情報処理学会論文誌 Vol.62 No.4 1008–1018 (Apr. 2021)

組合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

西浦 生成[1,a)] 渡辺 大輝[1,b)] 水野 修[1,c)] 崔 恩瀞[1,d)]

受付日 2020年8月3日, 採録日 2021年1月12日

概要：組合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値の組を特定すること（FIL）によって，開発者がソフトウェアシステムから不具合をとり除くための有益な情報が得られる．既存の FIL 手法はテスト結果が決定的であることを前提としている．一方で，同じテストケースの結果が一意に定まらない，非決定的なテストの存在が報告されており，既存の FIL 手法はこれに対応できない．本論文では，テスト結果が非決定的になる原因のうち実行順序に依存するものに焦点を絞り，非決定的なテスト結果に対応可能な新しい FIL 手法である F-CODE を提案した．F-CODE では，初めにテストの失敗を再現するテスト実行順序を特定し，それらを連続して実行するよう既存手法を改変することによって実行順序に依存する非決定性を排除することで，FIL 結果が正しく取得されることを可能にしている．さらに，実行順序に依存して不具合誘発要因が有効になる条件となる要因を同時に特定する．また実際のシステムに基づく人工的なテスト結果を用いた評価実験によって，提案手法の特定精度およびテストの追加実行回数の増加割合が良好であることを示した．

キーワード：ソフトウェアテスト，組合せテスト，不具合誘発要因特定

Faulty Interaction Localization Approach for Non-deterministic Failure-inducing Combination Depends on Execution Order

Kinari Nishiura[1,a)] Daiki Watanabe[1,b)] Osamu Mizuno[1,c)] Eunjong Choi[1,d)]

Received: August 3, 2020, Accepted: January 12, 2021

Abstract: Faulty interaction localization (FIL) is to locate interactions of input parameter values causing a defect from the result of the combinatorial testing. FIL provides developers useful information for removing the detect from the software system. Conventional FIL methods assume that test results are deterministic. On the other hand, the existence of non-deterministic tests has been reported, and conventional FIL methods cannot deal with them. In this paper, we focus on the causes of non-deterministic of test results that depend on the execution order, and propose a new FIL method, F-CODE, which can deal with non-deterministic test results. In F-CODE, by identifying the test execution sequence that reproduces the test failures, then customizing a conventional method to execute them consecutively, the non-determinism that depends on the execution sequence is eliminated, thus can obtain correct FIL results. Moreover, we confirmed that the identification accuracy and increase rate of the additional test execution of F-CODE were good by the evaluation experiment using the artificial test result based on real software systems.

Keywords: software testing, combinatorial testing, faulty interaction localization

1. はじめに

ソフトウェア開発において，ソフトウェアテストは製品の品質を維持するために重要な役割を果たす．一般に，1つのテストケースはテスト対象のシステムの 1 つの動作のみをテストするため，さまざまな条件下でのテストを行う

[1] 京都工芸繊維大学
Kyoto Institute of Technology, Kyoto 606–8585, Japan
[a)] k-nishiura@se.is.kit.ac.jp
[b)] d-watanabe@se.is.kit.ac.jp
[c)] o-mizuno@kit.ac.jp
[d)] echoi@kit.ac.jp

© 2021 Information Processing Society of Japan

には複数のテストケースが必要となる．ただし，大規模なソフトウェアをテストする場合，複数の入力パラメータの相互作用によって引き起こされる不具合を網羅的に検出するには膨大な数のテストケースが必要となり，そのすべてをテストすることは現実的ではない．こうした場合，組合せテストを使用することでテスト実行のコストを大幅に削減できる．組合せテストでは，特定の基準に対して入力パラメータがとる値のすべての組合せを網羅するようにテストケースを設計し，設計された一連のテストケースを使用してテストを実施する [1], [2].

また，不具合誘発要因特定（Faulty Interaction Localization，以下 FIL）は，得られた組合せテストの結果から不具合を含む動作を誘発する条件となるような入力パラメータ値の組合せを識別する工程である．開発者は組合せテストの結果を元に FIL を行って不具合動作を引き起こす条件を特定することで，欠陥のあるコンポーネントを簡単に特定し修復できる．既存研究ではいくつかの FIL 手法が提案されている [3], [4], [5], [6], [7]．例として，OFOT 法 [3] はパラメータ値を変更して追加のテストを行うことで誘発原因に関係するパラメータ値を特定する．

ただし，OFOT 法などの既存 FIL 手法では，同じテストケースからつねに同じ結果が得られると仮定している．したがって，同じテストケースの結果が一意に定まらないような非決定的なテスト結果から正しい FIL 結果を導くことは，既存の FIL 手法では非常に困難である．一方で，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 のテストに関するコミットを調査し，非決定性の要因を 10 種類に分類した [8]．さらに，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が報告されている．

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 Considered Order Dependency）を提案する．本手法は最初に，非決定的なテストケースがつねに欠陥のある動作を検出するために必要なテストケースの順序を特定する．特定されたテスト実行順序を用いてテスト結果の非決定性を排除することで，OFOT 法などの既存 FIL 手法の適用が可能となり，不具合誘発要因を特定できる．加えて本手法は，順序依存性の原因となる要因を合わせて特定する．これは，そのパラメータ値を含むテストケースを事前に実行することで，順序依存性を持つテストケースをつねに失敗させるようなものを指す．こうした依存関係を特定することで，より効果的なデバッグの提供が期待できる．

実際のソフトウェアシステムに基づく人工的なテスト結果を用いて F-CODE を評価し，次の結果が示された．

- F-CODE は，不具合を誘発する単一の要因をつねに特定できる．一方，同じ対象への単純な OFOT 法の特定成功率は 10%未満である．
- F-CODE は，テスト結果の非決定性を排除するために，従来の FIL 手法よりも多くのテストケースを追加実行する必要がある．ただし，実行回数の増加割合は，評価に使用したどのシステムでも平均で 2 倍未満である．
- 不具合を誘発する要因が複数存在する場合，F-CODE はそれらをつねに特定できるとは限らない．ただし，2 つおよび 3 つの独立した誘発要因が存在する場合に対しそれぞれ 92.6%および 79.9%と高い特定成功率を維持する．

2. 背景

2.1 *t*-way テスト

t-way テストは，組合せテストを実現する一種のテスト方法である．*t*-way テストでは，最小限のテストケース数で t 個以下のパラメータ値のすべての組合せが少なくとも 1 回テストされる．$t = 2$ の場合の *t*-way テストはペアワイズテストとしても知られる．いくつかの *t*-way テストスイート生成ツールの実装が提案されている [9]．たとえば，Czerwonka によって提案された PICT [10], [11] は，貪欲なヒューリスティックアルゴリズムを使用している．

表 1 に示す入力パラメータを持つシステムの例を考える．このシステムは p_a，p_b，p_c，p_d の 4 つの入力パラメータを有し，そのうち 2 つは 3 種類の値を，他の 2 つは 2 種類の値をとることができる．これらのパラメータ値間のすべての可能な組合せは $3 \times 3 \times 2 \times 2 = 36$ パターンあり，このすべてを網羅するには合計 36 のテストケースが必要となる．次に，PICT で生成されたこの例題システムの 2-way テストスイートを**表 2** に示す（テスト結果列は含まない）．2-way テストの定義どおり，任意の 2 つのパラメータ値の組合せが網羅されていることが確認できる．またこの例では，2-way テストスイートに含まれるテストケースは 9 つになる．このように，組合せテストを使用することで網羅基準とのトレードオフとしてテストケースの数を大幅に減らすことができる．

2.2 スキーマおよび MFS

スキーマ（Schema）は，入力パラメータ値の組合せの表

表 1 例題システムの入力パラメータ
Table 1 Input parameters of an example system.

パラメータ	とりうる値
p_a	0, 1, 2
p_b	0, 1, 2
p_c	0, 1
p_d	0, 1

© 2021 Information Processing Society of Japan

表 2 例題システムの 2-way テストスイート例
Table 2 An example 2-way test suite of an example system.

テストケース	p_a	p_b	p_c	p_d	テスト結果
t_1	0	0	0	0	失敗
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

現である．この表現は，いくつかの先行研究 [3], [7] で使用されている．たとえば，例題システムにおけるスキーマ (-, 2, 1, -) は $p_b = 2$ と $p_c = 1$ の組合せを表し，また p_a と p_d の値はこの組合せに無関係であることを意味する．また，このスキーマはテストケース t_3 と t_6 に含まれている．

最小不具合誘発要因スキーマ（Minimal Failure-inducing Schema，以下 MFS）は，テストケースに含まれていることが原因でテストが失敗する最小サイズのスキーマである．この表現は [3] で定義されている．FIL の目的は，テストの失敗を再現するための最小条件である MFS を特定することである．たとえば，$p_c = 0$ および $p_d = 0$ の場合に例題システムが失敗するとする．表 2 のテスト結果列は，この場合に得られる 2-way テストスイートのテスト結果を示している．FIL の目的は，スキーマ (-, -, 0, 0) を MFS として特定することである．また，たとえばスキーマ (-, 1, 0, 0) は (-, -, 0, 0) を含むためテストケースに含まれているとテストを失敗させるが，不具合誘発の原因となる最小の条件ではないため MFS ではない．

2.3 OFOT 法

代表的な FIL 手法として，Nie らは OFOT（One Factor One Time）法を提案している [3]．この手法では単一の失敗したテストケースを入力として使用し，そのパラメータ値を 1 つずつ変更して再テストすることで MFS を特定する．あるパラメータ値を変更することによってテスト結果が失敗から成功に変わる場合，そのパラメータ値が MFS の一部であることが分かることを利用している．

表 2 に示した組合せテスト結果への OFOT 法の適用例を以下に示す．失敗したテストケース t_1 に注目し，**表 3** に示すように各パラメータごとに値を変更した新しいテストケースを作成し実行する．その結果，t_{a1} と t_{a2} は依然としてテストに失敗するが，t_{a3} と t_{a4} はテスト結果が合格に変化した．このことから，テストケースがテストに失敗するためには，(-, -, 0, -) および (-, -, -, 0) が含まれる必要があることが分かる．したがって，(-, -, 0, 0) が MFS として特定される．これは実際にシステムが失敗する条件に一致

表 3 OFOT 法による FIL に必要となる追加テストケース
Table 3 Additional test cases in OFOT method.

テストケース	p_a	p_b	p_c	p_d	テスト結果
t_{a1}	1	0	0	0	失敗
t_{a2}	0	1	0	0	失敗
t_{a3}	0	0	1	0	成功
t_{a4}	0	0	0	1	成功

していることが確認できる．さらに，この MFS の存在によって t_1 以外のすべてのテストの失敗も説明できるため，MFS はほかに存在しないと見なすことが可能である．

2.4 Flaky Test

Google の 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 と名付けている [12]．また，Luo らは 51 件の Apache オープンソースプロジェクトから得られた 210 の Flaky Test を 10 種類の要因に分類した [8]．その結果，非同期待機，同時実行，順序依存性が上位 3 つの原因であることを明らかにした．

既存の FIL 手法では同一のテストケースからはつねに同一の結果が得られるといった仮定をおくなど，Flaky Test の存在に対応しておらず，その有効性に限界がある．我々はテスト結果の非決定性を引き起こす上位の要因のうち，順序依存性についてアルゴリズムによる解決が可能であることを本論文で示すことで，FIL 手法の持つ限界の拡張を試みる．

3. 提案手法：F-CODE

3.1 前提

手法の提案にあたり，以下の前提条件を設定する．

- テスト対象システムは，テストスイートとして構成されている複数のテストケースを順に連続して実行できる．テストスイートに含まれるすべてのテストケースの実行が終了すると，テスト対象システムの状態は初期化される．
- すべての MFS には，対応するトリガースキーマ（以下，**TS**）が 1 つ存在する．TS は，それを含むテストケースを実行すると，対応する MFS を含む後続のテストケースが失敗するスキーマを指す．対応する MFS と TS のペアを**不具合誘発条件対**と呼称する．
- MFS を含むテストケースは，対応する TS を含む 1 つ以上のテストケースがそれ以前に実行されているときに限り必ず失敗する．そうでない場合には必ず成功する．

3.2 アルゴリズム

F-CODE は実行されたテストスイートとその結果を入力として受け取り，特定されたすべての不具合誘発条件対を

© 2021 Information Processing Society of Japan

出力する．失敗したテストケースのリスト L_{fail} からテストケースを取り出し，そこに含まれる MFS および対応する TS の特定作業に移る．特定作業は以下の 3 つのフェイズで構成される．ただし，すでに発見された不具合誘発条件対がそのテストの失敗を説明できる場合，特定をスキップする．

フェイズ 1：TS を含むテストケースの特定 失敗したテストケースを t_{fail} とする．このフェーズでは，t_{fail} に含まれる MFS に対応する TS を含むテストケースを特定する．入力された一連のテストケースで t_{fail} よりも前に実行されるテストケースを順に選び，そのテストケースと t_{fail} のペアを続けて実行する．t_{fail} がテストに失敗した場合，TS は選択したテストケースに含まれている．このテストケースを t_{trg} とする．

フェイズ 2：MFS の特定 このフェイズでは，次のように OFOT 法を適用することにより，t_{fail} に含まれる MFS を識別する．OFOT 法によっていずれかのパラメータ値が変更された t_{fail} を実行する際に，t_{trg} の実行に続けて実行させる．これにより，t_{fail} のテスト結果の非決定性が排除されるため，OFOT 法が適切に機能する．

フェイズ 3：TS の特定 このフェイズでは，フェイズ 2 と同様に t_{trg} に OFOT 法を適用することにより，TS を特定する．具体的には，OFOT 法によって変更された t_{trg} に続いて実行される t_{fail} の実行結果の変化によって TS を特定できる．

フェイズ 1 およびフェイズ 3 では，特定のテストケースによって MFS が有効化されるかどうかを調べるため，TS を含むと思われるテストケースと MFS を含むと思われるテストケースの 2 つのみを連続して実行する必要がある．対してフェイズ 2 では，TS が含まれるテストケースに続けて，追加生成されたすべてのテストケースを続けて実行できる．これは，追加実行するテストケースが MFS を含むか否かのみによってテストの成否が決定され，他の追加実行するテストケースの存在に依存しないためである．

これらのフェイズを通じて，t_{fail} の失敗を説明する不具合誘発条件対が得られる．L_{fail} がまだ空でない場合，F-CODE は L_{fail} から次の t_{fail} を取得し，その t_{fail} に対してフェイズ 1 から 3 を再度実行して，新しい不具合誘発条件対を特定する．これを L_{fail} が空になるまで繰り返すことで，すべてのテストの失敗を説明するのに十分なだけの不具合誘発条件対が得られる．

3.3 適用例

表 4 に示されたテスト結果に対して F-CODE を適用する．これは，2 章で扱った例題システムの 2-way テストスイートであり，MFS も同じく (-, -, 0, 0) である．ただし，この MFS は順序依存性を有しており，対応する TS である (-, 2, 1, -) を含むテストケースが実行されない限り，テストは失敗しない．TS を含むテストケースは t_3 で初めて実行されるため，t_1 は MFS を含むにもかかわらずテストに失敗していない．

表 4 F-CODE 適用例への入力テスト結果
Table 4 Input test result used for application example of F-CODE.

テストケース	p_a	p_b	p_c	p_d	テスト結果
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

表 5 F-CODE 適用例における各フェイズで実行されるテストケースおよびその結果
Table 5 Executed test cases and their results in each phase of application example of F-CODE.

フェイズ	テストケース	p_a	p_b	p_c	p_d	テスト結果
フェイズ 1	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
フェイズ 2	t_3	0	2	1	0	-
	t_{a1}	2	1	0	0	失敗
	t_{a2}	1	2	0	0	失敗
	t_{a3}	1	1	1	0	成功
	t_{a4}	1	1	0	1	成功
フェイズ 3	t_{a5}	1	2	1	0	-
	t_5	1	1	0	0	失敗
	t_{a6}	0	0	1	0	-
	t_5	1	1	0	0	成功
	t_{a7}	0	2	0	0	-
	t_5	1	1	0	0	成功
	t_{a8}	0	2	1	1	-
	t_5	1	1	0	0	失敗

F-CODE はまず，失敗したテストケースのリスト $L_{fail} = \{t_5, t_8, t_9\}$ を取得し，最初に t_5 を取り出す．**表 5** は各フェイズで作成および実行されたテストケースを示している．フェイズ 1 では，t_5 を失敗させるテストケースを特定するために，t_5 より前に実行されたテストケースと t_5 を続けて順に実行する．t_3 と t_5 を続けて実行した場合に初めて t_5 が失敗したので，t_3 には，t_5 の MFS を誘発する TS が含まれていることが分かる．フェイズ 2 では，OFOT 法を適用するため t_5 の一部を変更した追加テストケースとし

© 2021 Information Processing Society of Japan

て t_{a1} から t_{a4} を作成し，t_3 に続けてそれらを順に実行する．$t_{a1}/t_{a2}/t_{a3}/t_{a4}$ の各テストケースはまったく依存がなく独立に実行される．この例では t_{a1} や，t_{a2} の実行で失敗しているが，そのあとに実行される t_{a3} や t_{a4} の実行に影響しない．したがってそれらのテストケースの実行は成功する．t_{a3} および t_{a4} がテストに通過したことから，t_5 に含まれる MFS は (-, -, 0, 0) であることが分かる．フェイズ 3 では，TS が含まれる t_3 に OFOT 法を適用するためその一部を変更した t_{a5} から t_{a8} を作成し，各テストケースと t_5 が連続するよう順に実行する．t_{a6} および t_{a7} がテストに通過したことから，t_3 に含まれる TS は (-, 2, 1, -) であることが分かる．

こうして発見された，MFS が (-, -, 0, 0) かつ TS が (-, 2, 1, -) という不具合誘発条件対は，他の失敗したテストケース t_8 および t_9 の失敗を説明できる．したがって L_{fail} が空になり，F-CODE は動作を終了する．この適用例を通じて，F-CODE は 8 つの追加テストケースを作成し，19 のテストケースを追加実行した．

4. 実験準備

4.1 概要

F-CODE の性能を評価するために 2 つの実験を行った．

実験 1 システムに単一の不具合誘発条件対が存在する場合の組合せテストの結果に F-CODE を適用し，その不具合誘発条件対を実際に特定できたかを調べることで，F-CODE の精度，および特定に必要な追加テスト実行数を調査する．実験対象には，いくつかの実際のシステムの入力モデルから人工的にランダムベースで生成された多くの組合せテスト結果を使用する．さらに，F-CODE の結果を従来の FIL メソッドの結果と比較する．

実験 2 F-CODE は単一の不具合誘発条件対の存在を前提として提案されているが，複数の不具合誘発条件対が独立に存在する場合に対してもある程度の効果があることが期待できる．このことを確認するため，特定すべき不具合誘発条件対が単一ではなく複数存在する場合について，実験 1 と同様に評価する．

4.2 実験対象

これまでの先行研究 [5], [7], [13] 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．また先行研究で行われているように，わずか数件の評価結果では結果を一般化できない．こうした理由をふまえ，実際のシステムにおける架空の不具合を多数想定し，それを元に得たテスト結果に提案手法を適用することで評価を行う．この不具合の想定は 4.3 節で詳述する．

表 6 テスト対象システムの情報
Table 6 Informations of systems under test.

システム名	入力パラメータモデル	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 7 各 t-way テストスイートを構成するテストケース数
Table 7 The number of test cases composed each t-way test suite.

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

テスト対象となるシステムの入力モデルを**表 6** に示す．これらのシステムは，先行研究 [7] で実験対象として使用されていたことから本実験でも使用した．また組合せテスト作成ツールとして，実テストで広く用いられている PICT を使用した．PICT によって各システムの t-way テストスイートを生成すると，テストケース数は t のそれぞれの値に対して**表 7** のようになった．

4.3 準備

まず，テスト対象のシステムごとに t-way テストスイート（$t = 2, 3, 4$）を作成する．次に，各システムごとに MFS と TS をランダムに選択することで，想定する不具合誘発条件対を設定する．ここで，MFS と TS のスキーマサイズはどちらも 4 以下であり，MFS と TS は同一でないよう選択する．続けて，設定した不具合誘発条件対の存在を想定した組合せテストの結果を生成する．生成には，設定した不具合誘発条件対による不具合の検出を最小限の網羅度 t で行える t-way テストを使用する．このような t の値は以下の方法で得られる．

- $t = 2$ の場合から順に，t-way テストの結果を確認する．テストケースが 1 つでも失敗した場合は，その t-way テストの結果を実験に用いる．一方，すべてのテストケースが合格した場合は，続けて $(t+1)$-way テストの結果を確認する．4-way テストでもすべてのテストケースが成功した場合，設定した不具合誘発条件対は使用せず，もう 1 度ランダムに不具合誘発条件対を設定する．

表 1 に示した例題システムを用いて，本節で述べる組合せテスト結果生成の例を示す．制約を考慮して MFS およ

© 2021 Information Processing Society of Japan

び対応する TS をランダムに選択した結果として，3.3 節の適用例と同様に，MFS として (-, -, 0, 0) が，TS として (-, 2, 1, -) が選択されたとすると，これらがシステムに存在すると想定した場合の 2-way テスト結果として表 4 に示したものと同じテスト結果が得られる．このテストスイートのうち t_5，t_7，t_8 がテストに失敗したので，生成されたテスト結果は FIL 能力を評価する実験に使用できる．また別の場合として，MFS として (0, 0, 1, -) が，対応する TS として (-, 2, 1, -) が選択された場合，用意した 2-way テストスイートの中のどのテストケースも失敗しないため，順に 3-way テストスイートおよび 4-way テストスイートについて，失敗するテストケースが存在するか確認する必要がある．

実験 1 のため，対象システムごとに，単一の不具合誘発条件対の存在を想定した 1,000 個の組合せテスト結果を重複なく収集する．実験 2 のため，対象システムごとに，2 種類または 3 種類の不具合誘発条件対の存在を想定した 1,000 個の組合せテスト結果を重複なく収集する．

4.4 比較手法

単純な OFOT 法を比較手法として使用した．ただし，ここでは失敗したテストケースの一部のパラメータ値を変更した追加テストケースを前もって必要なだけ作成し，それらを連続して実行するという方法をとる．これは，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連続した実行が必要なためである．

4.5 評価指標

準備した組合せテスト結果に提案手法と比較手法を適用し，以下の評価指標を得た．

特定精度 すべての正しい不具合誘発条件対を特定した FIL の成功率を表す．以下の式によって計算される．

$$\text{特定精度} = \frac{\text{FIL 成功数}}{\text{FIL 実施件数（= 1,000）}}$$

本実験では，FIL は，その出力結果がそれまでに実行されたテストケースに含まれる想定した不具合誘発条件対とすべて一致する場合に成功と判断される．これは想定された複数の不具合誘発条件対のうちの一部が，PICT によって設計されたテストスイート，および，FIL の過程で追加実行されるテストケースに 1 度も含まれない場合，その不具合誘発条件対を特定対象と見なさず，実行されたテストケースに含まれていた不具合誘発条件対のみを特定対象として評価すべきという思想に基づく．

追加テストケース数 FIL を達成するために必要となる追加のテストケースの実行回数を表す．

特定再現率 1 度の FIL において，無関係であるにもかかわらず誤って不具合誘発条件対と特定されたものの少なさを表す．0 から 1 の値をとり，1 に近いほど正しい不具合誘発条件対のみを特定結果として検出できたことを表す．この評価尺度は実験 2 でのみ使用される．以下の式によって計算される．

$$\text{特定再現率} = \frac{\text{正しい不具合誘発条件対数}}{\text{出力された全不具合誘発条件対数}}$$

5. 実験結果

5.1 実験 1

表 8 に，単一の不具合誘発条件対が存在する場合の各システムの組合せテスト結果を対象とした提案手法および比較手法の特定精度を示す．**表 9** に，同様の場合における追加実行されたテストケース数の平均値，中央値，最小値，および最大値を示す．ここから，以下の結果を観察できる．

- F-CODE はすべての場合で単一の不具合誘発条件対を完全に特定することができた．一方，比較手法である単純な OFOT 法の特定精度は，どのシステムのテスト結果を対象とした場合でも 0.1 以下にとどまった．
- F-CODE は Tcas を除くすべての対象システムで，比較手法よりも多くの追加テストケースの実行を必要とした．F-CODE が必要とする追加テストケース数は，単純な OFOT 法と比較して全体の平均で 107%，中央値の平均で 154%となった．ただし，最大値は比較手法がはるかに高くなった．

5.2 実験 2

順序依存性による非決定的なテスト結果に対する比較手法のパフォーマンスが非常に低いことが実験 1 で示されているため，実験 2 では F-CODE にのみ焦点を当てる．

表 8 実験 1 における結果：特定精度の比較
Table 8 The result of experiment 1: the localization accuracy.

システム名	F-CODE	OFOT 法
Tomcat	1.000	0.083
Hsqldb	1.000	0.082
Gcc	1.000	0.095
Jflex	1.000	0.064
Tcas	1.000	0.100
Ave.	1.000	0.085

表 9 実験 1 における結果：追加テストケース数の比較
Table 9 The result of experiment 1: the number of additional test cases.

システム名	F-CODE				OFOT 法			
	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	33	188	40.2	26	13	507
Tcas	85.7	49	33	1,541	116.7	48	12	6,504
Ave.	54.8	42.4	-	-	51.4	27.6	-	-

© 2021 Information Processing Society of Japan

表 10 実験 2 における結果：不具合誘発条件対が n 個存在する場合の F-CODE の特定精度および平均特定再現率

Table 10 The result of experiment 2: the localization accuracy and the average of recall of F-CODE with n fault-inducing pairs.

システム名	特定精度		平均特定再現率	
	$n=2$	$n=3$	$n=2$	$n=3$
Tomcat	0.928	0.776	0.983	0.957
Hsqldb	0.923	0.796	0.991	0.954
Gcc	0.908	0.770	0.982	0.957
Jflex	0.929	0.793	0.981	0.958
Tcas	0.942	0.862	0.992	0.956
Ave.	0.926	0.799	0.986	0.956

表 11 実験 2 における結果：不具合誘発条件対が n 個存在する場合の F-CODE の追加テストケース数

Table 11 The result of experiment 2: the number of additional test cases of F-CODE with n fault-inducing pairs.

システム名	$n=2$				$n=3$			
	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
Ave.	71.1	57.4	-	-	89.0	87.2	-	-

表 10 に，不具合誘発条件対が 2 個および 3 個存在する場合の F-CODE の特定精度および平均特定再現率を示す．**表 11** に，同様の場合における F-CODE で追加実行されたテストケース数を示す．ここから，以下の結果を観察できる．

- 複数の不具合誘発条件対が存在する場合，F-CODE は完全な特定を達成できなかった．また不具合誘発条件対の数が増えるごとに特定精度が低下した．ただし，3 つの不具合誘発条件対が存在する場合でも，特定精度は極端に低くなることはなく，たとえば 77%を下回ることはなかった．
- どの対象システムでも，平均特定再現率は 95%を超えた．したがって，F-CODE の特定精度が低い場合でも，誤った不具合誘発条件対を特定結果として出力することはほとんどない．
- 不具合誘発条件対の数が増えると，追加実行されるテストケース数が増えた．実行数は不具合誘発条件対が 1 つの場合と比較して，不具合誘発条件対が 2 つの場合では平均 130%，3 つの場合では平均 162%増加した．

6. 議論

6.1 比較手法の特定失敗に関する原因分析

実験 1 の結果から，F-CODE はすべての場合で不具合誘発条件対を特定できたが，比較手法である単純な OFOT 法ではほとんどの場合で特定に失敗した．本節では，比較手法が特定に失敗した理由を分析する．

OFOT 法のメカニズム，および，4.4 節で示した我々の運用を考慮すると，使用した比較手法は次の特性を持つ．

- 連続して実行される追加テストケースのうち最初のテストケースはつねにテストに合格する．これは，仮に最初のテストケースに MFS が含まれていた場合でも，そのテストケースが失敗するためには，それ以前に対応する TS を含むテストケースが実行されていることが必要となるからである．しかし，純粋な OFOT 法では TS の存在を想定していないためそれが保証されず，最初のテストケースに MFS が含まれていたとしても，テストに合格することになる．したがって，最初の追加テストケースが，失敗したテストケースの n 番目（通常は 1 番目）のパラメータの値を変更したものである場合，失敗したテストケースの n 番目のパラメータの値が自動的に MFS の一部と見なされてしまい，誤検出の発生原因となる．
- MFS を含む追加テストケースが失敗するためには，それ以前に実行される追加テストケースの中に対応する TS が偶然に含まれる必要があるが，多くの場合これは低い確率でしか起こらない．OFOT 法において MFS を含むにもかかわらず失敗しないテストケースが存在する場合，誤検出の発生原因となる．

以上を理由として，単純な OFOT 法は実行順序に依存するテスト結果の FIL に適さないといえる．加えて，運良く MFS を特定できたとしても，対応する TS を特定するメカニズムを持たないため，不具合の発生を再現する正確な条件が得られず，結果が無意味となる可能性が高い．

6.2 追加テストケース数に関する考察

表 9 に示す実験 1 の結果から，F-CODE および単純な OFOT 法が必要とするテストケースの追加実行数を比較すると，だいたい同程度の平均値および中央値となっていた．ただし多くの場合，F-CODE は単純な OFOT 法よりも 10 から 25 程度多くの追加テストケースの実行を必要とした．これは，F-CODE が，OFOT 法では表 3 に，F-CODE では表 5 のフェイズ 2 に示したような MFS の特定作業に加えて，表 5 のフェイズ 1 に示す TS の特定，およびフェイズ 3 に示す TS を含むテストケースの特定にもテストの追加実行を必要とすることが原因である．フェイズ 1 における追加実行数はテストスイート内での失敗したテストケースの位置に依存し，フェイズ 3 における追加実行数はテストケースのパラメータ数に依存する．

次に，対象システムによる結果の差について考察する．実験に使用したテスト対象システムはパラメータ数にほぼ差がないため，提案手法のフェイズ 2 およびフェイズ 3 で発生する追加テストケース数にもあまり差は出ず，フェイ

© 2021 Information Processing Society of Japan

ズ 1 で発生する追加テストケース数に大きく依存することになる．前述のとおり，この数はテストスイート内での失敗したテストケースの位置に依存する．したがって提案手法では，追加実行数の最大値および平均値はテストスイートを構成するテストケース数に依存することが分かる．たとえば，表 9 における Tcas に F-CODE を適用した際の追加実行数の最大値である 1,541 は，表 7 に示した Tcas の 4-way テストスイートを構成するテストケース数である 1,382 という数に近く，使用した 4-way テストスイートの最後尾付近のテストケースが初めて失敗したことに由来することが分かる．一方で，比較手法では提案手法におけるフェイズ 2 の処理しか行わないため，追加実行数は単純にシステムのパラメータ数に依存する．ただし，比較手法は特定精度が低いため，他のテスト失敗を説明できる不具合誘発条件対を見つけることができず，その場合は特定作業を何度も繰り返すことになる．最大値で比較した場合に比較手法が上回ったのはこのことが原因だと思われる．加えて，Tcas ではテストスイートに含まれるテストケース数が突出して多いため，最大値も高くなり，それによって Tcas では比較手法の追加実行数の平均値が提案手法を上回ったのだと考えられる．

さらに実験 2 の結果から，特定すべき不具合誘発条件対の数が増えると，F-CODE が要する追加実行回数も増えることも分かった．この単純な理由は，複数の不具合誘発条件対を特定するためにフェイズ 1 からフェイズ 3 の処理を複数回繰り返す必要があるためである．

6.3 複数の要因に対する提案手法の特定失敗に関する原因分析

実験 2 の結果から，複数の不具合誘発条件対が存在する場合，F-CODE はそのすべてを特定できない場合があった．特定に失敗した数十件のケースを目視で確認し，失敗の原因を以下の 3 種類に分類した．簡単のため，具体例の説明には表 1 に示した例題システムおよび表 2 に示したその 2-way テストに置き換えたものを用いる．

6.3.1 複数の MFS が同じテストケースに含まれる場合

同じテストケースに複数の MFS が含まれ，そのテストケースのみが失敗し，またそれらに対応する TS が両方ともフェイズ 1 で特定されたテストケースに含まれている場合，F-CODE による特定は失敗する．これはフェイズ 2 において，OFOT 法によるパラメータ値の変更によって片方の MFS がとり除かれたとしても，別の MFS によってテストが失敗することで OFOT 法による特定を失敗させるためである．

たとえば，MFS1-TS1 と MFS2-TS2 という 2 つの不具合誘発条件対が存在し，TS1 と TS2 は共通の (0, -, -, -) であるとする．また MFS1 が (0, 2, -, -)，MFS2 が (-, -, 1, 0) とする．この場合のテスト結果を**表 12** に示す．F-CODE を適用すると，まずフェイズ 1 が適切に機能し，t_1 が先に実行されることで t_3 が失敗することを特定する．次に，フェイズ 2 が**表 13** に示すように動作したとする．t_{a1} と t_{a2} には MFS1 は存在しないが，MFS2 が存在するため失敗する．同様に，t_{a3} と t_{a4} には MFS2 は存在しないが，MFS1 が存在するため失敗する．t_{a1} から t_{a4} がすべて失敗したため，MFS が (-, -, -, -) である，つまり無条件でテストが失敗すると判断されるが，これは誤った結果となる．

表 12 原因 1・原因 2 の説明で用いる FIL の対象となるテストスイートおよびその結果

Table 12 FIL targeted test suite and the results used for explanation of cause 1 and cause 2.

テストケース	p_a	p_b	p_c	p_d	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 13 原因 1 の説明で用いるフェイズ 2 で追加実行されたテストスイートおよびその結果

Table 13 Additional executed test suite and the results in phase 2 used for explanation of cause 1.

テストケース	p_a	p_b	p_c	p_d	結果
t_1	0	0	0	0	-
t_{a1}	1	2	1	0	失敗
t_{a2}	0	0	1	0	失敗
t_{a3}	0	2	0	0	失敗
t_{a4}	0	2	1	1	失敗

この原因に対して効果的な改善を行うことは難しく，一度に複数のパラメータを改変して追加テストケースを設計するような，OFOT 法のメカニズムに代わる方法が必要である．

6.3.2 OFOT 法の過程で新たな MFS が混入する場合

フェイズ 2 において，t_{fail} の一部のパラメータ値を変更することで別の MFS が含まれてしまい，かつその別の MFS に対応する TS がフェイズ 1 で特定された t_{trg} に含まれている場合，テスト結果は成功に変化せず，変更したパラメータ値が MFS とは無関係であると判断されてしまうため，F-CODE は特定に失敗する．

たとえば，前節の例と同様に MFS1-TS1 と MFS2-TS2 という 2 つの不具合誘発条件対が存在し，TS1 と TS2 は共通の (0, -, -, -) であるとする．また MFS1 が (0, 2, -, -)，MFS2 が (-, 0, 1, 0) である場合を考える．テスト結果は，これも前節の例と同様に表 12 のとおり得られている．ま

© 2021 Information Processing Society of Japan

表 14 原因 2 の説明で用いるフェイズ 2 で追加実行されたテストスイートおよびその結果

Table 14 Additional executed test suite and the results in phase 2 used for explanation of cause 2.

テストケース	p_a	p_b	p_c	p_d	結果
t_1	0	0	0	0	-
t_{a1}	1	2	1	0	成功
t_{a2}	0	0	1	0	失敗
t_{a3}	0	2	0	0	失敗
t_{a4}	0	2	1	1	失敗

表 15 原因 3 の説明で用いる FIL の対象となるテストスイートおよびその結果

Table 15 FIL targeted test suite and the results used for explanation of cause 3.

テストケース	p_a	p_b	p_c	p_d	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

たこれも同様に，フェイズ 1 は適切に機能し，t_1 が先に実行されることで t_3 が失敗することを特定できる．続くフェイズ 2 が，**表 14** に示すように動作した場合を考える．t_{a1} で p_a の値を 0 から 1 に変更すると，MFS1 が削除され，テスト結果が成功に変わる．しかし，t_{a2} で，p_b の値を 2 から 0 に変更すると，MFS1 は削除されるが，新たに MFS2 が含まれるようになるため，結果は失敗のままとなる．t_{a3} と t_{a4} は MFS1 を削除しないため失敗のままである．t_{a2}，t_{a3}，t_{a4} が失敗したという結果から (0, -, -, -) が MFS と判断されるが，これは誤った結果となる．

こちらも OFOT 法のメカニズムに由来する失敗原因であるため，こうした問題に効果的な別の FIL 手法を開発し取り入れない限り，根本的な改善は不可能である．

6.3.3 発見済みの不具合誘発条件対によって未発見の不具合誘発条件対が隠される場合

F-CODE では効率を最大化するため，すでに発見された不具合誘発条件対で説明できるテストの失敗に対する特定作業をスキップする．ただし，スキップされたテストケースに別の MFS が含まれている場合，そちらについては出力されず，したがって完全な特定には至らない．

たとえば，これまでの原因と同じく 2 種類の不具合誘発条件対が存在し，TS は (0, -, -, -) で共通している．また MFS1 が (-, 1, -, -)，および MFS2 が (1, -, 0, 0) とする．この場合のテスト結果を**表 15** に示す．t_2，t_5，t_8 の 3 つのテストケースがテストに失敗する．このうち t_2 と t_8 には MFS1 のみが含まれ，t_5 には MFS1 と MFS2 の両方が含まれている．まず，t_2 を対象に F-CODE による不具合誘発要因特定を行う．その結果，t_2 を失敗に導いた MFS1-TS1 について問題なく特定される．さらに，t_5 および t_8 にも MFS1 が含まれていることが分かるため，t_5 および t_8 のテスト失敗はどちらも MFS1-TS1 の存在によって説明できることになる．したがって F-CODE は t_5 および t_8 に対する特定作業をスキップし，MFS1-TS1 だけを報告して動作を完了する．しかし実際には t_5 には MFS2 も含まれておりこちらは特定されていないため，ソフトウェアの修正によって MFS1-TS1 だけが誘発する不具合箇所がとり除かれたとしても，t_5 は MFS2-TS2 によって失敗してしまう．

この原因による特定失敗はスキップ機能を無効にすることで改善できるが，代償として追加テストケース実行回数は大幅に増加すると考えられる．

7. 妥当性への脅威

7.1 内的妥当性

実験に用いた提案手法 F-CODE および比較手法を実現するプログラムはいずれも著者が作成したものであり，実装に誤りがあった場合に結果に影響をもたらす可能性がある．したがって事前に試運転を行い，動作の正しさを入念に確認することで妥当性を高めた．

7.2 外的妥当性

実験では，13 個を上限とする入力パラメータ数を持つシステムを対象とした 4-way テスト以下の t-way テストを用い，一度に最大 3 つの不具合誘発条件対を想定したテスト結果を実験対象とした．そのため，これらの上限を上回る実験対象を用いた場合は異なる実験結果が得られる可能性がある．ただし，妥当性の確保のため，先行研究で用いられたものと同じ 5 種類のテスト対象システムを使用した．

7.3 構成概念妥当性

我々の提案および評価は，3 章に示した前提に基づいている．そのため，提案手法は前提に従わない順序依存性を持つ非決定的なテスト結果の FIL に適していない可能性がある．たとえば他のバリエーションとして，1 つの MFS に対応する TS が複数存在する場合や，TS に依存せず単に実行回数に依存する場合が考えられる．ただし，我々の前提は実行順序への依存の形式として最も一般的なものであり，今回我々が提案したアイデアを適切に拡張することで個別に対応が可能であると考えている．

8. 関連研究

これまでにいくつかの FIL 手法が提案されている．これらは大きく 2 種類に分類できる．

© 2021 Information Processing Society of Japan

1つは，事前に設計され実行された組合せテストの結果に基づいてテストを追加実行することでFILを達成する方法である．OFOT法 [3] および我々の提案手法F-CODEはこれに該当する．以下，この区分に含まれる他の代表的な手法を紹介する．Zhangらによる FIC法 [4] はOFOT法の拡張であり，反復作業によって複数の要因を特定できる．FIC法の仕組みを取り入れることで，特定対象として複数の要因が含まれる場合のF-CODEの動作を改善できる可能性がある．Zhengらによる comFIL [13] は，失敗したテストケースにのみ含まれるパラメータ値の組合せをMFSの候補として保持し，追加のテストケースを実行することで候補を絞り込んでいく．Ghandihariらによる BEN [5], [14], [15] は，特定の計算式によって各MFS候補の不審さを計算することで，優先付けによって効率的な特定を行う．Nishiuraらは，ロジスティック回帰分析を利用して複雑なMFSを予測的に特定する方法を提案した [16]．Zellerらによるデルタデバッギング [6] やその改良手法 [17], [18] は，厳密には組合せテストの結果を対象とするFIL手法ではないが，新たなテストの作成と実行に基づいてプログラムの不具合原因を特定する部分で共通している．

もう1つは，テスト結果から不具合誘発要因を特定できるよう拡張された組合せテストスイートを事前に作成する方法である．Colbournらによって提案されたLDA [19] は，組合せサイズ $d+t$ を網羅するテストスイートを設計することで，t 以下のパラメータ値で構成された d 個以下のMFSをテスト結果から特定できる．Martínezらによる ELA [20] はその拡張であり，パラメータのとりうる値の数も考慮して設計する．こうしたアプローチはテストスイートの設計および実行プロセスとFILプロセスを完全に分離できる一方，MFSの数と大きさに対する仮定が必要であること，また単純なテストスイートに比べてテストケースの数が大幅に増えることが不利益となる．

こうした既存手法はいずれも非決定的なテスト結果を処理できるよう構築されておらず，我々の提案には大きな新規性がある．また，Niuらの研究 [7] は，提案手法を評価する際にテスト結果の非決定性を考慮した我々の知る唯一の研究である．彼らは同じテストケースを複数回実行することで非決定性の緩和を試みたが，この方法は確率的ゆえに不確実であり，また実行順序ではなく完全なランダム性に由来する非決定性にのみ機能する．

9. おわりに

本研究では，FIL手法の非決定的なテスト結果への拡張として，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種類のシステムを対象とする組合せテスト結果に提案手法と既存手法を適用し，FILの精度と追加テストケース数を比較した．また不具合要因が単一の場合と複数の場合でこれを行った．その結果，既存手法がほぼすべての特定に失敗した一方で，提案手法は単一の要因に関しては完全に特定でき，また複数の要因についても高い水準で特定が可能であることが示された．また，本手法によって追加されるテストケース数についても極端に多くはならないことが示された．今後の発展として，複数の要因により対応できるよう，また必要な追加テストケースの数を減らすために，アルゴリズムを改善することが望まれる．

謝辞 本研究はJSPS科研費 JP19K20240，JP18H04094の助成を受けた．

参考文献

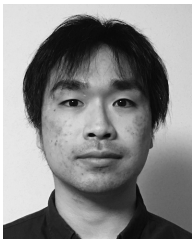
[1] Nie, C. and Leung, H.: A survey of combinatorial testing, *ACM Computing Surveys* (2011).
[2] Kuhn, D.R., Kacker, R.N. and Lei, Y.: *Introduction to combinatorial testing*, CRC Press (2013).
[3] Nie, C. and Leung, H.: The Minimal Failure-Causing Schema of Combinatorial Testing, *ACM Trans. Softw. Eng. Methodol.*, Vol.20, No.4, pp.15:1–15:38 (online), DOI: 10.1145/2000799.2000801 (2011).
[4] Zhang, J., Ma, F. and Zhang., Z.: Faulty interaction identification via constraint solving and optimization, *Proc. International Conference on Theory and Applications of Satisfiability Testing* (*SAT*), pp.186–199 (2012).
[5] Ghandehari, L.S.G., Lei, Y., Xie, T., Kuhn, R. and Kacker, R.: Identifying failure-inducing combinations in a combinatorial test set, *Proc. IEEE 5th International Conference on Software Testing, Verification and Validation* (*ICST*), pp.370–379 (2012).
[6] Zeller, A. and Hildebrandt, R.: Simplifying and isolating failure-inducing input, *IEEE Trans. Software Engineering*, Vol.28, No.2, pp.183–200 (2002).
[7] Niu, X., n. changhai, Leung, H.K.N., Lei, Y., Wang, X., Xu, J. and Wang, Y.: An interleaving approach to combinatorial testing and failure-inducing interaction identification, *IEEE Trans. Software Engineering*, p.1 (2018).
[8] Luo, Q., Hariri, F., Eloussi, L. and Marinov, D.: An Empirical Analysis of Flaky Tests, *Proc. 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, *FSE 2014*, pp.643–653 (2014).
[9] Khalsa, S.K. and Labiche, Y.: An Orchestrated Survey of Available Algorithms and Tools for Combinatorial Testing, *2014 IEEE 25th International Symposium on Software Reliability Engineering*, pp.323–334 (2014).
[10] PICT: Pairwise Independent Combinatorial Tool (online), available from ⟨http://github.com/Microsoft/pict⟩ (accessed 2020-07-30).
[11] Czerwonka, J.: Pairwise testing in the real world: Practical extensions to test-case scenarios (online), available from ⟨https://msdn.microsoft.com/en-us/library/cc150619.aspx⟩ (accessed 2020-07-30).
[12] Micco, J.: Flaky Tests at Google and How We Mitigate Them (online), available from ⟨https://testing.googleblog.com/2016/05/flaky-tests-at-google-and-how-we.html⟩ (accessed 2020-07-30).
[13] Zheng, W., Wu, X., Hu, D. and Zhu, Q.: Locating Minimal Fault Interaction in Combinatorial Testing, *Advances in Software Engineering*, Vol.2016, p.10 (2016).

© 2021 Information Processing Society of Japan

[14] Ghandehari, L.S., Chandrasekaran, J., Lei, Y., Kacker, R. and Kuhn, D.R.: BEN: A combinatorial testing-based fault localization tool, *Proc. IEEE 8th International Conference on Software Testing, Verification and Validation Workshops* (*ICSTW*), pp.1–4 (2015).
[15] Sh. Ghandehari, L., Lei, Y., Kacker, R., Kuhn, D.R.R., Kung, D. and Xie, T.: A Combinatorial Testing-Based Approach to Fault Localization, *IEEE Trans. Software Engineering*, p.1 (2018).
[16] Nishiura, K., Choi, E. and Mizuno, O.: Improving Faulty Interaction Localization Using Logistic Regression, *Proc. 2017 IEEE International Conference on Software Quality, Reliability & Security* (*QRS2017*), pp.138–149 (2017).
[17] Wang, Z., Xu, B., Chen, L. and Xu., L.: Adaptive Interaction Fault Location Based on Combinatorial Testing, *Proc. IEEE 10th International Conference on Quality Software* (*QSIC*), pp.495–502 (2010).
[18] Li, J., Nie, C. and Lei, Y.: Improved Delta Debugging Based on Combinatorial Testing, *2012 12th International Conference on Quality Software*, pp.102–105 (2012).
[19] Colbourn, C.J. and McClary, D.W.: Locating and detecting arrays for interaction faults, *Journal of combinatorial optimization*, Vol.15, No.1, pp.17–48 (2008).
[20] Martínez, C., Moura, L., Panario, D. and Stevens, B.: Locating errors using ELAs, covering arrays, and adaptive testing algorithms, *SIAM Journal on Discrete Mathematics*, Vol.23, No.4, pp.1776–1799 (2009).

西浦 生成 （学生会員）

平成 30 年京都工芸繊維大学大学院工芸科学研究科博士前期課程情報工学専攻修了．修士（工学）．同大学大学院工芸科学研究科博士後期課程設計工学専攻在学中．平成 28 年より国立研究開発法人産業技術総合研究所リサーチアシスタント．ソフトウェアのテスト・不具合局所化に関する研究に従事．

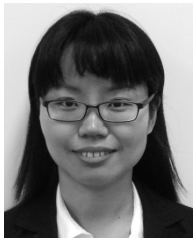


渡辺 大輝

令和 2 年京都工芸繊維大学大学院工芸科学研究科博士前期課程情報工学専攻修了．修士（工学）．ソフトウェアのテスト・不具合局所化に関する研究に従事．

水野 修 （正会員）

平成 11 年大阪大学大学院基礎工学研究科助手．平成 13 年博士（工学）大阪大学．平成 22 年京都工芸繊維大学大学院工芸科学研究科准教授．平成 29 年同情報工学・人間科学系教授．主にソフトウェアのバグ検出手法，ソフトウェアリポジトリのマイニングに関する研究に従事．IEEE 会員．



崔 恩瀞 （正会員）

平成 27 年大阪大学大学院情報科学研究科博士後期課程修了．同年同大学大学院国際公共政策研究科助教．平成 28 年奈良先端科学技術大学院大学情報科学研究科助教．平成 30 年同大学先端科学技術研究科助教（改組による）．同年京都工芸繊維大学情報工学・人間科学系助教．博士（情報科学）．コードクローン管理やリファクタリング支援手法に関する研究に従事．

© 2021 Information Processing Society of Japan