	12	384
Gcc	42.6	37	33	149	32.3	20	10	490
Jflex	53.1	46	42	188	40.2	26	13	507
Tcas	85.7	49	39	1,541	116.7	48	12	6,504

表 6.3　バグセットが複数の場合の精度

	特定成功率 (%)		単体特定精度 (%)	
システムモデル	bugset = 2	bugset = 3	bugset = 2	bugset = 3
Tomcat	92.8	77.6	98.3	95.7
Hsqldb	92.3	79.6	99.1	95.4
Gcc	90.8	77.0	98.2	95.7
Jflex	92.9	79.3	98.1	95.8
Tcas	94.2	86.2	99.2	95.6

表 6.4　バグセットが複数の場合の追加テストケース数

システムモデル	追加テストケース数 (個)							
	bugset = 2				bugset = 3			
	Ave.	Med.	Min.	Max.	Ave.	Med.	Min.	Max.
Tomcat	54.4	43	33	262	66.8	68	33	243
Hsqldb	65.4	51	39	184	78.1	80	39	507
Gcc	56.2	43	33	272	67.8	68	33	418
Jflex	68.4	50	42	264	82.7	86	42	440
Tcas	110.8	91	39	800	149.7	134	39	2,330

7. 考察

7.1 研究設問への回答

RQ1 提案した手法はどれほど正確に順序依存性を持つ不具合原因パラメータを特定できるか.

表 6.1 よりバグセットが単一の場合，F-CODE は全てのバグセットを正確に特定することができたことが分かる．この要因の考察は RQ3 の回答と同時に詳述する．

反対に，比較手法はほとんどバグセットを特定できなかった．比較手法がバグセットを正確に特定するためには以下の条件を満たす必要がある．

- MFS スキーマの中に一番目のパラメータが含まれている．
- MFS を含む追加テストケースを実行する前に TS を含む追加テストケースを実行する．

一つ目の条件が必要な理由として，MFS の不具合誘発には TS を含むテストケースの事前の実行が必要な本実験においては一番最初に実行されるテストケース，つまり一番最初のパラメータに変更を加えたテストケースは必ず成功する．OFOT 法では追加テストケースの実行結果が成功した場合，変更を加えたパラメータは MFS の構成パラメータであると判断されるため，一番最初のパラメータは必ず MFS を構成するパラメータとして含まれてしまう．よって MFS 構成パラメータに一番目のパラメータが無い場合必ず誤認となるため，比較手法が MFS を特定できうるのは MFS 構成パラメータに一番目のパラメータが必要である．

また MFS を含むテストケースは失敗する必要があるため，二つ目の条件が必要なのは自明である．そのためには，失敗したテストケースの中に TS が含まれている，もしくは OFOT 法によりパラメータ値を変更することによって TS が出現する必要がある．いずれの場合もその確率は高くなく，更に一つ目の条件を同時に満たす必要があることを考えると比較手法が MFS を特定する確率は低くなることが分かる．

以上より，比較手法が MFS を正確に特定することは困難であると考えられる．特定が正確に行えたとしても，それは比較手法のアルゴリズムが正常に作用したことによるものではなく上述の偶然の因子が積み重なった結果である．ゆえに，F-CODE は順序に依存する MFS 特定の正確性において有効性を示すことができたと言える．

RQ2 提案した手法はどれほど効率的に順序依存性を持つ不具合原因パラメータを特定できるか．

表 6.2 より，F-CODE において追加テストケース数は Tcas が目立って多い結果となった．表 5.1 に示した通り Tcas はテストスイートのテストケース数が他のシステムモデルと比べて非常に多いため，TS を含むテストケースの特定作業であるフェイズ 1 において必要なテストケース数が多くなったと考えられる．

他の 4 つのシステムモデルは Tcas 程大きな差はないものの，Jflex，Hsqldb，Tomcat，Gcc の順に多い結果となった．理由として，OFOT 法のアルゴリズムが関連していると考えられる．OFOT 法は失敗したテストケースのパラメータのパラメータ値を一つずつ変化させるため，パラメータの数だけ追加テストケースが生成されることになる．本実験に用いたシステムモデルのパラメータ数を比較すると，Jflex が 13 個と最も多く，Hsqldb が 12 個，Tomcat と Gcc が 10 個と続く．これは追加テストケース数の順にも一致しており，この考えを支持することが分かる．

F-CODE と比較手法を比べると，最大値以外は F-CODE の方が追加テストケース数が多い結果となった．F-CODE は MFS 特定作業に加えて TS 特定作業と TS を含むテストケースの特定作業を行うため，その分比較手法より追加テストケース数が多くなったと考えられる．最大値は比較手法が F-CODE を遥かに上回る結果となったが，これは MFS 特定が失敗した場合，失敗したテストケース全てに対して特定作業を行うためであると考えられる．特にテストスイートのテストケース数が多い Tcas では顕著である．

RQ3 提案手法は順序依存性を持つ複数の不具合原因パラメータを特定できるか．

表 6.3 より，バグセットが複数の場合 F-CODE は全てのバグセットを正確に特定することはできなかった．特定失敗要因を表 4.1 のシステムモデルを用いて説明する．

失敗要因 1：同じテストケースに複数の TS と MFS が存在する場合

TS_1 が $(a, b, c, d) = (0, -, -, -)$，$TS_2$ が $(a, b, c, d) = (-, 0, -, -)$，$MFS_1$ が $(a, b, c, d) = (0, 2, -, -)$，$MFS_2$ が $(a, b, c, d) = (-, -, 1, 0)$ である場合を考える．この場合の実行結果を表 7.1 に示す．TS_1 と TS_2 は t_1 に含まれていることからフェイズ 1 では t_1 の実行の後に MFS_1 と MFS_2 を含む t_3 は失敗し，ゆえに t_1 が TS を含むテストケース

であることが分かる．続くフェイズ 2 において生成されるテストケースを表 7.2 に示す．まず TS を含む t_1 が実行されるが，t_1 は TS_1 と TS_2 を共に含んでいるため以降の実行では MFS_1 も MFS_2 も不具合を誘発するようになる．更に t_3 は MFS_1 と MFS_2 を共に含んでいるため，OFOT 法により生成した t_{10}〜t_{13} は全て MFS_1 か MFS_2 のいずれか一方を含む．この場合 t_{10}〜t_{13} は全て失敗し，MFS 特定結果が $(a, b, c, d) = (-, -, -, -)$ となり，結果的に MFS の特定が失敗に終わる．同様の理由で TS の特定も失敗に終わる．

失敗要因 2：OFOT 法により新たな MFS が出現する場合

TS_1 が $(a, b, c, d) = (0, -, -, -)$，$TS_2$ が $(a, b, c, d) = (-, 0, -, -)$，$MFS_1$ が $(a, b, c, d) = (1, 0, -, -)$，$MFS_2$ が $(a, b, c, d) = (-, 1, 0, 1)$ である場合を考える．この場合の実行結果を表 7.3 に示す．失敗要因 1 と同様に t_1 が TS を含むテストケースであることが分かる．続くフェイズ 2 において生成されるテストケースを表 7.4 に示す．失敗要因 1 と同様に TS_1 と TS_2 を共に含む t_1 が実行され，以降の実行では MFS_1 も MFS_2 も不具合を誘発するようになる．t_4 は MFS_1 のみを含んでいるが，OFOT 法によりパラメータ b を変更した t_{11} において MFS_2 が新たに出現するため t_{11} は失敗する．この場合 MFS 特定結果が $(a, b, c, d) = (1, -, -, -)$ となり，結果的に MFS の特定が失敗する．t_4 自体に含まれている MFS は一つだけであるため TS の特定結果は正しい結果が得られる．

失敗要因 3：特定した MFS を含むテストケースの実行が省略される場合

TS_1 が $(a, b, c, d) = (0, -, -, -)$，$TS_2$ が $(a, b, c, d) = (-, 0, -, -)$，$MFS_1$ が $(a, b, c, d) = (-, 1, -, -)$，$MFS_2$ が $(a, b, c, d) = (-, 1, 0, 0)$ である場合を考える．この場合の実行結果を表 7.5 に示す．今回の例の場合では一つ目の失敗したテストケース t_3 における特定作業によって TS_1 と MFS_1 のバグセットは正しく特定される．よって，残りの失敗したテストケースである t_5 と t_8 は MFS_1 を含むため特定作業が行われない．しかし，TS_2 は t_1 で実行され，t_5 と t_8 は MFS_2 を含んでいることからこれらの失敗は TS_2 と MFS_2 のバグセットによるものでもある．このように，一方の MFS 特定結果によりもう一方の MFS 特定作業が行われないことによって，結果的に出現したバグセット全てを発見できないことが分かる．

表 7.1　失敗要因 1 に対する 2-way テスト実行結果

テストケース	a	b	c	d	実行結果
t_1	0	0	0	0	成功
t_2	0	1	1	1	成功
t_3	0	2	1	0	失敗
t_4	1	0	0	1	成功
t_5	1	1	0	0	成功
t_6	1	2	1	1	成功
t_7	2	0	1	1	成功
t_8	2	1	0	0	成功
t_9	2	2	0	0	成功

表 7.2　失敗要因 1 のフェイズ 2 において生成される追加テストケース

テストケース	a	b	c	d	実行結果
t_1	0	0	0	0	——
t_{10}	1	2	1	0	失敗
t_{11}	0	0	1	0	失敗
t_{12}	0	2	0	0	失敗
t_{13}	0	2	1	1	失敗

表 7.3　失敗要因 2 に対する 2-way テスト実行結果

テストケース	a	b	c	d	実行結果
t_1	0	0	0	0	成功
t_2	0	1	1	1	成功
t_3	0	2	1	0	成功
t_4	1	0	0	1	失敗
t_5	1	1	0	0	成功
t_6	1	2	1	1	成功
t_7	2	0	1	1	成功
t_8	2	1	0	0	成功
t_9	2	2	0	0	成功

表 7.4　失敗要因 2 のフェイズ 2 において生成される追加テストケース

テストケース	a	b	c	d	実行結果
t_1	0	0	0	0	——
t_{10}	2	0	0	1	成功
t_{11}	1	1	0	1	失敗
t_{12}	1	0	1	1	失敗
t_{13}	1	0	0	0	失敗

表 7.5　失敗要因 3 に対する 2-way テスト実行結果

テストケース	a	b	c	d	実行結果
t_1	0	0	0	0	成功
t_2	0	1	1	1	失敗
t_3	0	2	1	0	成功
t_4	1	0	0	1	成功
t_5	1	1	0	0	失敗
t_6	1	2	1	1	成功
t_7	2	0	1	1	成功
t_8	2	1	0	0	失敗
t_9	2	2	0	0	成功

表 6.3 より特定精度が特定成功率を上回っていることから，これらの中では余分な特定作業を必要としない失敗要因 3 が主な要因となっていると予想される．

またバグセットが単一である場合において，これら 3 つの要因はいずれも関与しないため，実験 1 においては特定成功率が 100%であったと考えられる．

追加テストケース数においてはバグセットの数が増えるほど多くなった．一つ目の理由として，バグセットの数が増えると，テストスイート内に実際に出現するそれらの最大個数が増加するためであると考える．二つ目の理由として，単体特定精度の低下から特定失敗数が増加していると考えられ，特定失敗時に使用された追加テストケース数が要因になったと考える．しかし，Tcas のみバグセットが 1 個のときの平均値が 2 個のときの平均値を上回った結果となった．ここで最大値を比較してみると，バグセットが 1 個のときの最大値が 1,541 であるのに対し 2 個のときの最大値が 800 であり，バグセットが 1 個のときを大きく下回る結果となった．今回の実験では TS と MFS は全てランダムに設定しており，バグセットが 1 個のときに多量のテストケース数で構成されるテストスイートの実行において，偶然，その実行順序における末尾付近で初めてテストが失敗したためであると考えられる．Tcas はテストスイートのテストケース数が他のシステムモデルと比べて非常に多く，一度このようなことが起こると平均値に大きな影響を与えることが考えられる．従って，このような結果が得られたのは実験のランダム性による偶発的なものであると考える．

7.2 妥当性への脅威

7.2.1 内的妥当性

本実験で用いた提案手法 F-CODE，および比較手法を実現するツールはいずれも筆者が作成したものであり，これらに誤りがあった場合得られる結果に差が出る可能性がある．しかし，評価実験を行う前に試運転を行い，正しく動作しているかを入念に確認することで妥当性を高めた．

7.2.2 外的妥当性

本実験では各システムモデルに対して 4-way テストスイートまでを用いて評価を行った．しかし，システムモデルの規模が大きくなればなるほど出現する MFS の大きさは大きくなることが予想され，より網羅度を高くして FIL を行う必要があると考えられる．それらのシステムモデルに対して評価実験を行った場合，本実験で得られた結論とは異なる可能性がある．

7.2.3 構成概念妥当性

今回は精度を評価する指標として特定成功率と単体特定精度を用いた．特定成功率のみで評価をすると，出現したバグセットは全て特定できたが失敗した特定作業も含まれていた場合の実行を全て成功として扱うこととなり，1 回の特定作業単位での成功率を見落とす可能性があった．そこで単体特定精度を用い，1 回の特定作業単位で評価を行うことでその可能性を排除した．

7.3 今後の課題

7.3.1 追加テストケース数の削減

F-CODE は t_{tri} を特定するフェイズ 1 の操作として，t_{fail} 以前に実行されたテストケースを 1 つずつ確認している．単純かつ確実性のある方法ではあるが，1 回の確認ごとに失敗したテストケースも実行しなければならないため冗長である．また，t_{tri} がテストスイートの後ろにあればある程，その数は比例して増加する．よって，t_{tri} の特定の確実性を保ちながらより効率的に行う操作が求められる．例えば，二分探索法を用いればテストスイートのテストケース数が多い大規模なシステムモデルでは追加テストケース数を削減できると考えられる．

また MFS と TS を特定するフェイズ 2 とフェイズ 3 の操作として，単一の失敗したテストケースのみから特定操作を行える OFOT 法を採用しているが，システムモデルのパラメータ数の分追加テストケース数が必要なため，こちらも効率的とは言えない．複数の既存の FIL 手法に対して評価実験を行うことで，より最適な手法を選定できる可能性があると考える．

7.3.2 決定的 MFS との複合

F-CODE は全ての MFS が順序依存に起因する非決定的なテスト結果を与えることを前提として開発したものである．しかし，実際のソフトウェア開発現場において出現する不具合が全てそうであるとは限らず，そのような場合に F-CODE を適用すると決定的なテスト結果を与える MFS を非決定的な結果を与える MFS であると誤認してしまう．よって，それらが混合して存在する場合にも適用ができ，なおかつそれらを判別できるような手法を開発することでその有効性が更に増すと考える．

8. 関連研究

FIL 手法は大きく二種類に分類できる．

一つは，テスト結果を基に追加テストケースを作成し FIL を行う手法である．F-CODE はこれに該当する．Zheng らによって提案された comFIL[1] は，成功したテストケースには含まれず，失敗したテストケースのみに含まれるパラメータ組を MFS の候補とし，それぞれ候補を含む追加テストケースを作成し実行することで FIL を行う．Ghandehari らによって提案された BEN[2] は，comFIL と同様に MFS 候補を挙げ，特定の式によりそれぞれの不審度を算出することで FIL を行う．BEN は一部の関連研究から有効性の高い FIL 手法として考えられている [15, 16]．Zhang らによって提案された FIC[13] は，OFOT 法と同様に一つの失敗したテストケースの一つのパラメータ値に変更を加えたテストケースをそれぞれ生成し実行することで FIL を行うが，その操作を複数回繰り返すことで，一つの失敗したテストケースに複数の MFS が存在する場合にも特定を可能にしている．Zeller らによって提案された Delta Debugging[17, 18, 19] は，成功したテストケースと失敗したテストケースの入力データを二分探索的に比較し FIL を行う．加えて，パラメータ間の伝播関係の抽出も可能にしている．

もう一つは，FIL を考慮したテストスイートを初めから設計する手法である．Coburn らによって提案された LDA[20] は，スキーマの大きさが t 以下である最大 d 個の MFS を，網羅度（d + t）のテストスイートを設計し実行することで FIL を行う．Martinez らによって提案された ELA[21] は LDA と比較して，MFS の大きさだけでなくパラメータ値の取りうる値も考える．

9. 結言

本研究では，非決定的なテスト結果が得られる原因に着目し，実行順序に依存して不具合を誘発する要因の特定に焦点を絞った新しいFIL手法であるF-CODEを提案した．

評価実験では，5つのシステムモデルを対象とする組み合わせテスト結果に提案手法と既存手法を適用させFIL結果の精度と追加テストケース数を比較した．バグセットが単一の場合と複数の場合でこれを行った．

その結果，既存手法がこれらをほとんど特定できなかったのに対し，提案手法は高い水準で特定が可能であったことからその有効性が示された．

今後の課題として，特定に要する追加テストケース数の削減や，決定的不具合誘発要因と混合したテスト結果に対する不具合検出を可能にさせることなどが挙げられる．

謝辞

本研究を行うにあたり，研究課題の設定や研究に対する姿勢，本報告書の作成に至るまで，全ての面で丁寧なご指導を頂きました，本学情報工学・人間科学系 水野修教授，崔恩瀞助教，加えて本学設計工学専攻 西浦生成先輩に厚く御礼申し上げます．本報告書執筆にあたり貴重な助言を多数頂きました，本学ソフトウェア工学研究室の皆さん，学生生活を通じて著者の支えとなった家族や友人に深く感謝致します.

参考文献

[1] D.H. Wei Zheng, Xiaoxue Wu, and Q. Zhu, “Locating minimal fault interaction in combinatorial testing,” Proceedings of the Advances in Software Engineering（ASE）, vol.2016, pp.1-10, 2016.

[2] L. S. G. Ghandehari, Y. Lei, T. Xie, R. Kuhn, and R. Kacker, “Identifying Failure-Inducing Combinations in a Combinatorial Test Set,” Proceedings of the IEEE International Conference on Software Testing, Verification and Validation （ICST）, pp.370-379, 2012.

[3] John Micco, “Flaky Tests at Google and How We Mitigate Them,” 入手先（オンライン）, 入手先〈https://testing.googleblog.com/2016/05/flaky-tests-at-google-and-how-we.html〉（参照 2020-2-09）.

[4] Q. Luo, F. Hariri, L. Eloussi, and D. Marinov, “An empirical analysis of flaky tests,” Proceedings of the 22nd ACM International Symposium on Foundations of Software Engineering. ACM, 2014, pp. 643-653.

[5] C. Nie and H. Leung, “A survey of combinatorial testing,” Proceedings of the ACM Computing Surveys, vol.43, no.2, p.11, 2011.

[6] D.R. Kuhn, R.N. Kacker, and Y. Lei, “Introduction to combinatorial testing”, Proceedings of the CRC Press, 2013.

[7] S. K. Khalsa and Y. Labiche, “An orchestrated survey of available algorithms and tools for combinatorial testing,” Proceedings of the Department of Systems and Computer Engineering Carleton University, Tech. Rep., 2014.

[8] J. Czerwonka, “Pairwise testing in the real world: Practical extensions to test case generators,” Microsoft Corporation Software Testing Technical Articles, 2008.

[9] Microsoft Corp., “Pairwise Independent Combinatorial Testing,” 入手先（オンライン）, 入手先〈http://github.com/Microsoft/pict（参照 2020-2-08）.

[10] E. Choi, T. Kitamura, C. Artho, A. Yamada, and Y. Oiwa. “Priority integration for weighted combinatorial testing,” Proceedings of the 39th Annual International

Computer Software and Applications Conference, pp.242-247, 2015.

[11] R. Bryce and C. Colbourn, "Prioritized intersction testing for pair-wise coverage with seeding and constraints," Proceedings of the Information and Software Technology（IST）, Vol. 48, No. 10, pp.960-970, 2006.

[12] C. Nie and H. Leung, "The minimal failure-causing schema of combinatorial testing,"Proceedings of the ACM Transactions on Software Engineering and Methodology (TOSEM), vol.20, no.4, p.15, 2011.

[13] Z. Zhang and J. Zhang, "Characterizing failure-causing parameter interactions by adaptive testing," Proceedings of the 2011 International Symposium on Software Testing and Analysis, pp.331-341, ISSTA '11, ACM, New York, NY, USA, 2011.

[14] X. Niu, C. Nie, H. K. Leung, Y. Lei, X. Wang, J. Xu, and Y. Wang, "An interleaving approach to combinatorial testing and failure-inducing interaction identification", Proceedings of the IEEE Transactions on Software Engineering, pp.1-33, 2018.

[15] A. Gargantini, J. Petke, M. Radavelli, Proceedings of the 2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 239-248. DOI 10.1109/ICSTW.2017.44

[16] J. Bonn, K. Foegen, H. Lichter, in 2019 IEEE International Conference on Software Testing, Verification and Validation Workshops（ICSTW）, pp. 224-233. DOI 10.1109/ICSTW.2019.00057

[17] A. Zeller, R. Hildebrandt, "Simplifying and isolating failure-inducing input," Proceedings of the IEEE Transactions on Software Engineering 28（2）, pp.183-200,2002.

[18] Z. Wang, B. Xu, L. Chen, L. Xu., Proceedings of the IEEE 10th International Conference on Quality Software（QSIC）, pp.495-502, 2010.

[19] J. Li, C. Nie, Y. Lei, Proceedings of thein 2012 12th International Conference on Quality Software（QSIC）, pp.102-105. DOI 10.1109/QSIC.2012.28.

[20] C.J. Colbourn, D.W. McClary, "Locating and detecting arrays for interaction faults," Proceedings of the Journal of combinatorial optimization 15（1）, pp.17-48, 2008.

[21] C. Martinez, L. Moura, D. Panario, B. Stevens, “Locating errors using elas, covering arrays, and adaptive testing algorithms,” Proceedings of the SIAM Journal on Discrete Mathematics 23（4）, pp.1776-1779, 2009.